9

you did your strong nine furlong mile in slick and slapstick record time

LDPC and turbo codes

Classes of codes defined on graphs exist that can approach Shannon's capacity bound quite closely, and with a reasonable decoding complexity. All these codes are obtained by connecting simple component codes through an interleaver. Decoding consists of iterative decodings of these simple codes. In this chapter we describe in some detail turbo codes and low-density parity check codes, with special attention to their performance and their decoding algorithms. Their distance properties are also given some attention.

9.1 Low-density parity-check codes

A low-density parity-check (LDPC) code is a long linear block code whose parity-check matrix $\mathbf{H}$ has a low density of ones. Specifically, $\mathbf{H}$ is sparse, i.e., contains a small fixed number w_c of ones in each column and a small fixed number w_r of ones in each row. If the block length is n, we say that $\mathbf{H}$ characterizes an $\langle n, w_c, w_r \rangle$ LDPC code. These codes may be referred to as *regular* LDPC codes to distinguish them from *irregular* codes, whose values of w_c and w_r are not constant. The matrix $\mathbf{H}$ of the latter has *approximately* w_r ones in each row and w_c ones in each column.

The normal graph of a (regular) LDPC code is shown in Fig. 9.1. With this representation, we have that an LDPC code is a binary linear code such that every coded symbol participates in exactly w_c parity-check equations, while each one of the m sum-check equations involves exactly w_r bits. For consistency, we have $nw_c = mw_r$.

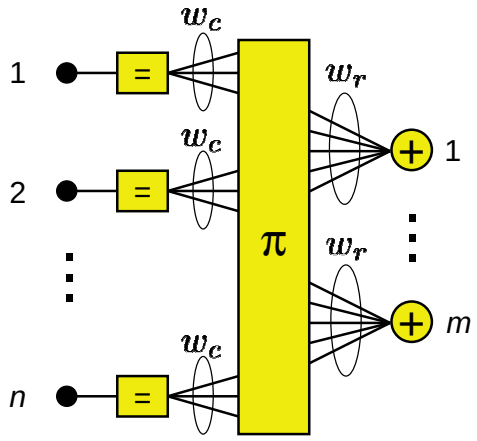

Figure 9.1: *Normal graph of a regular* $\langle n, w_c, w_r \rangle$ *LDPC code.*

It follows from the definition of an LDPC code that $\mathbf{H}$ has nw_c/w_r rows: in fact, the total number of ones in $\mathbf{H}$ is nw_c; dividing by w_r, we obtain the number of rows. Since $\mathbf{H}$ is in general an $m \times n$ matrix, if $\mathbf{H}$ has full rank the code rate is

$$\frac{n-m}{n} = 1 - \frac{m}{n} = 1 - \frac{w_c}{w_r} \tag{9.1}$$

The above equality yields the constraint $w_c \leq w_r$. Notice that the actual rate ρ of the code might be higher than $m/n = w_c/w_r$, because the parity-check equations summarized by $\mathbf{H}$ might not be all independent. We call $\rho^* \triangleq 1 - w_c/w_r$ the *design rate* of the code.

Example 9.1

The parity-check matrix of a $\langle 20, 3, 4\rangle$ LDPC code with $\rho^* = 1/4$ is shown below.

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

In this example we observe that $\mathbf{H}$ can be viewed as composed of three submatrices, each of which contains a single "1" in each column. The second and third submatrices are obtained from the first one by permuting the column order. □

9.1.1 Desirable properties

While the ultimate quality of an LDPC code is defined in terms of its rate, coding gain, and complexity, some simple considerations may guide the selection of a candidate code. First, for good convergence properties of the iterative decoding algorithm, the Tanner graph of the code should have a large girth. In particular, short cycles must be avoided. (Observe that the shortest possible cycle in a bipartite graph has length 4, as shown in Fig. 9.2 along with the structure of the parity-check matrix that generates it.) Next, regularity of the code eases implementation. Finally, for small error probability at high $\mathcal{E}_b/N_0$ on the AWGN channel, the minimum Hamming distance of the code must be large. This is especially interesting, because LDPC codes are known to achieve a large value of $d_{\mathrm{H,min}}$. Roughly speaking, if $w_c > 2$ this grows linearly with the block length n, and hence a large random LDPC code will exhibit a large $d_{\mathrm{H,min}}$ with high probability. More specifically, it has been proved [9.12, 9.18] that, for a large enough block length n, an LDPC code exists with rate $\rho \geq 1 - 1/\lambda$, and minimum distance $d_{\mathrm{H,min}} \geq \delta n$, for any $\delta < 0.5$

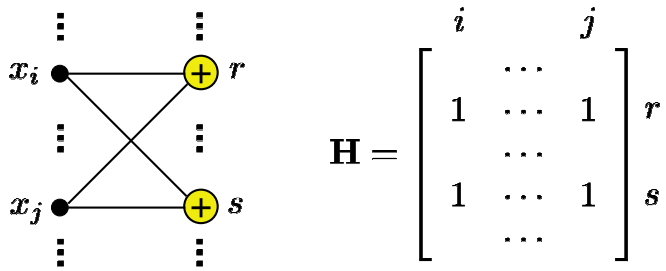

Figure 9.2: *Four-cycle in a Tanner graph, and corresponding parity-check matrix.*

that satisfies the inequality

$$-\delta \log \delta - (1-\delta)\log(1-\delta) < 1/\lambda$$

9.1.2 Constructing LDPC codes

Several techniques for the design of parity-check matrices of LDPC codes have been proposed and analyzed. They can be classified under two main rubrics: *random* and *algebraic* constructions. Here we provide an example of each.

Random constructions

These are based on the generation of a parity-check matrix randomly filled with 0s and 1s, and such that the LDPC properties are satisfied. In particular, after one selects the parameters n, ρ^*, and w_c, for regular codes the row and column weights of $\mathbf{H}$ must be exactly w_r and w_c, respectively, with w_r and w_c small compared to the number of columns and rows. Additional constraints may be included: for example, the number of 1s in common between any two columns (or two rows) should not exceed one (this constraint prevents four-cycles).

In general, randomly constructed codes are good if n is large enough, but their performance may not be satisfactory for intermediate values of n [9.11,9.16]. Also, usually they are not structured enough to allow simple encoding.

A method for the random construction of $\mathbf{H}$ was developed by Gallager in [9.12]. The transpose of the matrix $\mathbf{H}$ of a regular $\langle n, w_c, w_r \rangle$ has the form

$$\mathbf{H}' = \left[\mathbf{H}'_1 \;\; \mathbf{H}'_2 \;\; \cdots \mathbf{H}'_{w_c}\right] \tag{9.2}$$

where $\mathbf{H}_1$ has n columns and n/w_r rows, contains a single 1 in each column, and its ith row contains 1s in columns $(i-1)w_r + 1$ to iw_r. Matrices $\mathbf{H}_2$ to $\mathbf{H}_{w_c}$ are

obtained by randomly permuting (with equal probabilities) the columns of $\mathbf{H}_1$. The matrix $\mathbf{H}$ of Example 9.1 is generated in this way, although there the permutations are not random.

Another algorithm for the generation of the parity-check matrix of an $\langle n, w_c, w_r \rangle$ LDPC code works as follows:

Step 1. Set $i = 1$.

Step 2. Generate a random binary vector with length nw_c/w_r and weight w_c. This is the ith column of $\mathbf{H}$.

Step 3. If the weight of each row of $\mathbf{H}$ at this point is $\leq w_r$, and the scalar product of each pair of columns is ≤ 1 (four-cycle constraint), then set $i = i + 1$. Otherwise, go to Step 2.

Step 4. If $i = n$, then stop. Otherwise, go to Step 2.

Since there is no guarantee that there are exactly w_r 1s in each row of $\mathbf{H}$, this algorithm may generate an irregular code. If a regular code is sought, suitable modifications to the procedure should be made.

Algebraic constructions

Algebraic LDPC codes may lend themselves to easier decoding than random codes. In addition, for intermediate n, the error probability of well-designed codes algebraic codes may be lower [9.1, 9.20].

A simple algebraic construction works as follows [9.10, 9.13]. Choose $p > (w_c - 1)(w_r - 1)$, and consider the $p \times p$ matrix obtained from the identity matrix $\mathbf{I}_p$ by cyclically shifting its rows by one position to the right:

$$\mathbf{J} \triangleq \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

The ℓth power of $\mathbf{J}$ is obtained from $\mathbf{I}_p$ by shifting cyclically its rows by ℓ mod p positions to the right. After defining $\mathbf{J}^0 \triangleq \mathbf{I}_p$, construct the matrix

$$\mathbf{H} = \begin{bmatrix} \mathbf{J}^0 & \mathbf{J}^0 & \mathbf{J}^0 & \cdots & \mathbf{J}^0 \\ \mathbf{J}^0 & \mathbf{J}^1 & \mathbf{J}^2 & \cdots & \mathbf{J}^{w_r-1} \\ \mathbf{J}^0 & \mathbf{J}^2 & \mathbf{J}^4 & \cdots & \mathbf{J}^{2(w_r-1)} \\ & & & \cdots & \\ \mathbf{J}^0 & \mathbf{J}^{(w_c-1)} & \mathbf{J}^{2(w_c-1)} & \cdots & \mathbf{J}^{(w_c-1)(w_r-1)} \end{bmatrix}$$

This matrix has $w_c p$ rows and $w_r p$ columns. The number of 1s in each row and column is exactly w_r and w_c, respectively. Hence, this construction generates a $\langle w_r p, w_c, w_r \rangle$ LDPC code. It can be proved that no 4-cycles are present.

Combining random and algebraic constructions

A technique that combines random and algebraic construction is proposed in [9.20]. Start with the $m \times n$ parity-check matrix $\mathbf{H}(0)$ of a good "core" LDPC code. Next, substitute for each 1 in $\mathbf{H}(0)$ a $p_1 \times p_1$ permutation matrix chosen randomly. We obtain the new $mp_1 \times np_1$ parity-check matrix $\mathbf{H}(1)$. Since the probability of repeating the same permutation matrix in the construction of $\mathbf{H}(1)$ is $1/p_1$!, it is suggested to choose $p_1 \geq 5$. The construction is repeated by substituting for each 1 in $\mathbf{H}(1)$ a $p_2 \times p_2$ random permutation matrix, which yields the $mp_1p_2 \times np_1p_2$ parity-check matrix $\mathbf{H}(2)$. This procedure can be repeated. In [9.20], it is shown that this construction preserves the girth and the minimum Hamming distance of the core code.

9.1.3 Decoding an LDPC code

Decoding can be performed using the sum-product or the max-sum algorithm as indicated in previous chapter. Here, however, since the Tanner graph of the code has cycles, the algorithm is not exact, nor does it converge in a finite number of steps. An iterative algorithm can be devised, which computes alternatively the messages associated with both directions of each branch, and stops according to a preassigned criterion. A possible stopping rule is the following: set $\hat{x}_i = 1$ if $p(x_i = 1 \mid \mathbf{y}) > p(x_i = 0 \mid \mathbf{y})$, and $\hat{x}_i = 0$ otherwise. If the vector $\widehat{\mathbf{x}} \triangleq (\hat{x}_1, \ldots, \hat{x}_n)$ is a code word (i.e., all parity checks are satisfied) then stop. Otherwise, keep on iterating to some maximum number of iterations, then stop and declare a failure.

Fig. 9.3 represents, in a schematic form, the two basic message-passing steps when an iterative version of the sum-product algorithm is used for decoding an LDPC code. We assume here that the messages are normalized so as to represent probabilities, and use a result from Problem 4 of Chapter 8. The algorithm starts with the intrinsic probabilities $\nu_i \triangleq p(y_i|x_i)$, and with uniform messages coming out of check nodes: $\mu_\ell = (0.5, 0.5)$ for all ℓ. Application of the SPA first computes the messages passing from symbol nodes to check nodes, then from check nodes to symbol (repetition) nodes. These two steps represent a single iteration of the SPA.

Fig. 9.4 shows the performance of two LDPC codes.

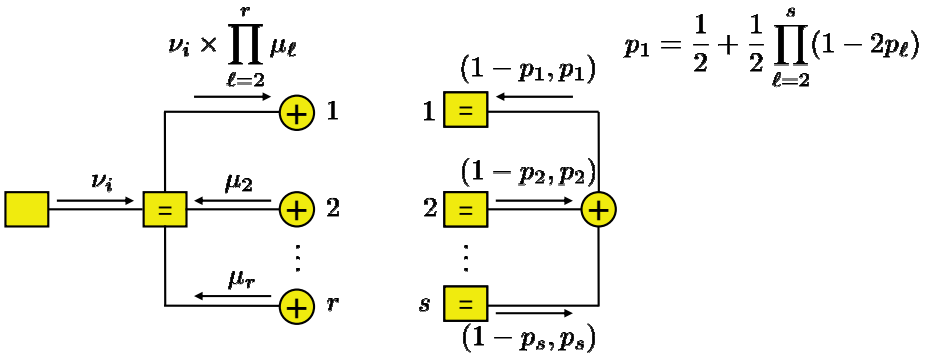

Figure 9.3: *Decoding an LDPC code: message-passing from a symbol node to a check node, and vice versa.*

A simple suboptimum decoding algorithm: bit flipping

An LDPC code can be suboptimally decoded by a simple iterative technique, called the *bit-flipping algorithm*. First, the symbols are individually "hard-decoded" by transforming the channel observations into 1s and 0s, so that the received vector $\mathbf{y}$ is transformed into the binary vector $\mathbf{b}$. Consider the syndrome $\mathbf{Hb}'$, whose components can be seen as the results of the sums computed in the right part of the graph. Each component of $\mathbf{b}$ affects w_c components of the syndrome. Thus, if only one bit is in error, then w_c syndrome components will equal 1. The bit-flipping algorithm is based on this observation, and works as follows. Each iteration step includes the computation of all check sums, and the computation of the number of unsatisfied parity checks involving each one of the n bits of $\mathbf{b}$. Next, the bits of $\mathbf{b}$ are flipped when they are involved in the largest number of unsatisfied parity checks. Steps are repeated until all checks are satisfied, or after a predetermined number of iterations.

Example 9.2

For illustration purposes, consider the rate-$1/3$ code (not exactly an LDPC code, as n is not large enough to yield a sparse $\mathbf{H}$) with parity-check matrix

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

corresponding to the Tanner graph of Fig. 9.5. Let the observed vector be $(.1, .3, -1.2, .02, .5, .9)$. The binary 6-tuple obtained by hard-decoding is (001000). This is not a code word.

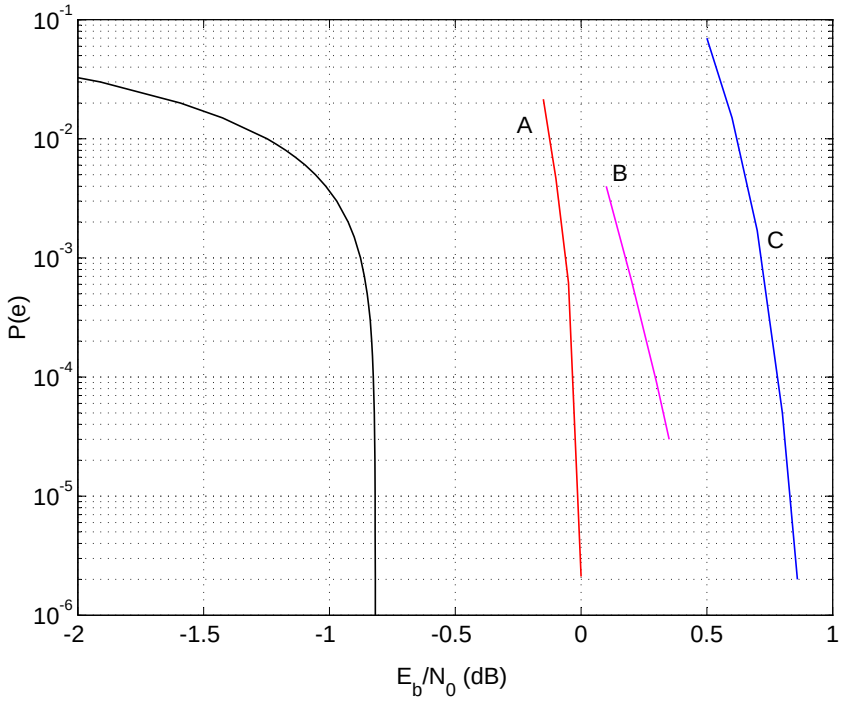

Figure 9.4: *Performance of rate-*$1/4$ *codes. Code B [9.7] is an irregular LDPC code with* $n = 16{,}000$*. Code C [9.18] is a regular LDPC code with* $n = 40{,}000$*. For reference's sake, Code A is a turbo code with* $n = 16{,}384$ *(see Fig. 9.17 for further details). The leftmost curve is the Shannon limit for* $\rho = 1/4$ *and unconstrained AWGN channel, as derived in Problem 8 of Section 3 (see also Fig. 1.5).*

The first iteration shows that the parity checks that fail are 1 and 4, which is interpreted as an error located among the symbols whose nodes are connected to adders 1 and 4. Now, symbol 4 corresponds to no failed check, symbols 1, 2, 5, and 6 correspond to 1 failed check, and symbol 3 to 2 failed checks. We flip the third bit, thus obtaining the code word (000000), which is accepted, as all parity checks are satisfied. □

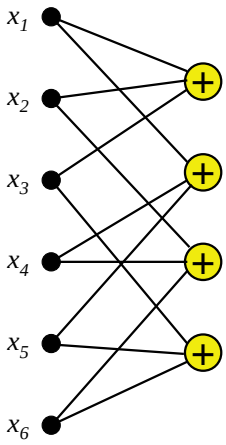

Figure 9.5: *Tanner graph of an LDPC code.*

9.2 Turbo codes

The general scheme of a turbo code based on "parallel concatenation" of two convolutional codes was shown in Fig. 8.14. There, $\mathcal{C}_1$ and $\mathcal{C}_2$ are binary terminated convolutional codes or block codes, realized in systematic form. Let the generator matrices of $\mathcal{C}_1$ and $\mathcal{C}_2$ be $\mathbf{G}_1 = [\mathbf{I} \quad \mathbf{P}_1]$ and $\mathbf{G}_2 = [\mathbf{I} \quad \mathbf{P}_2]$, respectively. If the vector to be encoded is $\mathbf{u}$, the first encoder outputs $[\mathbf{u} \quad \mathbf{c}_1]$, with $\mathbf{c_1} \triangleq \mathbf{u}\mathbf{P}_1$. The interleaver $\boldsymbol{\pi}$ applies a fixed permutation to the components of $\mathbf{u}$, and sends $\boldsymbol{\pi}\mathbf{u}$ to the second encoder, which generates $[\boldsymbol{\pi}\mathbf{u} \quad \mathbf{c}_2]$, with $\mathbf{c}_2 \triangleq (\boldsymbol{\pi}\mathbf{u})\mathbf{P}_2$.

If $\mathcal{C}_1$ and $\mathcal{C}_2$ have rates ρ_1 and ρ_2, respectively, the turbo-code rate is given by

$$\rho = \frac{\rho_1 \rho_2}{\rho_1 + \rho_2 - \rho_1 \rho_2} \tag{9.3}$$

To prove this, neglect the effect of the trellis termination, and observe that if k bits enter the encoder of Fig. 8.14, then $\mathbf{u}$ contains k bits, $\mathbf{c}_1$ contains $k/\rho_1 - k$ bits, and $\mathbf{c}_2$ contains $k/\rho_2 - k$ bits. The ratio between k and the total number of encoded bits yields (9.3). Note that if $\rho_1 = \rho_2$ we simply have

$$\rho = \frac{\rho_1}{2 - \rho_1} \tag{9.4}$$

The most popular turbo-code design has $\rho_1 = \rho_2 = 1/2$ (typically obtained with $\mathcal{C}_1 = \mathcal{C}_2$), and hence $\rho = 1/3$ [9.3, 9.4]. If the even bits of $\mathbf{c}_1$ and the odd bits of $\mathbf{c}_2$ are punctured, then $\rho_1 = \rho_2 = 2/3$, and $\rho = 1/2$.

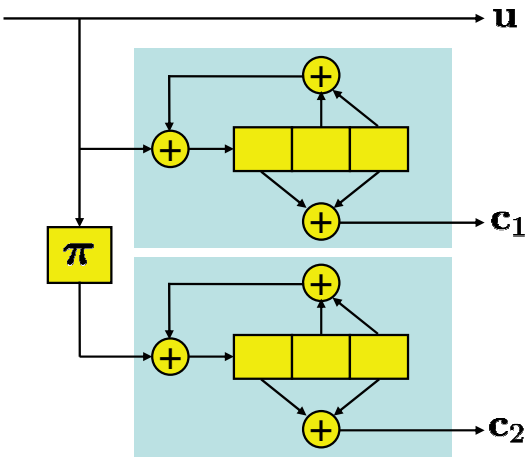

Figure 9.6: *Encoder of a parallel-concatenated turbo code with recursive component encoders, and* $\rho = 1/3$.

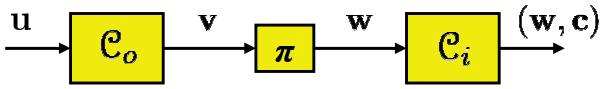

Figure 9.7: *General scheme of a serially-concatenated turbo code.*

The most common form of convolutional encoder used in general is nonsystematic and polynomial (as for example the rate-$1/2$ encoder of Fig. 6.3). Such an encoder cannot be used as a constituent of a turbo code, which requires systematic encoders. Nonrecursive (i.e., feedback-free) encoders should also be ruled out, because the resulting turbo code would exhibit poor distance properties. A turbo encoder including two systematic recursive codes is shown in Fig. 9.6.

Serially-concatenated turbo codes

A serially-concatenated turbo code is obtained by cascading two convolutional encoders as shown in Fig. 9.7. $\mathcal{C}_o$ is called the *outer* code, and $\mathcal{C}_i$ the *inner* code. Their rates are ρ_o and ρ_i, respectively. In practice, the outer code may be chosen as nonrecursive and nonsystematic or recursive and systematic; however, $\mathcal{C}_i$ should be recursive and systematic for better performance. The rate ρ of the concatenated code is simply given by the product of the two rates:

$$\rho = \rho_o \rho_i \tag{9.5}$$

For example, the rate $\rho = 1/2$ can be obtained by choosing two component code $\rho_o = 2/3$ and $\rho_i = 3/4$. Notice that this choice involves constituent codes with higher rates and complexity than for a rate-$1/2$ turbo code with parallel concatenation.

9.2.1 Turbo algorithm

Although, in principle, turbo codes can be optimally decoded by drawing their trellis and using Viterbi algorithm, the complexity of the resulting decoder would be generally prohibitive. Using an iterative version of the sum-product algorithm (the "turbo algorithm") provides instead extremely good performance with moderate complexity. This algorithm is conceptually similar to the message-passing algorithm described for LDPC codes, consisting in iterative exchanges of messages from symbol nodes to check nodes and vice versa (see Fig. 9.3). With turbo codes, the more complex structure of their factor graph (which includes convolutional codes in lieu of symbol nodes and check nodes: see Fig. 8.15) calls for a more complex algorithm. In fact, it requires the separate decoding of the component codes: each decoder operates on the received data, forms an estimate of the transmitted message, and exchanges information with the other decoder. After a number of iterations, this estimate is finally accepted. The algorithm is run for a fixed number of iterations, or can be stopped it as soon as a code word is obtained (see *supra*, Section 9.1.3).

Fig. 9.8 summarizes the general principle, whereby two decoders (one for $\mathcal{C}_1$

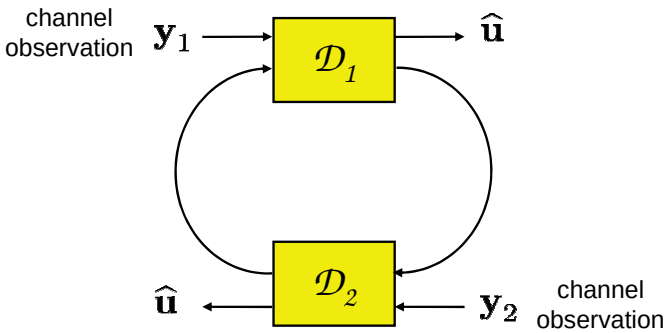

Figure 9.8: *General scheme of turbo decoding algorithm. Here* $\mathbf{y}_1$ *and* $\mathbf{y}_2$ *are channel observations generated by two independent encodings of the same block* $\mathbf{u}$ *.*

and one for $\mathcal{C}_2$) exchange messages back and forth: this decoding mechanism is

reminiscent of the working of a turbo-charged engine, which suggested the name "turbo" for the algorithm. Although relatively little is known about its theoretical convergence properties (which will be examined *infra*, in Section 9.2.4), its behavior with graphs having loops is surprisingly good.

To describe the turbo algorithm, we first examine the behavior of the two decoders of Fig. 9.8, and, in particular, the messages they exchange under the SPA. Consider a linear binary block code $\mathcal{C}$ with length n and k information symbols (if a convolutional code is used, let its termination generate a block code with the above parameters). Here we compute explicitly the a posteriori probabilities of the code symbols, examining separately systematic and a nonsystematic codes.

SISO decoder: Systematic codes

If the code is systematic, the first k entries of each word $\mathbf{x}$ coincide with the information symbols $\mathbf{u}$. We write $\mathbf{x} = (u_1, \dots, u_k, x_{k+1}, \dots, x_n)$, and we have

$$p(\mathbf{x}) = [\mathbf{x} \in \mathcal{C}] \prod_{i=1}^{k} p(u_i)$$

Hence, under our usual assumption of a stationary memoryless channel,

$$p(\mathbf{x} \mid \mathbf{y}) \propto [\mathbf{x} \in \mathcal{C}] \prod_{i=1}^{k} p(y_i \mid u_i) p(u_i) \prod_{j=k+1}^{n} p(y_j \mid x_j) \tag{9.6}$$

To compute the APPs of the information symbols u_i, $i = 1, \dots, k$, (and hence to soft-decode $\mathcal{C}$) we combine, according to (9.6), the "a priori information" $p(u_1), \dots, p(u_k)$ on the source symbols and the "channel information" $p(\mathbf{y} \mid \mathbf{x})$ into one intrinsic message (Fig. 9.9).

To describe the message-passing turbo algorithm it is convenient to introduce a *soft-input, soft-output* (SISO) decoder as shown in Fig. 9.10. This is system which, based on (9.6), accepts two sets of inputs: (a) the n conditional probabilities whose product forms $p(\mathbf{y} \mid \mathbf{x})$, and (b) the k a priori probabilities $p(u_j)$. It outputs (c) the k APPs $p(u_i \mid \mathbf{y})$, and (d) the k extrinsic messages (the "extrinsic information"). This block may be implemented using the BCJR algorithm, or, if this is computationally too intensive, an approximate version of it.

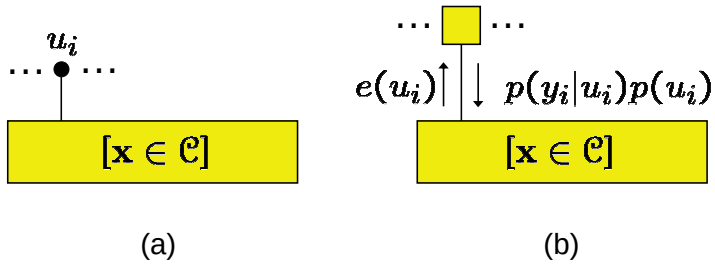

Figure 9.9: *(a) Factor graph for the systematic code $\mathcal{C}$. (b) Messages exchanged by the sum-product algorithm applied to the computation of the APPs $p(u_i|\mathbf{y})$.*

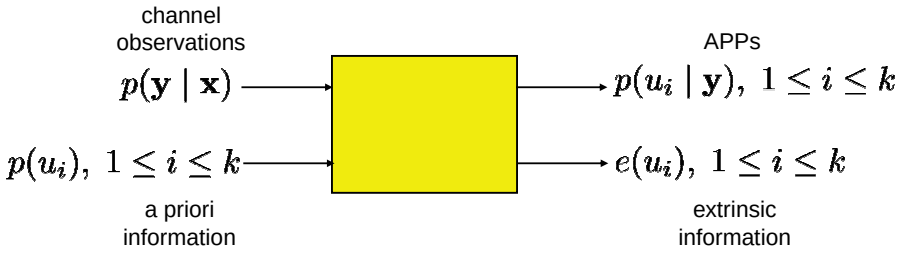

Figure 9.10: *Soft-input, soft-output decoder for systematic codes.*

SISO decoder: Nonsystematic codes

In this case, with the assumption of a stationary memoryless channel, the APP $p(\mathbf{x}|\mathbf{y})$ takes the form

$$p(\mathbf{x} \mid \mathbf{y}) \propto [\mathbf{x} \in \mathcal{C}] \prod_{i=1}^{n} p(y_i \mid x_i) p(x_i) \tag{9.7}$$

This equation implies the assumption that the symbols x_i are all independent, so that $p(\mathbf{x})$ can be factored as the product of individual probabilities $p(x_i)$. A priori, this does not seem to make sense: however, we shall see in the following that in turbo decoding algorithms the role of these probabilities will be taken by the extrinsic messages $e(x_i)$. Since one of the effects of a long interleaver is to make the random variables $e(x_i)$ independent (at least approximately), the above assumption becomes realistic for long enough blocks. The corresponding factor graph is shown in Fig. 9.11, while Fig. 9.12 illustrates the SISO decoder. This is a system accepting two sets of inputs: (a) the n conditional probabilities $p(y_j \mid x_j)$, and (b)

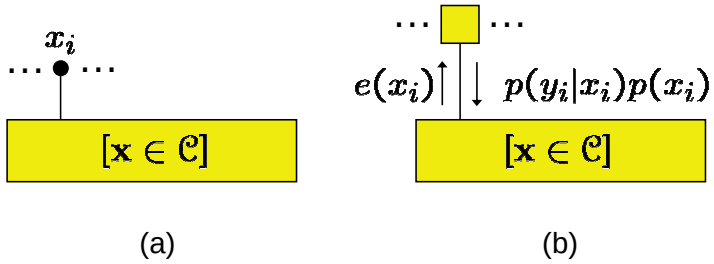

Figure 9.11: *(a) Factor graph for the nonsystematic code $\mathcal{C}$. (b) Messages exchanged by the sum-product algorithm applied to the computation of the APPs $p(x_i|\mathbf{y})$.*

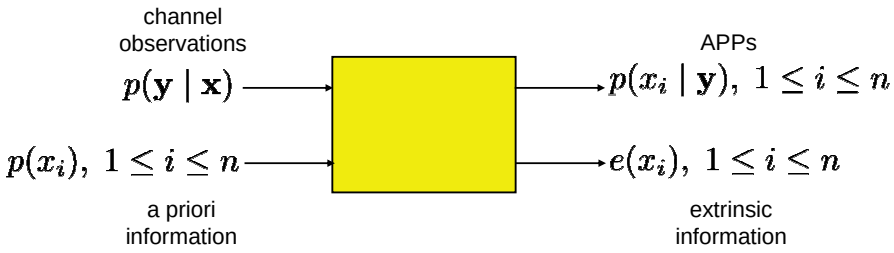

Figure 9.12: *Soft-input, soft-output decoder for nonsystematic codes.*

the n a priori probabilities $p(x_j)$. It outputs (c) the n APPs $p(x_i \mid \mathbf{y})$ and (d) the n extrinsic messages $e(x_i)$. (Notice that the a priori are unknown here.)

Turbo algorithm for parallel concatenation

Having defined SISO decoders, we can now specialize the general iteration scheme of Fig. 9.8. If codes $\mathcal{C}_1$ and $\mathcal{C}_2$ are connected together, they may exchange extrinsic information as suggested in Fig. 9.13. The complete scheme of Fig. 9.14 shows how two SISO decoders combine into the turbo algorithm. The algorithms starts by soft-decoding $\mathcal{C}_1$, which is done by the SISO decoder $\mathcal{D}_1$. At this step, the a priori probabilities of each bit are initialized to $1/2$. The output APPs are not used, while the extrinsic messages, suitably normalized to form probabilities, are used, after permutation, *as a priori probabilities* in $\mathcal{D}_2$, the SISO decoder for $\mathcal{C}_2$. The extrinsic messages at the output of $\mathcal{D}_2$ are permuted, and used as a priori probabilities for $\mathcal{D}_1$. These operations are repeated until a suitable stopping criterion is met. At this point the output APPs are used to hard-decode the information bits. Notice

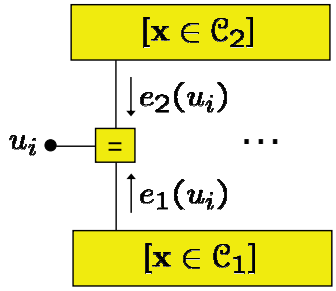

Figure 9.13: *Exchange of extrinsic information between two codes.*

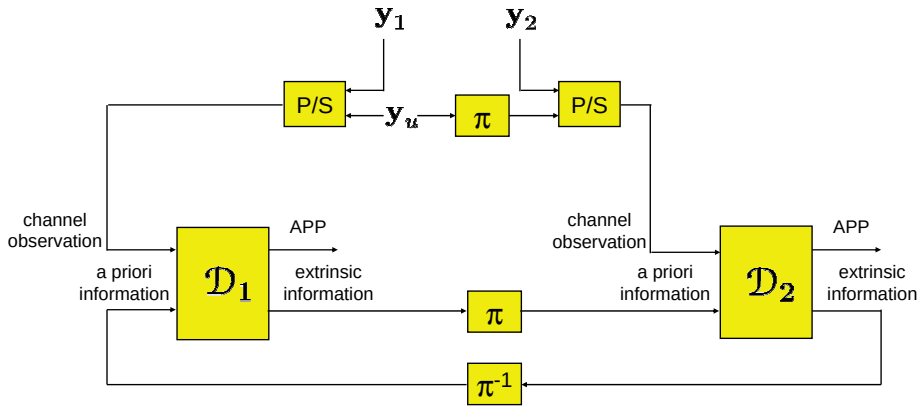

Figure 9.14: *General scheme of an iterative turbo decoder for parallel concatenation. P/S denotes a parallel-to-series converter,* $\mathcal{D}_1$ *and* $\mathcal{D}_2$ *are soft-input, soft-output decoders for code* $\mathcal{C}_1$ *and code* $\mathcal{C}_2$*, respectively,* π *denotes the same interleaver used in the encoder, and* π^{-1} *its inverse.*

that in the iterations the channel information gathered from the observation of $\mathbf{y}_u$, $\mathbf{y}_1$, and $\mathbf{y}_2$ does not change: only the "a priori information" inputs to the decoders vary.

By this algorithm, the operation of the individual SISO decoders is relatively easy, because $\mathcal{C}_1$ and $\mathcal{C}_2$ are weak codes. As such, neither $\mathcal{C}_1$ nor $\mathcal{C}_2$ can individually achieve a high performance. It is their combination that makes for a powerful code, and at the same time allows the decoding task to be split into simpler operations.

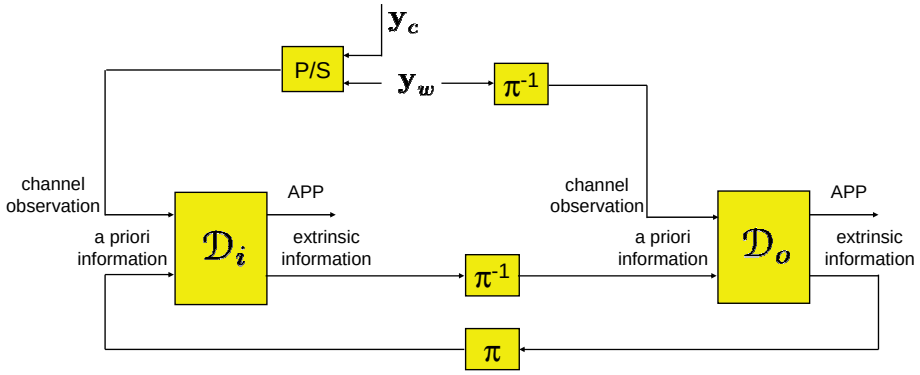

Figure 9.15: *General scheme of an iterative turbo decoder for serial concatenation. P/S denotes a parallel-to-series converter, $\mathcal{D}_i$ and $\mathcal{D}_o$ are soft-input, soft-output decoders for inner code $\mathcal{C}_i$ and outer code $\mathcal{C}_o$, respectively, π denotes the same interleaver used in the encoder, and π^{-1} its inverse.*

Turbo algorithm for serial concatenation

We assume here that the inner code is systematic, while the outer code is nonsystematic. Recalling Fig. 9.7, let $\mathbf{u}$, $\mathbf{v}$ denote input and output of $\mathcal{C}_o$, respectively, $\mathbf{w} \triangleq \boldsymbol{\pi}\mathbf{v}$ the permuted version of $\mathbf{v}$, and $(\mathbf{w}, \mathbf{c})$ the output of $\mathcal{C}_i$. Finally, let $\mathbf{y} = (\mathbf{y}_w, \mathbf{y}_c)$ denote the observed vector. The block diagram of a turbo decoder for serially-concatenated codes is shown in Fig. 9.15. The operation of this decoder is similar to that of Fig. 9.14; however, the two SISO decoders are different here: $\mathcal{D}_i$ has the structure illustrated in Fig. 9.10, while $\mathcal{D}_o$ corresponds to Fig. 9.12.

9.2.2 Convergence properties of the turbo algorithm

Fig. 9.16 shows qualitatively a typical behavior of the bit error rate of an iteratively-decoded turbo code. Three regions can be identified on this chart. In the low-SNR region, the BER decreases very slowly as the iteration order and the SNR increase. For intermediate values of SNR, the BER decreases rapidly as the SNR increases, and improves with increasing the number of iterations. This "waterfall region" is where turbo codes are most useful, as their coding gain approaches the theoretical limit. Finally, for large SNR, an "error floor" effect takes place: the performance

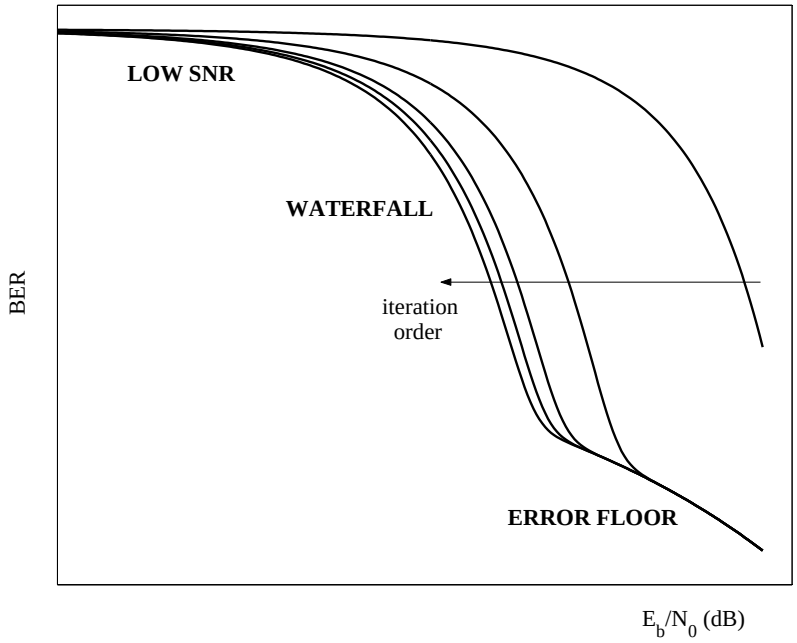

Figure 9.16: *Qualitative aspect of the BER curves vs.* $\mathcal{E}_b/N_0$ *and the number of iterations for turbo codes.*

is dictated by the minimum Hamming distance of the code, the BER-curve slope changes, and the coding gain decreases.[1]

Fig. 9.17 shows the performance of three turbo codes in the waterfall region. Their error probabilities are compared with the Shannon limits for the unconstrained AWGN channel, as derived in Problem 8 of Section 3 (see also Fig. 1.5).

[1]It has been argued [9.15] that the presence of this error floor makes turbo codes not suitable for applications requiring extremely low BERs. Their poor minimum distance, and their natural lack of error-detection capability, due to the fact that in turbo decoding only information bits are decoded (but see [9.26] for an automatic repeat-request scheme based on punctured turbo codes) make these codes perform badly in terms of block error probability. Poor block error performance also makes these codes not suitable for certain applications. Another relevant factor that may guide in the choice of a coding scheme is the decoding delay one should allow: in fact, turbo codes, as well as LDPC codes, suffer from a substantial decoding delay, and hence their application might be more appropriate for data transmission than for real-time speech.

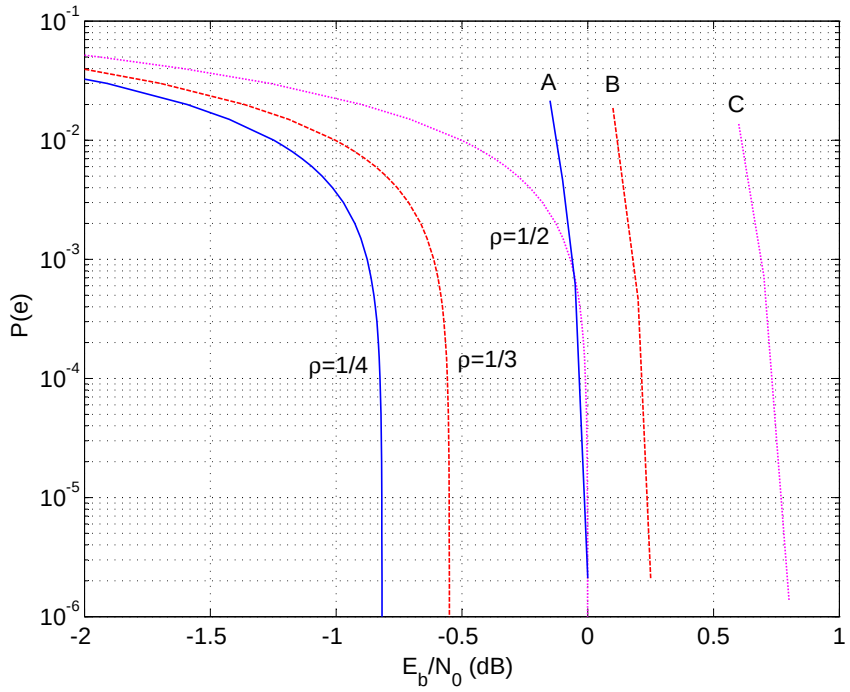

Figure 9.17: *Performance of three turbo codes with block length* $16{,}384$*, obtained by parallel concatenation of two convolutional codes. Code A has* $\rho = 1/4$*,* $16+16$ *states, and is decoded with* 13 *iterations. Code B has* $\rho = 1/3$*,* $16 + 16$ *states, and is decoded with* 11 *iterations. Code C has* $\rho = 1/2$*,* $2 + 32$ *states, and is decoded with* 18 *iterations.*

9.2.3 Distance properties of turbo codes

As just observed, for intermediate SNRs the good performance of turbo codes does not depend on their minimum-distance properties: it is rather affected by their small error coefficient (small number of nearest neighbors) in their low-weight words. For high SNRs, on the other hand, the error probability curve of turbo codes exhibits a "floor," caused by a relatively modest minimum distance.[2]

Let n denote the block length of the component codes (and hence the interleaver length), and B the number of parallel codes ($B = 2$ in all preceding discussions,

[2] An error floor may also occur with LDPC codes, caused by "near-code words," i.e., n-tuples with low Hamming weight whose syndrome has also a low weight. See [9.19].

but we can think of a more general turbo coding scheme). Moreover, let the component codes be recursive. Then, it can be shown [9.14] that the minimum Hamming distance grows like $n^{1-2/B}$. More precisely, given a code $\mathcal{C}$, let $\mathcal{C}(d)$ denote the set of its nonzero words with weight $1, \ldots, d$. If we choose at random a parallel concatenated code $\mathcal{C}$ using B equal recursive convolutional codes and the block length is n, then as $n \to \infty$ we have, for every $\epsilon > 0$,

$$\mathbb{P}\left[\left|\mathcal{C}\left(n^{1-2/B-\epsilon}\right)\right| = 0\right] \to 1 \qquad \text{and} \qquad \mathbb{P}\left[\left|\mathcal{C}\left(n^{1-2/B+\epsilon}\right)\right| = 0\right] \to 0 \quad (9.8)$$

Notice how this result implies that a turbo code with only two parallel branches has a minimum distance that does not grow as any power of n, whereas if three branches are allowed then the growth is $n^{1/3}$.[3]

For serially concatenated codes the minimum-distance behavior is quite different. Let us pick at random a code from an ensemble of serially concatenated turbo codes. Moreover, let d_o denote the free Hamming distance of the outer code. Then, as $n \to \infty$ we have, for every $\epsilon > 0$,

$$\mathbb{P}\left[\left|\mathcal{C}\left(n^{1-2/d_o-\epsilon}\right)\right| = 0\right] \to 1 \qquad \text{and} \qquad \mathbb{P}\left[\left|\mathcal{C}\left(n^{1-2/d_o+\epsilon}\right)\right| = 0\right] \to 0 \quad (9.9)$$

We see that if the outer code has a large d_o we can achieve a growth rate close to linear with n.

9.2.4 EXIT charts

Since the turbo algorithm operates by exchanging extrinsic messages between two SISO decoders, its convergence may be studied by examining how these evolve with iterations. A convenient graphical description of this process is provided by EXIT charts [9.28], which yield quite accurate, albeit not exact, results, especially in the waterfall region of the error-probability curve of turbo codes. An EXIT chart is a graph that illustrates the input-output relationship of a SISO decoder by showing the transformations induced on a single parameter associated with input and output extrinsic probabilities. The upside of EXIT-chart analyses is that only simulation of the behavior of the individual decoders is needed, instead of computer-intensive error counting with the full decoding procedure.

Let us focus on the binary alphabet $\mathcal{X} = \{\pm 1\}$, and assume an AWGN channel, so that the observed signal is

$$y = x + z$$

[3] For $B = 2$, an upper bound to the minimum distance of a turbo code for *all possible* interleavers is derived in [9.5].

with $z \sim \mathcal{N}(0, \sigma_z^2)$. Since

$$p(y \mid x) = \frac{1}{\sqrt{2\pi}\sigma_z} e^{-(y-x)^2/2\sigma_z^2}$$

the *log-likelihood ratio* (LLR)

$$\Lambda(y) \triangleq \ln \frac{p(y|x=+1)}{p(y|x=-1)}$$

takes value

$$\Lambda(y) = \frac{2}{\sigma_z^2}(x+z) \tag{9.10}$$

and hence, given x, Λ is conditionally Gaussian: we write

$$\Lambda(y) \mid x \sim \mathcal{N}\left(\frac{2}{\sigma_z^2}x, \frac{4}{\sigma_z^2}\right) \tag{9.11}$$

In summary, we may say that $\Lambda(y) \mid x \sim \mathcal{N}(\mu, \sigma^2)$, and that it satisfies the "consistency condition" [9.25]

$$\mu = x\sigma^2/2 \tag{9.12}$$

The above allows us to write

$$p(\Lambda \mid x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(\Lambda - x\sigma^2/2)^2/2\sigma^2} \tag{9.13}$$

EXIT-chart techniques are based on the empirical evidence that extrinsic messages, when expressed in the form of log-likelihood ratios, approach a Gaussian distribution satisfying the consistency condition (9.12). In addition, for large block lengths (and hence large interleavers) the messages exchanged remain approximately uncorrelated from the respective channel observations over many iterations [9.28]. Under the Gaussian assumption, the extrinsic messages are characterized by a single parameter, which is commonly and conveniently chosen to be the mutual information exchanged between the LLR and the random variable x (see [9.29] for experiments that justify this choice):

$$I(x;\Lambda) = \frac{1}{2} \sum_{x \in \{\pm 1\}} \int p(\Lambda|x) \log \frac{p(\Lambda|x)}{p(\Lambda)} \, d\Lambda \tag{9.14}$$

with $p(\Lambda) = 0.5[p(\Lambda|x=-1) + p(\Lambda|x=+1)]$ under the assumption that x takes on equally likely values. In particular, if Λ is conditionally Gaussian, and the

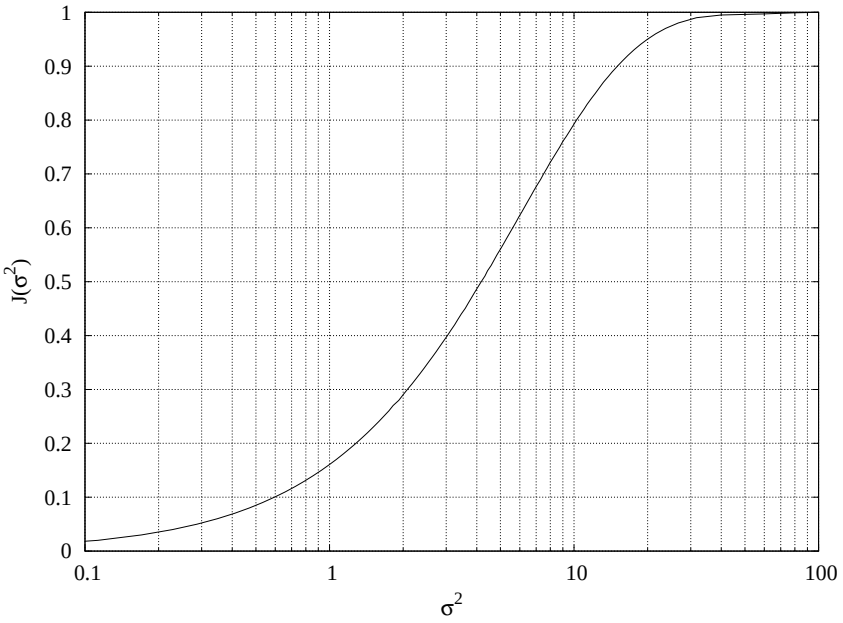

Figure 9.18: *Plot of the function $J(\sigma^2)$ defined in (9.15).*

consistency condition (9.12) is satisfied, then $I(x;\Lambda)$ does not depend on the value of x, and we have explicitly $I(x;\Lambda) = J(\sigma^2)$, where

$$\begin{aligned} J(\sigma^2) &\triangleq 1 - \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-[(w-x\sigma^2/2)^2/2\sigma^2]} \log(1+e^{-xz})\, dw \\ &= 1 - \mathbb{E}\left[\log\left(1-e^{-x\Lambda}\right)\right] \end{aligned} \tag{9.15}$$

where $\mathbb{E}$ is taken with respect to the pdf $\mathcal{N}\left(x\sigma^2/2, \sigma^2\right)$. The function $J(\sigma^2)$ (plotted in Fig. 9.18) is monotonically increasing, and takes values from 0 (for $\sigma \to 0$) to 1 (for $\sigma \to \infty$). If the assumption of conditional Gaussianity on Λ is not made, a convenient approximation of $I(x;\Lambda)$, based on the observation of N samples of the random variable Λ, is based on (9.15):

$$I(x;\Lambda) \approx 1 - \frac{1}{N}\sum_{i=1}^{N} \log\left(1+\exp(-x_i\Lambda_i)\right) \tag{9.16}$$

Recall now that we have four different messages at the input and output of a SISO decoder: a priori, channel observation, soft-decision, and extrinsic. we denote these messages by μ^{a}, μ^{o}, μ^{d}, and μ^{e}, respectively, and by I^{a}, I^{o}, I^{d}, and I^{e},

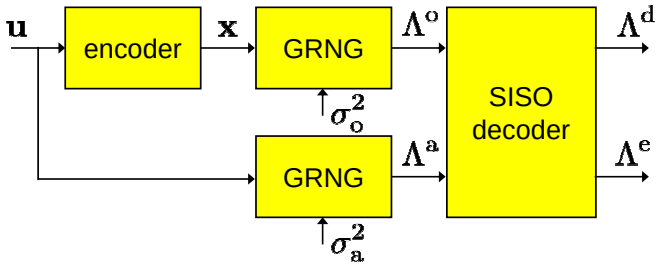

Figure 9.19: *Computing the transfer function T. GRNG is a Gaussian random noise generator.*

respectively, the mutual informations exchanged between their LLRs and x. We describe the behavior of a SISO processor used in iterative decoding by giving its *extrinsic information transfer* (EXIT) function

$$I^{\mathrm{e}} = T(I^{\mathrm{a}}, I^{\mathrm{o}}) \tag{9.17}$$

Fig. 9.19 schematizes the Monte Carlo derivation of the EXIT chart for a given code. Choose first the values of I^{a} and I^{o}. The random vector $\mathbf{u}$ of uncoded ± 1 symbols is encoded to generate the vector $\mathbf{x}$. A Gaussian random noise generator outputs, for each component x of $\mathbf{x}$, the LLR Λ^{o} such that

$$\Lambda^{\mathrm{o}} | x \sim \mathcal{N}\left(x\frac{\sigma_{\mathrm{o}}^2}{2}, \sigma_{\mathrm{o}}^2\right)$$

where $\sigma_{\mathrm{o}}^2 = J^{-1}(I^{\mathrm{o}})$. Similarly, another Gaussian random noise generator outputs, for each component u of $\mathbf{u}$, the LLR Λ^{a} such that

$$\Lambda^{\mathrm{a}} | u \sim \mathcal{N}\left(u\frac{\sigma_{\mathrm{a}}^2}{2}, \sigma_{\mathrm{a}}^2\right)$$

where $\sigma_{\mathrm{a}}^2 = J^{-1}(I^{\mathrm{a}})$. These two LLRs correspond to messages entering the SISO decoder. This outputs the LLRs Λ^{d} and Λ^{e}. Only the latter is retained, and N values of it are used to approximate I^{e} through (9.16), so that no Gaussian assumption is imposed on Λ^{e}.

Once the transfer functions of both decoders have been obtained, they are drawn on a single chart. Since the output of a decoder is the input of the other one, the second transfer functions is drawn after swapping the axes, as shown in the example of Fig. 9.20 (here the two decoders are equal). The behavior of the iterative decoding algorithm is described by a trajectory, i.e., a sequence of moves, along

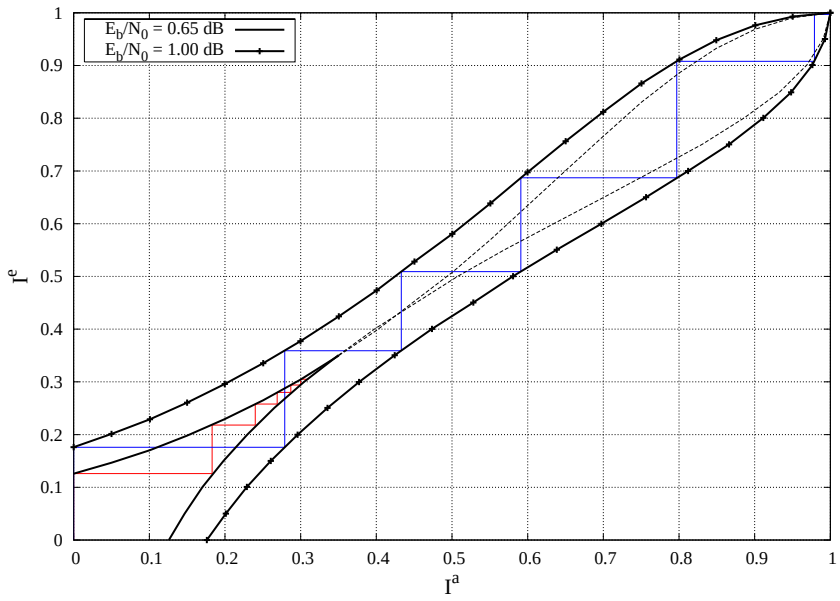

Figure 9.20: *EXIT chart for a rate-*$1/2$ *convolutional code and two values of* $\mathcal{E}_b/N_0$.

horizontal and vertical steps, through the pair of transfer functions. Iterations start with no a priori knowledge, so that $I^{\mathrm{a}} = 0$. Due to the channel observations, the corresponding value of I^{e} at the output of the first SISO decoder increases with $\mathcal{E}_b/N_0$. The resulting extrinsic message if fed to the second decoder, which corresponds to moving along a horizontal line joining the two transfer functions. We thus obtain the value of I^{e} at the output of the second decoder. The corresponding message is fed back to the first decoder, whose output yields the value of I^{e} obtained by joining the two curves with a vertical line, and so on.

Fig. 9.20 shows two examples of convergence behavior. For $\mathcal{E}_b/N_0 = 0.65$ dB, the two curves intersect, the trajectory is blocked, and we experience no convergence to large values of mutual information (which correspond to small error probabilities). For $\mathcal{E}_b/N_0 = 1$ dB, instead, we have convergence.

Estimates of the error probability of a coded system can be superimposed to EXIT charts to offer some extra insight in the performance of the iterative decoder. If the LLR Λ^{d} is assumed to be conditionally Gaussian, with mean $\mu_{\mathrm{d}} = x\sigma_{\mathrm{d}}^2/2$

and variance $\sigma_{\rm d}^2$, the bit error rate (BER) can be approximated in the form

$$P_b(e) = \mathbb{P}(\Lambda^{\rm d} > 0 \mid x = -1) \approx Q\left(\frac{|\mu_{\rm d}|}{\sigma_{\rm d}}\right) = Q\left(\frac{\sigma_{\rm d}}{2}\right) \tag{9.18}$$

Since $\Lambda^{\rm d} = \Lambda^{\rm o} + \Lambda^{\rm a} + \Lambda^{\rm e}$, the assumption of independent LLR's leads to

$$\sigma_{\rm d}^2 = \sigma_{\rm o}^2 + \sigma_{\rm a}^2 + \sigma_{\rm e}^2 \,,$$

which in turn yields

$$P_b(e) \approx Q\left(\frac{\sqrt{J^{-1}(I^{\rm o}) + J^{-1}(I^{\rm a}) + J^{-1}(I^{\rm e})}}{2}\right) \tag{9.19}$$

Notice that, due to (9.11), we have

$$\sigma_{\rm o}^2 = \frac{4}{\sigma_z^2} = 4\,\text{SNR} = 8\rho\frac{\mathcal{E}_b}{N_0}$$

where ρ is the rate of the concatenated code. Fig. 9.21 superimposes the EXIT chart corresponding to $\mathcal{E}_b/N_0 = 1$ dB to a set of constant-BER curves. Comparison of this Figure with Fig. 9.22, obtained by Monte Carlo simulation, shows a good match between the "true" BERs and those predicted by EXIT charts. In addition, observing the evolution of the constant-BER curves, one can observe how traversing the "bottleneck" region between the two curves corresponds to a slow convergence of the BER. Once the bottleneck is passed, faster convergence of BER is achieved.

Accuracy of EXIT-chart convergence analysis

In the upper portion of the EXIT chart, extrinsic messages become increasingly correlated, and the true evolution of $I^{\rm e}$ deviates from the behavior predicted by the chart. As correlations depend also on the interleaver size, it is expected that EXIT analyses become more accurate as it increases.

9.3 Bibliographical notes

Low-density parity-check codes were introduced by Gallager in his doctoral thesis [9.12], and rediscovered in the mid-90s [9.18]. Ref. [9.21] reviews techniques for constructing LDPC codes whose graph have large girth. LDPC decoding algorithms are analyzed in [9.12, 9.18]. LDPC codes over nonbinary alphabets are

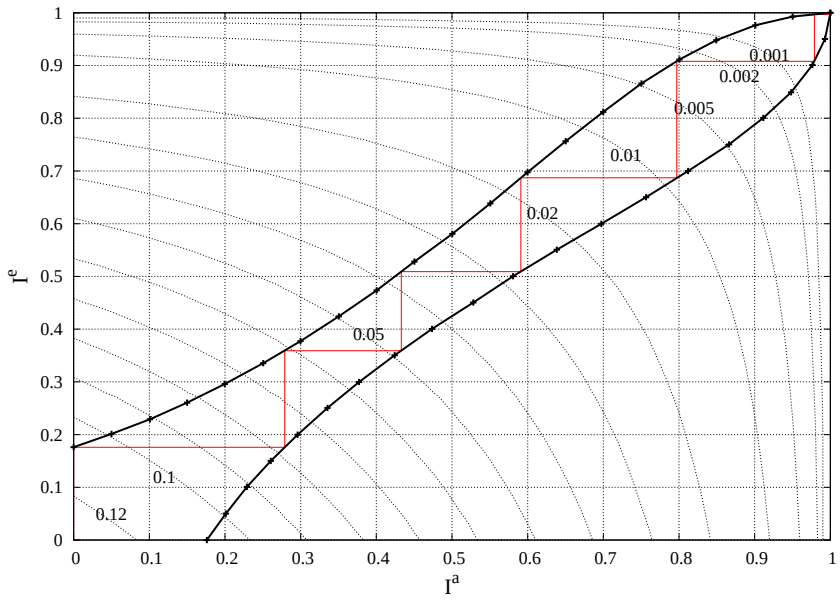

Figure 9.21: *EXIT chart as in Fig. 9.20, for* $\mathcal{E}_b/N_0 = 1$ *dB, superimposed to constant-BER curves.*

examined in [9.8]. Turbo codes, and their iterative decoding algorithm, were first presented to the scientific community in [9.4]. The iterative ("turbo") decoding algorithm was shown in [9.17] to be an instance of J. Pearl's "belief propagation" in graphs [9.22]. Our presentation of SISO decoders follows [9.23].

The capacity-approaching codes described in this chapter are now finding their way into a number of practical applications, ranging from UMTS to wireless local-area networks, deep-space communication, and digital video broadcasting. A list of practical implementations of LDPC codes can be found in [9.24].

Richardson and Urbanke [9.6] have introduced the study of the evolution of the probability distribution of the exchanged messages as a tool to study the convergence behavior of turbo algorithms. EXIT charts, which characterize these distributions using a single parameter, were advocated in [9.28]. Application of EXIT charts to LDPC codes, a topic not considered here, is described in [9.2].

Computation of bounds to the error probability of turbo codes can be found in [9.9, 9.27].

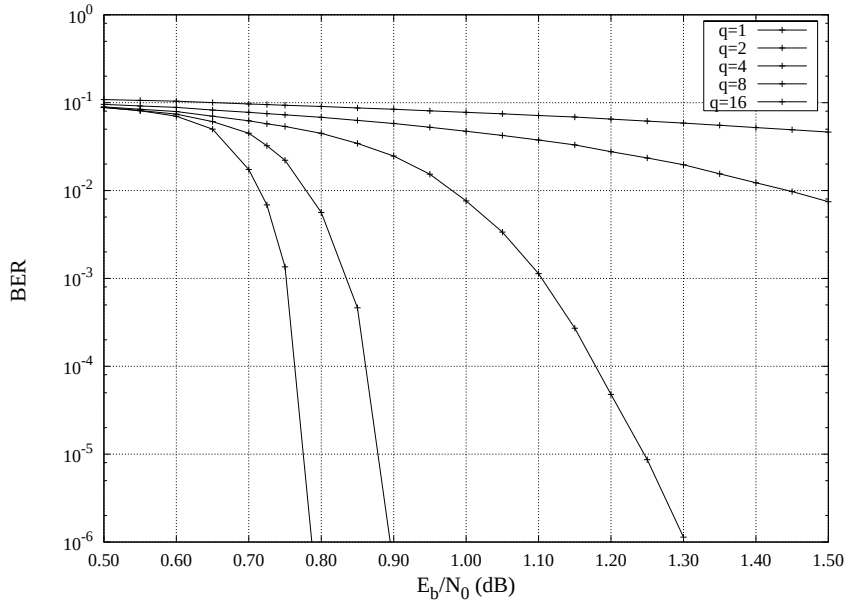

Figure 9.22: *Convergence of a turbo code based on two equal convolutional codes as in Fig. 9.20 (simulation results).*

9.4 Problems

1. Once the matrix $\mathbf{H}$ of an LDPC code is selected, show how the generator matrix $\mathbf{G}$ can be obtained. Consider separately the cases of $\mathbf{H}$ having or not having full rank. Is $\mathbf{G}$ a sparse matrix?

2. Derive EXIT charts for some simple convolutional codes assuming $I^{\mathrm{a}} = 0$. Interpret the shape of the functions.

3. Extend the EXIT-chart analysis to the frequency-flat, slow independent Rayleigh fading channel.

References

[9.1] B. Ammar, B. Honary, Y. Kou, J. Xu, and S. Lin, "Construction of low-density parity-check codes based on balanced incomplete block designs," *IEEE Trans. Inform. Theory*, Vol. 50, No. 6, pp. 1257–1268, June 2004.

[9.2] M. Ardakani and F. R. Kschischang, "Designing irregular LPDC codes using EXIT charts based on message error rate," *Proc. 2002 IEEE Int. Symp. Inform. Theory*, Lausanne, Switzerland, p. 454, June 30–July 5, 2002.

[9.3] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: Turbo codes," *IEEE Trans. Commun.*, Vol. 44, No. 10, pp. 1261–1271, October 1996.

[9.4] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding: Turbo codes," *Proc. IEEE 1993 Int. Conf. Commun. (ICC93)*, Geneva, Switzerland, pp. 1064–1070, May 1993.

[9.5] M. Breiling, J. B. Huber, "Combinatorial analysis of the minimum distance of turbo codes," *IEEE Trans. Inform. Theory*, Vol. 47, No. 7, pp. 2737–2750, Nov. 2001.

[9.6] S.-Y. Chung, T. J. Richardson, and R. L. Urbanke, "Analysis of sum-product decoding of low-density parity-check codes using a Gaussian approximation," *IEEE Trans. Inform. Theory*, Vol. 47, No. 2, pp. 657–670, Feb. 2001.

[9.7] M. C. Davey, *Error-correction using Low-Density Parity-Check Codes*. PhD Dissertation, Univ. of Cambridge, Dec. 1999.

[9.8] M. C. Davey and D. J. C. MacKay, "Low density parity chack codes over GF(q)," *IEEE Comm. Lett.*, Vol. 2, No. 6, pp. 165–167, June 1998.

[9.9] D. Divsalar and E. Biglieri, "Upper bounds to error probabilities of coded systems beyond the cutoff rate," *IEEE Trans. Commun.*, Vol. 51, No. 12, pp. 2011–2018, Dec. 2003.

[9.10] J. L. Fan, "Array codes as low-density parity-check codes," *Proc. 2nd Int. Symp. on Turbo Codes & Related Topics*, Brest, France, pp. 543–546, Sept. 4–7, 2000.

[9.11] M. P. C. Fossorier, "Quasi-cyclic low-density parity-check codes from circulant permutation matrices," *IEEE Trans. Inform. Theory*, Vol. 50, No. 8, pp. 1788–1793, Aug. 2004.

[9.12] R. G. Gallager, *Low-density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.

[9.13] B. Honary *et al.*, "On construction of low density parity check codes," *2nd Int. Workshop on Signal Processing for Wireless Communications (SPWC 2004)*, London, U.K., June 2–4, 2004.

[9.14] N. Kahale and R. Urbanke, "On the minimum distance of parallel and serially concatenated codes," *Proc. IEEE ISIT 1998*, Cambridge, MA, p. 31, Aug. 16–21, 1998.

[9.15] Y. Kou, S. Lin, and M. P. C. Fossorier, "Low-density parity-check codes based on finite geometries: A rediscovery and new results," *IEEE Trans. Inform. Theory*, Vol. 47, No. 7, pp. 2711–2736, November 2001.

[9.16] R. Lucas, M. P. C. Fossorier, Y. Kou, and S. Lin, "Iterative decoding of one-step majority logic decodable codes based on belief propagation," *IEEE Trans. Commun.*, Vol. 48, No. 6, pp. 931–937, June 2000.

[9.17] R. J. McEliece, D. J. C. MacKay, and J.-F. Cheng, "Turbo decoding as an instance of Pearl's "belief propagation" algorithm," *IEEE Journal Select. Areas Commun.*, Vol. 16, No. 2, pp. 140–152, February 1998.

[9.18] D. J. C. MacKay, "Good error correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, Vol. 45, No. 2, pp. 399–431, March 1999. (See also Errata, *ibidem*, Vol. 47, No. 5, p. 2101, July 2001.)

[9.19] D. J. C. MacKay and M. S. Postol, "Weaknesses of Margulis and Ramanujan-Margulis low-density parity-check codes," *Electronic Notes in Theoretical Computer Science*, Vol. 74, pp. 1–8, 2003.

[9.20] N. Miladinovic and M. Fossorier, "Systematic recursive construction of LDPC codes," *IEEE Commun. Letters*, Vol. 8, No. 5, pp. 302–304, May 2004.

[9.21] J. M. F. Moura, J. Lu, and H. Zhang, "Structured low-density parity-check codes," *IEEE Signal Processing Magazine*, Vol. 21, No. 1, pp. 42–55, Jan. 2004.

[9.22] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* San Francisco, CA: Morgan Kaufmann, 1988.

[9.23] O. Pothier, *Codes Composites Construits à Partir de Graphes et Leur Décodage Itératif*. Ph.D. Dissertation, École Nationale Supérieure des Télécommunications, Paris, France, Jan. 2000.

[9.24] T. Richardson and R. Urbanke, "The renaissance of Gallager's low-density parity-check codes," *IEEE Commun. Magazine*, Vol. 41, No. 8, pp. 126–131, Aug. 2003.

[9.25] T. Richardson, A. Shokrollahi, and R. Urbanke, "Design of provably good low-density parity-check codes," *Proc. 2000 IEEE Int. Symp. Inform. Theory (ISIT 2000)*, Sorrento, Italy, p. 199, June 25–30, 2000.

[9.26] D. N. Rowitch and L. B. Milstein, "On the performance of hybrid FEC/ARQ systems using rate compatible punctured turbo (RCPT) codes," *IEEE Trans. Commun.*, Vol. 48, No. 6, pp. 948–959, June 2000.

[9.27] I. Sason and S. Shamai (Shitz), "On improved bounds on the decoding error probability of block codes over interleaved fading channels, with applications to turbo-like codes," *IEEE Trans. Inform. Theory*, Vol. 47, No. 6, pp. 2275–2299, Sept. 2001.

[9.28] S. ten Brink, "Convergence behavior of iteratively decoded parallel concatenated codes," *IEEE Trans. Commun.*, Vol. 49, No. 10, pp. 1727–1737, Oct. 2001.

[9.29] M. Tüchler, S. ten Brink, and J. Hagenauer, "Measures for tracing convergence of iterative decoding algorithms," *Proc. 4th IEEE/ITG Conf. on Source and Channel Coding*, Berlin, Germany, pp. 53–60, Jan. 2002.