1

Coding

Gerhard Bauch,[1] Claude Berrou,[2] David Declercq,[3] Alexandre Graell I Amat,[2] Youssouf Ould-Cheikh-Mouhamedou,[4] Yannick Saouter,[2] Jossy Sayir,[5] and Marcos B.S. Tavares[6]

[1] *Universität der Bundeswehr Munich, Germany*
[2] *Telecom Bretagne, France*
[3] *ETIS ENSEA/Université de Cergy-Pontoise/CNRS, France*
[4] *King Saud University, Saudi Arabia (formerly with Telecom Bretagne, France)*
[5] *Cambridge University, United Kingdom*
[6] *Technische Universität Dresden, Vodafone Chair, Germany*

1.1 General Code Types

The most important coding schemes that can be decoded using an iterative (turbo) algorithm can be classified as parallel concatenated codes, serial concatenated codes and low-density parity check (LDPC) codes as indicated in Figure 1.1.

In parallel concatenated codes, the data sequence is encoded by the first constituent encoder. The second constituent encoder encodes an interleaved version of the data sequence. The data bits are sent only once as systematic bits of the concatenated code, whereas only the parity bits of the constituent encoders are transmitted. Usually, recursive systematic convolutional codes are used as constituent codes. However, other code types, for example block codes, can be used and more than two constituent codes can be concatenated with different interleavers. Parallel concatenated convolutional codes (PCCC) are usually referred to as "turbo codes."

In serial concatenated codes, the second (inner) constituent code encodes the interleaved code bits of the first (outer) constituent code. Convolutional codes are

Error Control Coding for B3G/4G Wireless Systems:Paving the Way to IMT-Advanced Standards, First Edition. Edited by Thierry Lestable and Moshe Ran. Published 2011 by John Wiley & Sons, Ltd.

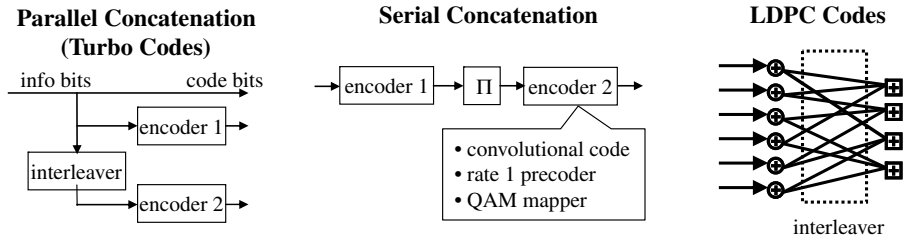

Figure 1.1 Coding schemes with iterative decoding

the most common constituent codes for serial concatenated coding schemes. However, this scheme can be generalized if we consider other components in the transmission chain as an inner encoder, for example the mapper of a QAM modulation scheme, the ISI/MIMO channel, a rate-1 precoder and the like.

Low-density parity check codes are block codes, where the codeword is generated by multiplying the data sequence $\mathbf{d} = [d_1, d_2, \ldots, d_N]^T$ with a generator matrix $\mathbf{G}$. The code is defined by a sparse parity check matrix $\mathbf{H}$, which satisfies $\mathbf{HG} = \mathbf{0}$. These LDPC codes are often represented by their Tanner graph as indicated on the right-hand side of Figure 1.1. The nodes on the left-hand side are called variable nodes. Each of them represents a code bit. The nodes on the right-hand side are called check nodes and represent the parity check equations. A connection between variable node i and check node j exists in the graph if the element h_{ji} in the parity check matrix $\mathbf{H}$ is 1. The modulo 2 check sum of all variable nodes that are connected to the same check node is 0.

Low-density parity check codes were invented in 1962 [Gal62]. They have attracted attention again more recently in the context of iterative decoding because the so-called message passing decoding algorithm can be viewed as iterative decoding between check nodes and variable nodes as constituent decoders. One main reason why LDPC codes have become so popular is that they allow parallel implementation to a great extent. While the trellis decoder for a convolutional code needs a backward and forward recursion through the trellis, all decoding operations in the variable nodes and the check nodes, respectively, can in principle be done in parallel. This allows a decoder implementation with high throughput as required in future wireless systems.

A disadvantage of LDPC codes is that, in general, the encoding complexity grows quadratically with the block size while the encoding complexity of convolutional codes grows only linearly with the block size. However, with structured LDPC codes as discussed in Section 1.3.2.3, the encoding complexity can be greatly reduced.

Many block codes can be regarded as special cases of LDPC codes. We will explain repeat accumulate codes as one example later in this section. Further variants are discussed in Section 1.3.2.3.

The three classes of coding schemes in Figure 1.1 have as a common feature the fact that they can be decoded by an iterative (turbo) decoding scheme as indicated in Figure 1.2.

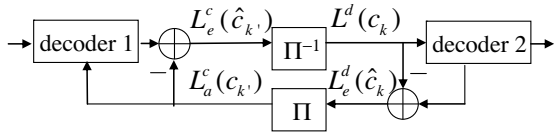

Figure 1.2 Iterative decoding

The received data serve as input to the decoder of the first constituent code, which produces soft a posteriori information – decisions plus reliability information. The output of decoder 1 is used by the second constituent decoder as a priori information on top of the received channel information. After both constituent codes have been decoded once, the output of the second constituent decoder is fed back to decoder 1 and used as additional a priori information in a further decoding step. Several decoding iterations can be performed in this way in order to lower the error rate. It is essential that only extrinsic information of the other constituent decoder is used in order to avoid multiple use of the same information.

Extrinsic information on a bit is the information which can be obtained from all the other bits in the block based on the code constraints. It is the part of the a posteriori information that is newly generated in the current decoding step. Using soft information in the form of log-likelihood ratios

$$L(d) = \log \frac{P(d=0)}{P(d=1)} \tag{1.1}$$

extrinsic information is obtained by bitwise subtraction of the input log-likelihood ratios from the output log-likelihood ratio. Usually, LDPC codes require a significantly larger number of iterations than PCCC or SCCC.

In the following we elaborate further on serial concatenated convolutional codes (SCCC) and compare them to parallel concatenated convolutional codes (PCCC) in terms of performance and complexity.

Simply put, the bit error rate (BER) performance of iterative decoders is characterized by three regions as indicated in Figure 1.3.

With a very low signal-to-noise ratio (SNR), iterative decoding cannot achieve a reasonable error rate. At a SNR where iterative decoding starts to become effective, the BER curve decreases with very steep slope. We call this area in the BER plot the waterfall region. At higher SNR we observe an error floor, which is determined by codewords with small Hamming distance. The error floor can be reduced by proper interleaver design. Simply put, parallel concatenated convolutional codes tend to converge at lower SNR – the waterfall region starts at lower SNR. On the other hand, serial concatenated convolutional codes tend to show a lower error floor. Consequently, serial concatenated coding schemes are more suited for applications which require very low BER whereas parallel concatenated codes are more suitable for

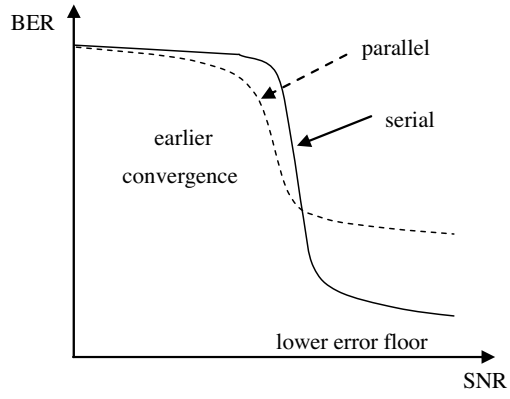

Figure 1.3 Bit error rate of PCCC and SCCC

applications that can handle a certain error rate, for example by means of ARQ or error concealment.

As an example for serial concatenated convolutional codes, we consider a proposal for an ESA telemetry standard where very low error probability is required [B + 05].

The main reason why we are interested in this scheme is a comparison with parallel concatenated convolutional codes and LDPC codes given in [B + 05]. Here, it is claimed that the serial concatenated scheme shows competitive performance with significantly lower complexity.

The encoder is depicted in Figure 1.4. A rate 1/2 convolutional code with memory 2 is used for both inner and outer code. The outer code is punctured to rate 2/3 such that the total code rate is $R = 1/3$. Other code rates for the concatenated scheme can be obtained by puncturing at the output of the inner encoder. However, it turns out that the puncturing pattern has to take the interleaver mapping into account for good performance. The systematic bits of the inner encoder are identical to the codebits of the outer encoder. Hence, we apply the puncturing pattern to the deinterleaved version of those systematic bits of the inner encoder in order to avoid puncturing patterns that cause poor performance of the outer code. In particular, no systematic bits of the outer code should be punctured. Table 1.1 gives permeability rates for puncturing of systematic and parity bits of the inner code, where ρ_s is the fraction of systematic bits which is

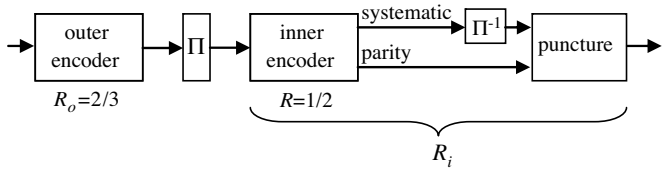

Figure 1.4 Serial concatenated convolutional code (SCCC)

Table 1.1 Permeability rates for SCCC

R	R_o	R_i	ρ_s	ρ_p
1/3	2/3	1/2	1	1
1/2	2/3	3/4	21/30	19/30
2/3	2/3	1	21/30	9/30
4/5	2/3	6/5	21/30	4/30
8/9	2/3	12/9	21/30	1/20

transmitted at the respective code rate, ρ_p is the fraction of parity bits that are transmitted.

Performance curves are depicted in Figures 1.5 and 1.6 for QPSK modulation in an AWGN channel. For the PCCC we use the UMTS turbo code, which has constituent codes with memory 3. Hence, the decoding complexity per iteration is significantly higher than for the considered SCCC.

In Figure 1.5, the required SNR per bit for a target block error rate (BLER) of 10^{-2} is shown versus the number of iterations. It can be concluded that PCCC needs fewer iterations in order to achieve the target BLER at a given SNR. For example, the BLER of PCCC after four iterations is smaller than the BLER of SCCC after eight iterations. Hence, we can reduce the complexity of PCCC by reducing the number of iterations by half and still obtain better BLER than with SCCC.

Figure 1.6 compares the BLER performance of serial concatenated convolutional codes with different memory for the constituent codes. It can be concluded that serial concatenated codes should be run with relatively simple constituent codes. Increasing

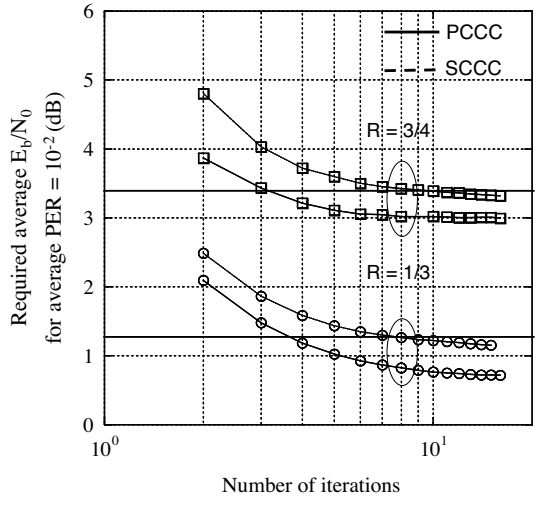

Figure 1.5 Block error rate (BLER) for PCCC and SCCC versus number of iterations

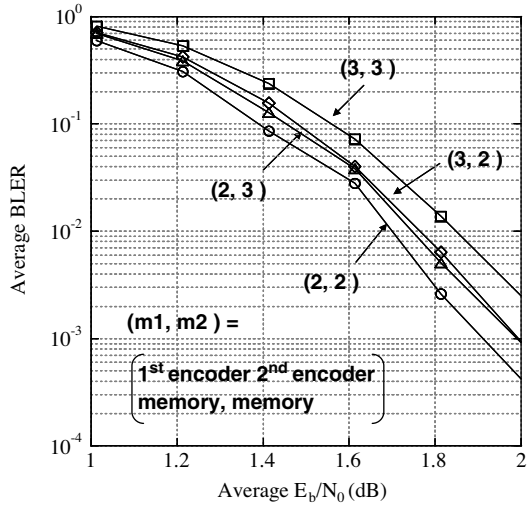

Figure 1.6 Block error rate (BLER) for SCCC with constituent codes with different memory

the memory of the constituent codes does not improve the performance, or results in worse performance.

Figure 1.7 gives a performance comparison between PCCC with constituent codes of different memory and SCCC with memory 2 constituent codes. It can be concluded that PCCC outperforms SCCC by 0.3–0.6 dB at a BLER of 10^{-2}.

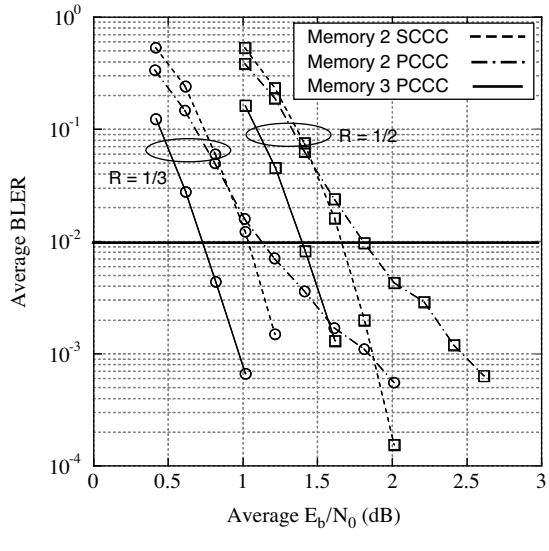

Figure 1.7 Block error rate (BLER) of PCCC and SCCC

For codes with comparable complexity, in other words memory 2 constituent codes in both PCCC and SCCC, PCCC shows a significantly higher error floor. In order to lower this error floor, constituent codes with higher memory have to be used, for example memory 3 as applied in the UMTS turbo code. SCCC with memory 2 constituent codes is about 49% less complex in terms of number of operations than PCCC with memory 3 constituent codes.

1.2 Designing Codes Based on Graphs

The design of codes based on graphs can be understood as a multivariable multiconstraint optimization problem. The constraints of this problem are the performance requirements, flexibility (block lengths, rates, and so forth) and encoding/decoding complexity. The latter also includes the complexity of hardware realizations of the encoder and decoder, as well as latency issues. Figure 1.8 shows the constraints and variables involved in the design of codes defined on graphs. Some of the typical variables that can be optimized to meet the specified constraints are represented as circles in Figure 1.8.

One of the first decisions that should be taken when designing codes defined on graphs is whether the graphs should have a pseudorandom or an algebraic or a combinatorial underlying structure. These different structures have their advantages and downsides. For instance, pseudorandom structures provide the code designer with a

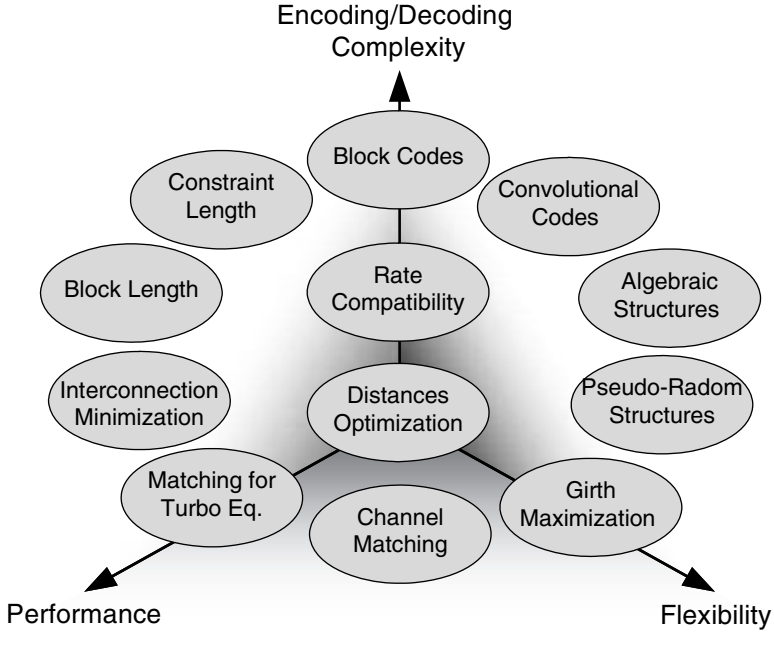

Figure 1.8 Design of codes defined on graphs as an optimization problem

lot of freedom. The codes originating from these structures can have practically any rate and block length. However, these codes have proven to be difficult to implement because of their complete lack of regularity. On the other hand, algebraic and combinatorial designs, which we will call structured designs from now on, cannot exist for all rates and block lengths. This happens because the algebraic or combinatorial designs are based on group and number theory and therefore they are inherently of a quantized nature, mainly based on prime numbers. Another characteristic of structured designs is that it is normally possible to obtain good codes for small to medium block lengths. Otherwise, pseudorandom designs have better performance for long block lengths. From an implementation point of view, structured designs have a lot of advantages. For instance, the regular connections in their graphs facilitate tremendously the hardware implementation of the decoders for these codes. The algebraic or combinatorial structure can also be exploited to simplify the encoding algorithm.

The following sections give an overview of existing codes for both pseudorandom and structured designs.

1.3 Pseudorandom Designs

The history of pseudorandom designs coincides with the history of codes on graphs. Already in his seminal work [Gal63], which introduced the LDPC codes, Gallager considered pseudorandom designs. Throughout the years other researchers have also considered some important pseudorandom designs. This is mainly because such codes are very flexible, as mentioned earlier; and also because they enable the use of some powerful techniques to study their asymptotic behavior.

In this section, some of these designs will be discussed for turbo codes and LDPC codes.

1.3.1 Pseudorandom Designs for Turbo Codes

When we mention the different designs of turbo codes, we are currently referring to their interleavers. The performance of a turbo code depends on how effectively the data sequences that produce low-weight codewords at the output of one encoder are matched with permutations of the same data sequence that yield higher encoded weights at the outputs of the others. Random interleavers do a very good job of combining low weights with high weights for the vast majority of possible information sequences. In this section, the most widely known interleavers of this class will be presented.

1.3.1.1 S-Random Interleavers

S-random interleavers were introduced by Divsalar and Pollara in [DP95]. The design of an S-random interleaver guarantees that, if two input bits to an interleaver Π are within distance S_1, they cannot be mapped to positions less than S_2 apart at the

interleaver output, and usually $S_1 = S_2 = S$ is chosen. So, considering two indices i, j such that

$$0 < |i-j| < S \tag{1.2}$$

the design imposes that

$$|\Pi(i)-\Pi(j)| > S. \tag{1.3}$$

When designing these interleavers, it was observed that $S < \sqrt{N/2}$ usually produces a solution in reasonable time, where N is the length of the interleaver to be designed.

Simulation results for the S-random interleavers and comparisons with other interleavers are shown later in Figure 1.14.

1.3.1.2 Pseudorandom Designs for LDPC Codes

It is well known that the message passing algorithm used to decode LDPC codes converges to the optimum a posteriori probability (APP) solution if the Tanner graph representing the parity-check matrix of the code has a tree structure. In light of this, Gallager, in his 1963 work, considered some pseudorandom designs that avoid short cycle lengths. Appendix C of his thesis [Gal63] presents the algorithms for generation of codes that avoid a certain minimum cycle length, called the *girth* of the graph.

In this section, we present the state-of-the-art in pseudorandom LDPC code generation with large girths.

1.3.1.3 Progressive Edge-Growth Tanner Graphs

The main idea behind this generation method, which was presented in [HEA01], is to establish progressively the edges or connections between variable and check nodes in an edge-by-edge manner so that the resulting graph shows the desired girth properties.

In summary, the progressive edge-growth (PEG) algorithm works as follows. Given the number of variable nodes n, the number of check nodes m, and the symbol–node degree sequence of the graph [RSU01], an edge-selection procedure is started such that the placement of a new edge on the graph has as small an impact on the girth as possible. After a *best-effort* edge has been determined, the graph with this new edge is updated and the procedure continues with the placement of a next edge. As we see, the PEG algorithm is a general, non-algebraic method for constructing graphs with large girths. Below, some necessary definitions and notations are shown, and based on that a more precise description of this algorithm is presented.

Definitions and Notations

$\mathbf{H}$ is the code's parity-check matrix with dimension $m \times n$, $h_{i,j}$ is the element in the i-th row and j-th column of $\mathbf{H}$.

A Tanner graph is denoted as (V, E) with V being the set of nodes, i.e., $V = V_c \cup V_v$, where $V = \{c_0, c_1, \ldots, c_{m-1}\}$ is the set of check nodes and $V = \{v_0, v_1, \ldots, v_{n-1}\}$ is the set of variable nodes. E is the set of edges such that $E = V \times V$ with edge $(c_i, v_j) \in E(c_i, v_i) \in E$ if $h_{i,j} \neq 0$.

A Tanner graph is called (d_v, d_c)-regular if every variable node participates in d_v check nodes and every check node involves d_c symbol nodes; otherwise it is called irregular.

The sequence of variable nodes degrees is denoted by $D_v = \{d_{v_0}, d_{v_1}, \cdots, d_{v_{n-1}}\}$, in which d_{v_j} is the degree of variable node v_j. On the other hand, the sequence of parity-check nodes degrees is given by $D_c = \{d_{c_0}, d_{c_1}, \cdots, d_{c_{m-1}}\}$, where d_{c_i} is the degree of check node c_i.

The set of edges E can be partitioned in terms of V_v as $E = E_{v_0} \cup E_{v_1} \cup \cdots \cup E_{v_{n-1}}$, with E_{v_j} containing all edges incident on variable node v_j.

The k-th edge incident on variable node v_j is denoted by $E_{v_j}^k$.

For a given variable node v_j, we define its neighbor within depth l, $N_{v_j}^l$, as the set consisting of all check nodes reached by a tree spreading from variable node v_j within depth l, as shown in Figure 1.9. Its complementary set, $\bar{N}_{v_j}^l$, is defined as $N_{v_j}^l \cup \bar{N}_{v_j}^l = V_s$.

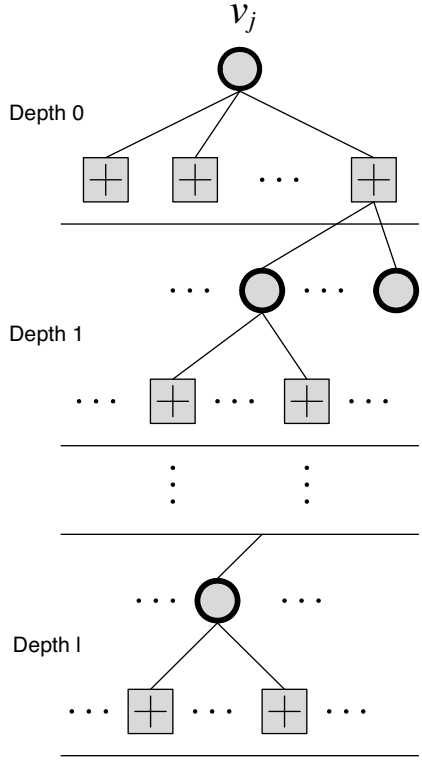

Figure 1.9 Neighbor $N_{v_j}^l$ within depth l of variable node v_j

Progressive Edge-Growth Algorithm
for $j = 0$ to $n - 1$ **do**
begin
  for $k = 0$ to $d_{v_j} - 1$ **do**
  begin
    if $k = 0$
      $E_{v_j}^0 \leftarrow (c_i, v_j)$, where $E_{v_j}^0$ is the first edge incident to v_j, and c_i is a check node having the lowest degree under the current graph setting $E_{v_0} \cup \cdots \cup E_{v_{j-1}}$.
    else
      expanding a tree from variable node v_j up to depth l under the current graph setting such that $\bar{\mathcal{N}}_{v_j}^l \neq \varnothing$ but $\bar{\mathcal{N}}_{v_j}^{l+1} = \varnothing$, or the cardinality of $\bar{\mathcal{N}}_{v_j}^l$ stops increasing but is less than m, then $E_{v_j}^k \leftarrow (c_i, v_j)$, where $E_{v_j}^k$ is the k-th edge incident to v_j and c_i is one check node picked from the set $\bar{\mathcal{N}}_{v_j}^l$ having lowest check-node degree.
  end
end

Figure 1.10 Progressive edge-growth algorithm

A description of the PEG algorithm can be found in Figure 1.10. It is worth mentioning at this point that it is possible to obtain codes with linear time encoding complexity using this algorithm. In this case, the edges associated with the m variable nodes of the codeword should be placed to form a so-called *zigzag* pattern [HEA01]. After these edges are placed, the conventional PEG algorithm can be used to place the remaining edges. The code obtained this way can be encoded using the *back-substitution* procedure.

In Figure 1.11, the performance of the codes obtained from the PEG algorithm is shown, especially regular PEG Tanner-graph codes are compared to Mackay's codes [MacBib] and random graph codes. The PEG codes are constructed such that their girth is 8. MacKay's codes and the random codes have girth equal to 6. The local girth distributions are shown in the legend of Figure 1.11. We can observe that the random codes perform much worse than the PEG codes and MacKay's codes. On the other hand, the PEG codes and MacKay's codes have similar performance, with the PEG code being slightly better for high SNRs.

Figure 1.12 shows the performances of irregular PEG codes compared with MacKay's codes and random codes with the same degree distribution. As we can

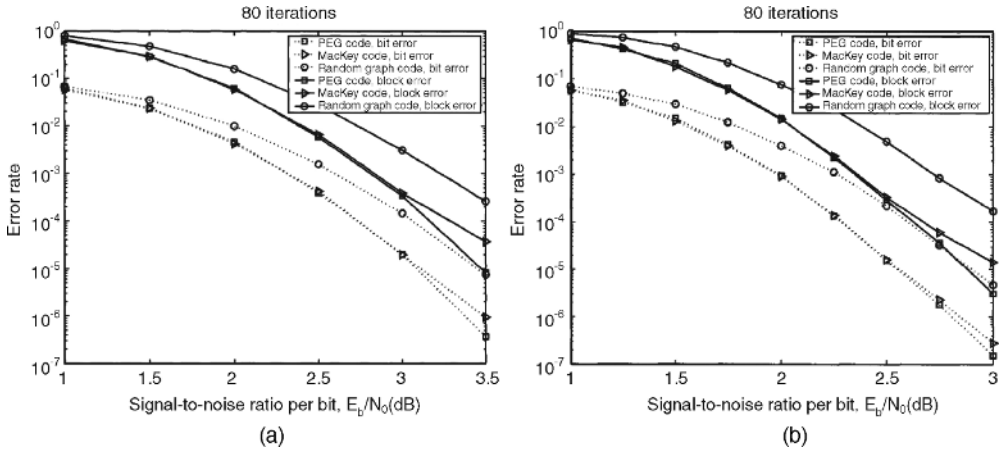

Figure 1.11 Bit and block error rates of PEG Tanner-graph codes, MacKay's codes and random graph codes [Hu02]. All codes are regular with distributions $d_v = 3$ and $d_c = 6$ with rate 1/2. (a) $n = 504$, $m = 252$; local girth distributions: PEG: 8; MacKay: 6 (63%), 8 (37%); random: 6 (79%), 8 (21%). (b) $n = 1008, m = 504$; local girth distributions: PEG: 8 (17%), 10 (83%); MacKay: 6 (39.5%), 8 (60.3%); random: 6 (55.6%), 8 (44.2%)

observe, the irregular PEG codes show the best performance. The irregular random codes show error floors very early. The local girth distributions are given in the legend of Figure 1.12.

In Figure 1.13 we show the performance of PEG codes exhibiting the zigzag pattern against the turbo codes. The PEG codes have degree distribution given by

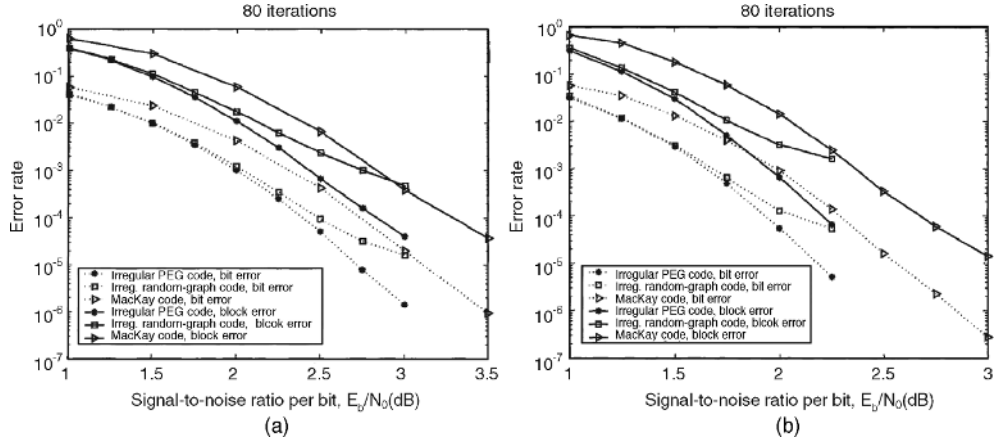

Figure 1.12 Bit and block error rates of PEG Tanner-graph codes, MacKay's codes and random graph codes [Hu02]. The PEG and random codes are irregular with rate 1/2. MacKay's codes are regular with rate 1/2. The degree distribution for the irregular PEG and random codes is given by $\Lambda(x) = 0.47532x^2 + 0.279537x^3 + 0.0348672x^4 + 0.108891x^5 + 0.101385x^{15}$. (a) $n = 504$, $m = 252$. (b) $n = 1008$, $m = 504$

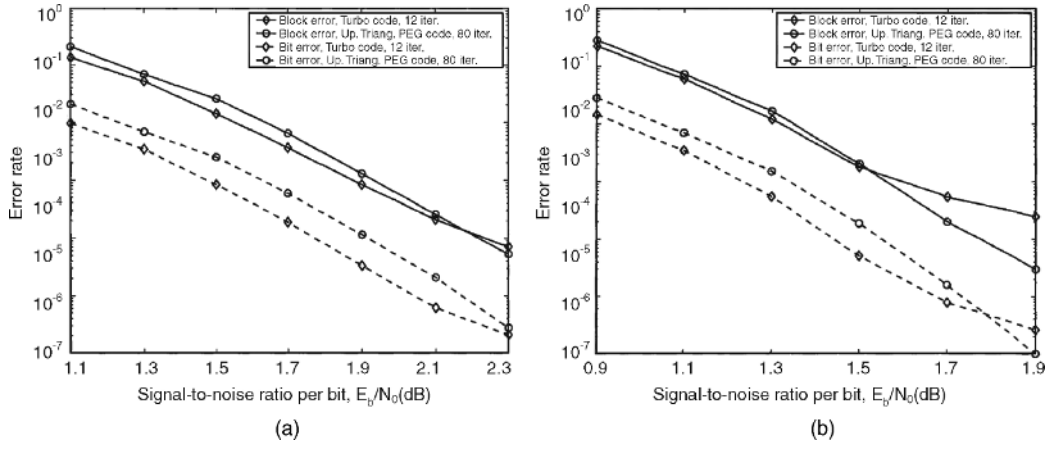

Figure 1.13 Bit and block error rates of linear time encodable PEG codes and turbo codes [Hu02]. All codes have an approximate rate of 1/2. (a) Block length 1024. (b) Block length 2048

$$\Lambda(x) = 0.477081x^2 + 0.280572x^3 + 0.0349963x^4 + 0.0963301x^5 + 0.0090884x^7 + 0.00137443x^{14} + 0.100558x^{15}$$

The turbo codes are the same standardized for the CDMA2000 system. They consist of two systematic, recursive, eight-state convolutional encoders concatenated in parallel, with an interleaver. The transfer function for the turbo encoders is given by

$$G(D) = [\,1 \quad n_0(D)/d(D) \quad n_1(D)/d(D)\,]$$

where

$$d(D) = 1 + D^2 + D^3,\ n_0(D) = 1 + D + D^3$$

and $n_1(D) = 1 + D + D^2 + D^3$.

The turbo codes are decoded using the BCJR algorithm with 12 iterations. The LDPC codes are decoded using 80 iterations of the message passing algorithm so that the decoding complexity for both code types is almost the same. As we can observe, the PEG codes are serious competitors for the turbo codes. For instance, they have similar performance for low SNRs and do not show the error floor behavior presented by the turbo codes in higher SNRs. Moreover, they have a low-complexity linear time encoding algorithm based on back-substitution.

1.3.2 Structured Designs

The main motivation for studying codes with structured designs is that simplified encoding/decoding algorithms can be derived and also some important characteristics (for example, distances or bounds) can be determined easily.

In this section, we provide an overview of some important constructions.

1.3.2.1 Structured Designs for Turbo Codes

The parallel processing of the iterative decoding of turbo codes is of interest for high-speed communication systems. Interleaving of extrinsic information is one important aspect to be addressed in a parallel decoder because of the memory access contention problem [TBM04]. The first approach to solve the memory access contention problem is by simply constraining the interleavers to be contention-free [Nim04]. However, if the interleaver is required to be unconstrained, then the memory contention problem can still be solved as shown in [TBM04], but at a cost of additional complexity.

In this section, we present a class of algebraic structured interleavers that are contention-free. The advantage of these interleavers is that they result in low-complexity parallel decoders induced from the algebraic structure, with good performance when compared against some good interleavers.

1.3.2.2 Maximum Contention-Free Permutation Polynomial Interleavers

This class of structured interleavers was introduced by Sun and Takeshita in [ST05]. The main elements of this construction are the permutation polynomials over integer rings. We start this section by defining these polynomials.

Definition
Given an integer $N > 2$, a polynomial $f(x) = f_1(x) + f_2(x) \pmod N$, where f_1 and f_2 are non-negative integers, is said to be a quadratic permutation polynomial (QPP) over the ring of integers Z_N when $f(x)$ permutes $\{0, 1, 2, \ldots, N-1\}$.

The conditions that f_1 and f_2 must fulfill for the existence of $f(x)$, as well as the search procedure for f_1 and f_2, are presented in [ST05]. In [Tak05] it was proved that every quadratic permutation polynomial generates a maximum contention-free interleaver (MCF). (If an interleaver is contention-free for all window sizes W dividing the interleaver size N, it is called a *maximum contention-free interleaver*.)

Table 1.2 shows examples of MCF interleavers obtained from permutation polynomials over integer rings. The polynomials $g(x)$ are the inverses of $f(x)$ and are obtained using the methods presented in [RT05].

Figure 1.14 shows the performance of turbo codes with MCF quadratic permutation polynomial (QPP) interleavers compared to turbo codes with S-random and 3GPP standardized interleavers. As can be observed, the codes with MCF-QPP interleavers

Table 1.2 Examples of MCF interleavers from permutation polynomials over integer rings

N	$f(x)$	$g(x)$
256	$159x + 64x^2 \pmod N$	$95x + 64x^2 \pmod N$
1024	$31x + 64x^2 \pmod N$	$991x + 64x^2 \pmod N$
4096	$2113x + 128x^2 \pmod N$	$4033x + 1920x^2 \pmod N$
15120	$11x + 210x^2 \pmod N$	$14891x + 210x^2 \pmod N$

have the same performance as the other codes in the low SNR regime. In higher SNRs, we can observe that codes resulting from the MCF-QPP interleavers show better error floor behavior than the other codes. This fact is most evident for the block length 4096.

At this point, we should make some observations about the implementation complexity of the codes with MCF-QPP interleavers. Firstly, because these interleavers are inherently contention-free, full parallel decoders can be implemented with no need for additional units to serialize the memory access due to access conflicts. Moreover, there is no need for look-up tables to perform the interleaving. The algebraic structure imposed by the permutation polynomial makes the online calculation of the interleaving addresses possible. These online calculations are performed with minimum effort: only three multiplications and one addition are necessary.

1.3.2.3 Structured Designs for LDPC Codes

1.3.2.3.1 Quasi-Cyclic LDPC Block Codes

The research on quasi-cyclic (QC) LDPC codes was motivated by the results obtained by Tanner for the [15, 64, 20] code in [Tan00]. After this paper, other studies [Fos04]

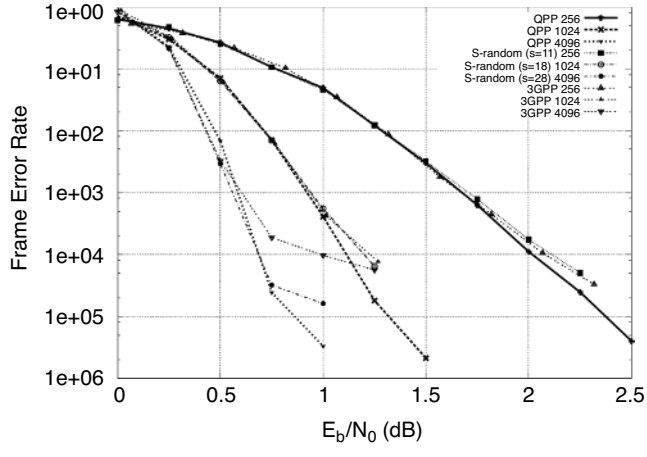

Figure 1.14 Frame error rate (FER) curves comparing the performances of turbo codes with S-random, 3GPP and MCF quadratic polynomial interleavers [Tak05]. All codes have a nominal rate of 1/3 and are evaluated for the block lengths 256, 1024 and 4096

[Tan04] extended these results to obtain codes for several block lengths and rates. We start this section by defining the construction procedure for QC LDPC block codes.

For a prime m, the integers $\{0, 1, \ldots, m-1\}$ form a field under addition and multiplication modulo m – the Galois field GF(m). The nonzero elements of GF(m) form a cyclic multiplicative group. Let a and b be two nonzero elements with multiplicative orders $o(a) = k$ and $o(b) = j$, respectively. Then we form the $j \times k$ matrix P of elements from GF(m), which has as its (s, t)-th element $P_{s,t} = b^s a^t$ as follows:

$$\mathbf{P} = \begin{bmatrix} 1 & a & a^2 & \cdots & a^{k-1} \\ b & ab & a^2 b & \cdots & a^{k-1} b \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ b^{j-1} & ab^{j-1} & a^2 b^{j-1} & \cdots & a^{k-1} b^{j-1} \end{bmatrix}. \tag{1.4}$$

In the matrix above, $0 \leq s \leq j-1$ and $0 \leq t \leq k-1$. The LDPC code is constructed by specifying its parity-check matrix $\mathbf{H}$ as the $j \times k$ array of circulant submatrices given by

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_1 & \mathbf{I}_a & \mathbf{I}_{a^2} & \cdots & \mathbf{I}_{a^{k-1}} \\ \mathbf{I}_b & \mathbf{I}_{ab} & \mathbf{I}_{a^2 b} & \cdots & \mathbf{I}_{a^{k-1} b} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \mathbf{I}_{b^{j-1}} & \mathbf{I}_{ab^{j-1}} & \mathbf{I}_{a^2 b^{j-1}} & \cdots & \mathbf{I}_{a^{k-1} b^{j-1}} \end{bmatrix}, \tag{1.5}$$

where $\mathbf{I}_x$ is an $m \times m$ identity matrix with rows cyclically shifted to the left by x positions. The circulant submatrix in position (s, t) within $\mathbf{H}$ is obtained by cyclically shifting the rows of the identity matrix to the left by $P_{s,t}$ positions. The resulting binary parity-check matrix is of size $jm \times km$, which means the associated code has a rate $R \geq 1-(j/k)$. Actually, we have this inequality because some parity equations of $\mathbf{H}$ may happen to be linearly dependent. By construction, we also note that every column of $\mathbf{H}$ contains j ones and every row contains k ones, and so $\mathbf{H}$ represents a (j, k) regular LDPC code.

The construction above can also be generalized to generate irregular codes. In this case certain circulant submatrices of the original parity-check matrix are substituted by all-zero matrices of the same dimension. This procedure is shown in [Tan04]. We can also improve the distances of these QC codes if, instead of considering circulants formed by single shifted identity matrices, we also consider the sum of shifted identity matrices. These considerations were presented in [SV04].

Figure 1.15 shows the performance of several QC LDPC block codes with the format of (1.5) compared to random LDPC codes with the same block lengths. The rate of all codes is approximately 0.4 and they were decoded using a

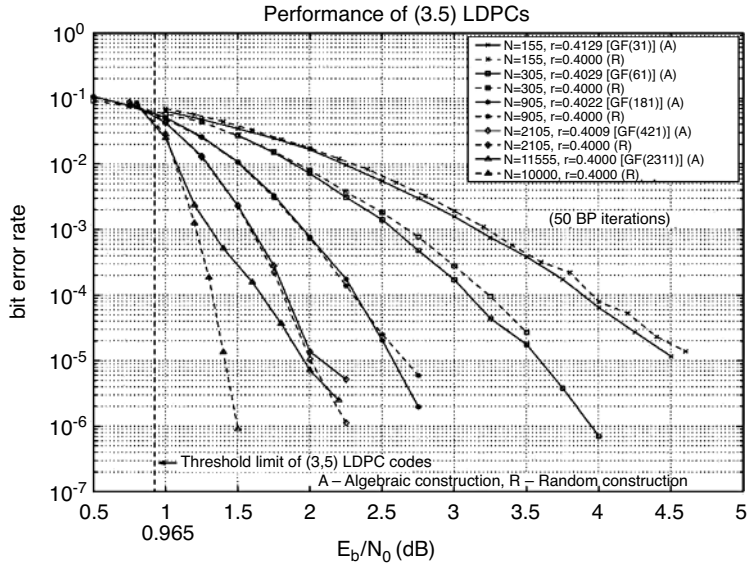

Figure 1.15 Performances of quasi-cyclic LDPC codes compared against random LDPC codes [Tak04]

maximum of 50 iterations of the message passing algorithm. As we can note, the codes have similar performance for small to medium block lengths. However, for bigger block lengths the random codes perform better and the QC LDPC codes start showing some error floor.

1.3.2.3.2 Encoders for QC LDPC Block Codes

Using Tanner's transform theory [Tan88], it is possible to show that every parity-check matrix formed by circulant submatrices results in a generator matrix formed also by circulant submatrices. As we know, generators with these properties can be implemented using shift registers.

As an example, we show in Figure 1.16 the matrices defining a QC LDPC block code. In this case, the parity-check matrix is full-rank. This implies that Tanner's transform theory will be simplified to the case that the generator matrix is obtained by partitioning the parity-check matrix as $\mathbf{H} = [\mathbf{Q}\,\mathbf{S}]$ ($\mathbf{S}$ is square) and by finding $\mathbf{W}$, such that $\mathbf{WS} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix. The generator matrix will then be given by $\mathbf{G} = [\mathbf{I}\,(\mathbf{WQ})^T]$ [ADT05]. As we can observe in Figure 1.16, the generator matrix is not low density, but because of the cyclic structure it can be encoded easily using shift registers. The scheme for such encoders is shown in Figure 1.17.

1.3.2.3.3 LDPC Convolutional Codes

One of the major challenges in the implementation of decoders for LDPC codes is the interconnection between the processing elements (variable and check nodes). As

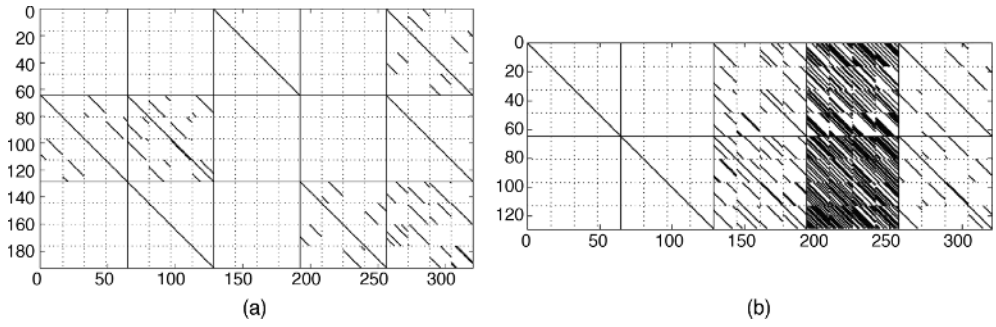

Figure 1.16 Matrices defining a QC LDPC block code. (a) Parity-check matrix. (b) Generator matrix [ADT05]

shown in [BH02], several problems arise from a full parallel implementation of these decoders – for example, die size, power consumption, congestion and limitations of the clock frequency. For this reason, full parallel implementations for LDPC codes with large block length becomes prohibitive.

An elegant way to circumvent the interconnection problem is by adding some algebraic structure to the parity-check matrix **H** in such a way that the interconnection problem can be minimized. In this context, Jiménez-Felström and Zigangirov [JZ99] have proposed a new class of codes, the LDPC convolutional codes.

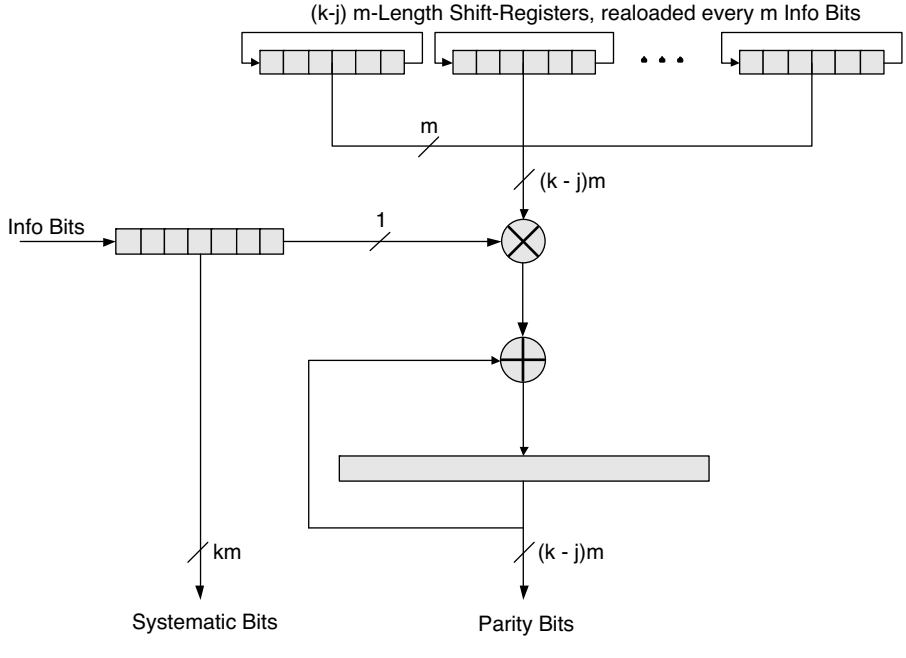

Figure 1.17 Encoder scheme for QC LDPC block codes based on circulant submatrices

An (m_s, J, K)-regular LDPC convolutional code is a set of sequences v satisfying the equation $vH^T = 0$, where

$$H^T = \begin{bmatrix} H_0^T(0) & \cdots & H_{m_s}^T(m_s) & & \\ & \ddots & & \ddots & \\ & & H_0^T(t) & \cdots & H_{m_s}^T(t+m_s) \\ & & \ddots & & \ddots \end{bmatrix}. \tag{1.6}$$

Here, $\mathbf{H}$ is the semi-infinite syndrome former matrix. For a rate $R = b/c$ $(b < c)$ LDPC convolutional code, the elements $H_i^T(t)$, $i = 0, 1, \ldots, m_s$, are binary $c \times (c-b)$ submatrices defined as

$$H_i^T(i) = \begin{bmatrix} h_i^{(1,1)}(t) & \cdots & h_i^{(1,c-b)}(t) \\ \vdots & & \vdots \\ h_i^{(c-1,1)}(t) & \cdots & h_i^{(c,c-b)}(t) \end{bmatrix}. \tag{1.7}$$

The value m_s is called the syndrome former memory and the associated constraint length is defined as $v_s = (m_s + 1) \cdot c$.

From its definition, we can conclude that the Tanner graph of an LDPC convolutional code has an infinite number of nodes. However, the distance between two variable nodes that are connected by the same check node is limited by the syndrome former memory m_s of the code. As a consequence, the decoding of two variable nodes that are at least $(m_s + 1)$ time units apart from each other can be performed independently, since they do not participate in the same parity-check equation. This allows continuous decoding that operates on a finite window sliding along the received sequence. Furthermore, the I iterations can be realized in parallel by I identical processors that work on different sections of the Tanner graph. Alternatively, since the processors implemented in the decoder hardware are identical, a single "hopping" processor that runs on different sections of the decoder memory successively can also be employed.

The pipeline decoding architecture mentioned above was also proposed in [JZ99]. The pipeline decoder outputs a continuous stream of decoded data once an initial decoding delay has elapsed. The operation of this decoder on the Tanner graph for a simple time-invariant rate $R = 1/3$ LDPC convolutional code with $m_s = 3$ is shown in Figure 1.18.

The VLSI implementation of LDPC convolutional decoders is based on replicating identical units called processors. As illustrated in Figure 1.19, the complete decoder can be constructed by concatenating a number of these processors together. It is now obvious from Figures 1.18 and 1.19 that the routing complexity in the case of LDPC

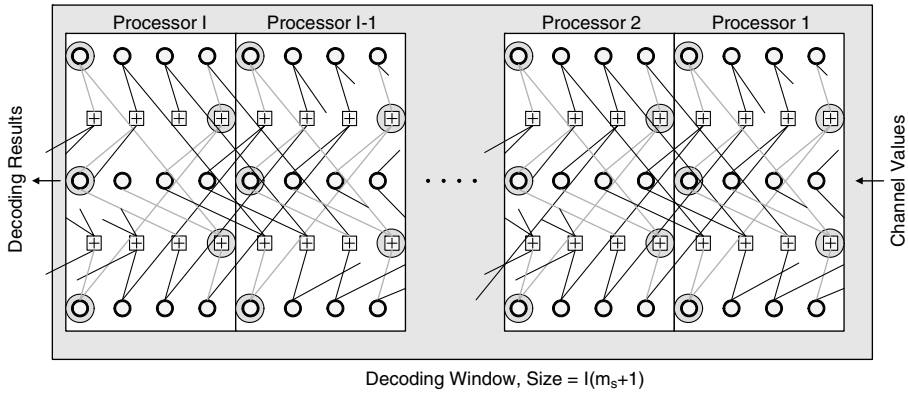

Figure 1.18 Tanner graph of an $R = 1/3$ LDPC convolutional code and an illustration of pipeline decoding

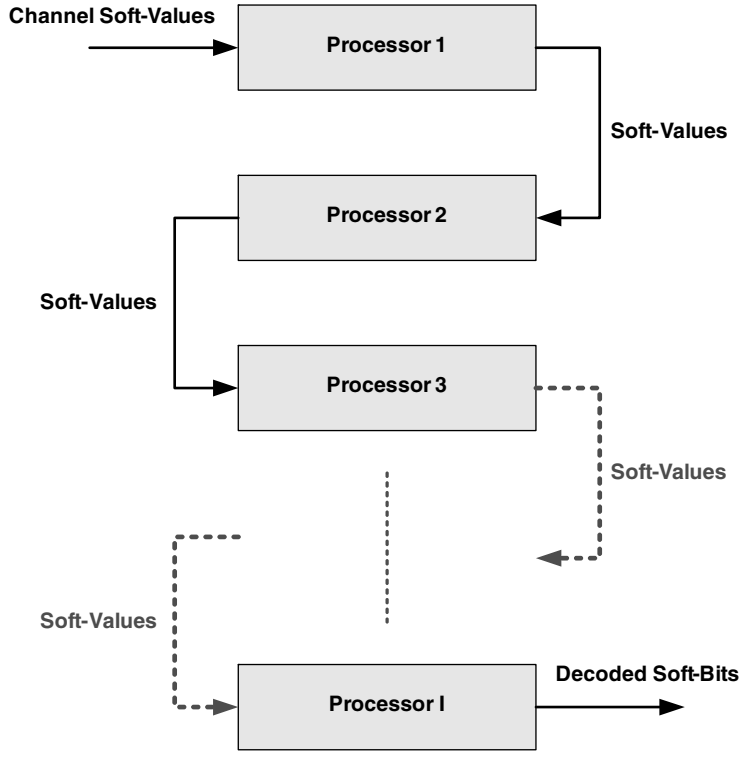

Figure 1.19 Low-density parity-check convolutional decoder by concatenation of identical processors

convolutional codes was reduced to the interconnection within each processor, which is an order of magnitude less than for a block code.

Another advantage of LDPC convolutional codes is their encoding complexity. As shown in [RU01], the encoding complexity of LDPC block codes after some operations in their parity-check matrices is upper-bounded by $O(N + g^2)$, where N is the block length and g is a small factor that depends on the structure of the parity-check matrix. In the case of LDPC convolutional codes, the encoding complexity is proportional to the density of ones in the columns of $\mathbf{H}$ and because of their intrinsic convolutional structure, the encoding can be realized by shift-register operations [JZ99].

Figure 1.20 [Jim06] shows the performance of LDPC convolutional codes compared to random LDPC block codes. The LDPC convolutional codes were generated using the *unwrapping* procedure presented in [JZ99]. The curves are plotted for different block lengths and the rate loss due to the termination of the LDPC convolutional codes is already expressed in their results as SNR loss. As we can observe, the LDPC convolutional codes perform very well. Moreover, from the implementation complexity point of view, the LDPC convolutional codes show several

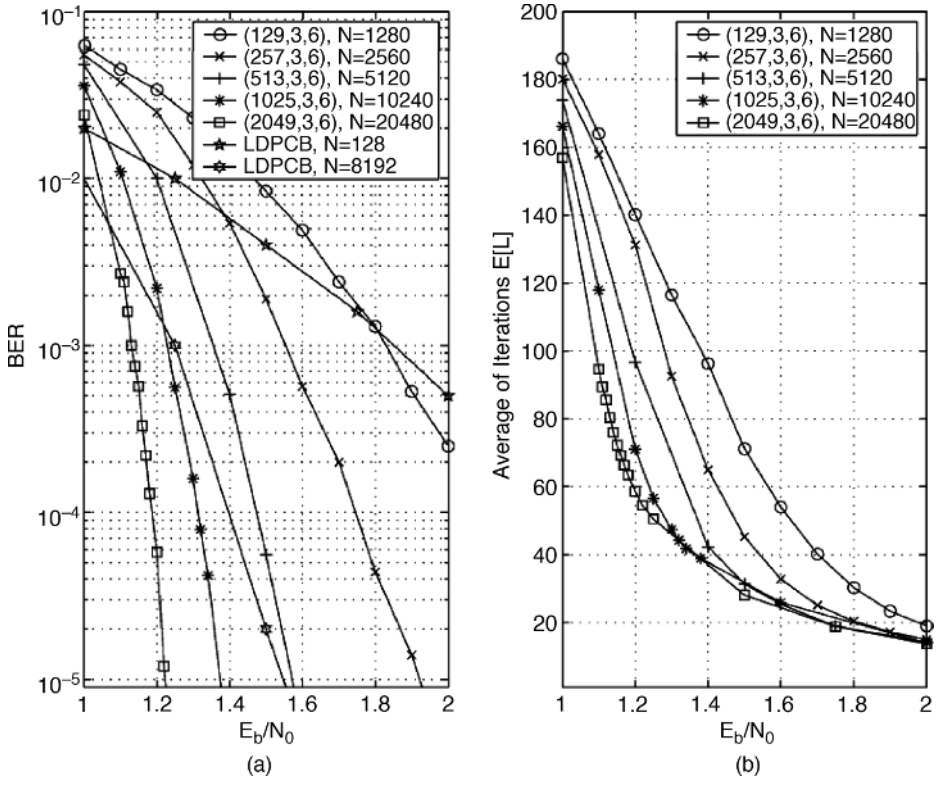

Figure 1.20 Performance of LDPC convolutional codes compared against random LDPC block codes. All codes have rate approximately 1/2 [Jim06]. (a) BER curves. (b) Number of iterations as a function of the SNR

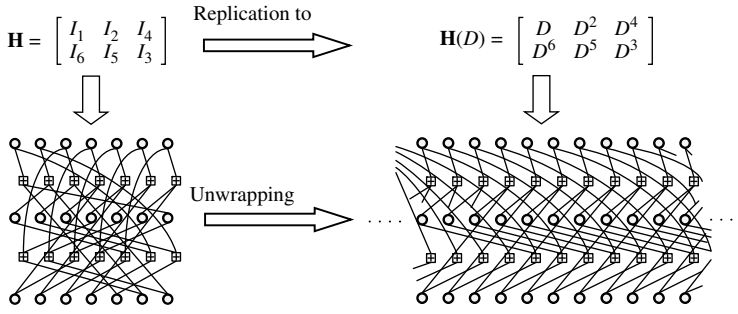

Figure 1.21 Relation between QC LDPC block codes and LDPC convolutional codes

advantages. For instance, it is fair to compare the memory of a convolutional code with the block length of a block code. In this case, we can see in Figure 1.20 that the convolutional code with memory $m_s = 513$ has similar performance as the block code with length 8192. Here, the implementation complexity of the last code is almost 16 times larger than the convolutional code. In Figure 1.20, we also show the number of required iterations for a certain SNR value.

1.3.2.3.4 LDPC Convolutional Codes and QC LDPC Block Codes

It was shown in [Tan04] that there is a direct relationship between QC LDPC block codes and LDPC convolutional codes. Actually, LDPC convolutional codes can be obtained from QC LDPC block codes by replicating their graphs to infinity. This operation is equivalent to unwrapping the graph of the QC code, so that a QC code can be seen as a tail-biting convolutional code.

Figure 1.21 depicts what is said in the paragraph above. As we can observe, the graph of the convolutional codes is infinite but very well structured and it also preserve the $m_s + 1$ maximum distance between nodes property. We should also mention that the codes obtained from the QC codes are time-invariant. On the other hand, the codes constructed with the unwrapping procedure [JZ99] can be time-invariant or time-variant.

1.3.3 Code Optimization

Code design and optimization before the invention of turbo coding in 1993 concentrated on code properties, such as minimum distance or free distance. Decoding was performed using maximum likelihood decoders, so the choice of a code uniquely determined the decoder and the performance of the coding system only depended on the choice of the code. With iterative decoding, the performance of the coding system depends on the specific code description given to the decoder, and so the focus has shifted to decoder design rather than code design.

Constructing a good short- to medium-length code that will decode well under iterative decoding is an art, with many heuristic rules and constraints to fulfill, some to do with decoder performance, some with easy encoder implementation, and so forth.

To speak of "code optimization" in this context is to make an overstatement, and the final stages of practical code design are usually performed by tweaking and analyzing the simulated error performance curve in its three stages of "error floor," "waterfall," and "error ceiling." The following sections will elaborate on the design of practical short- to medium-length codes.

In the asymptotical regime when the code length goes to infinity, the performance of the coding system can be predicted very accurately using techniques such as *density evolution* or *EXIT charts*. Based on these techniques, it is possible to run numerical algorithms to optimize the code design. For very long codes (length above 10^4), the resulting code designs and performance predictions are close approximations of the measured performance. Although the predictions and designs become less accurate when the code length becomes smaller, the parameters calculated for the asymptotical regime are commonly used as one of several guidelines for the design of short- and medium-length codes as well.

In the asymptotical regime of infinite code length, the error performance of a coding system becomes binary: above a certain threshold in terms of SNR or channel capacity, the probability of error tends to zero, and below the threshold it tends to one. Therefore, analysis techniques concentrate on predicting and optimizing the threshold.

The most accurate method for predicting the threshold of an LDPC code is *density evolution* [RU03]. This method involves tracking the probability density function of the messages exchanged in the decoder graph throughout the iterations. This can be done analytically for some channels and decoders, or approximated numerically using sampled densities if no analytical expressions can be found for the density mapping at the graph components. The analysis is done for isolated components in the graph, assuming their input messages to be *independent*, and deductions are made for the convergence of the global algorithm. This *independence assumption* is verified asymptotically when the code length goes to infinity and interleavers are random, which is why density evolution is only accurate in the asymptotical regime.

When analytical expressions can be found for the density mappings as a function of the code parameters, it is possible to optimize the code parameters to maximize the threshold. This is the case for LDPC codes, where there exist designs that match the channel capacity to the rate as closely as desired. The threshold for LDPC codes is a function of the *degree polynomials*, which specify the density of ones per column and per row in the parity-check matrix used by the iterative decoder. A popular web-based source of binary LDPC designs is available at Rüdiger Urbanke's website (http://lthcwww.epfl.ch/research/ldpcopt/), where capacity-approaching degree polynomials for the binary-input AWGN channel and for the binary erasure channel can be generated.

Density evolution can be simplified by tracking a reduced set of parameters of the message densities instead of tracking the densities. *EXtrinsic Information Transfer (EXIT) charts* [AKB04] plot the *mutual information* between the messages and their corresponding code digits. An example EXIT chart for a regular LDPC code is given in Figure 1.22.

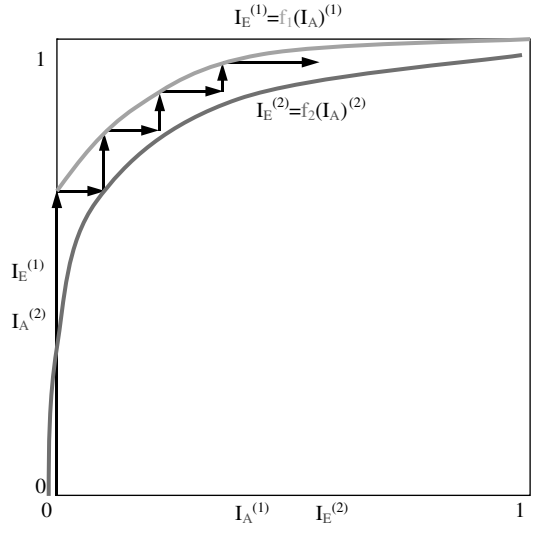

Figure 1.22 EXIT chart of regular LDPC codes

The upper curve maps the mutual information at the input of a variable node to the output of a variable node, and depends on the channel capacity. The lower curve maps (with inverted coordinates) the mutual information at the input of a check node in the graph to the output of the check node. The "trajectory" of the decoder is plotted as a succession of arrows where the mapping of each curve is applied in turn. If one changes the capacity of the channel, the starting point of the upper curve on the *y*-axis will vary, and the curve will be shifted up or down accordingly. As long as the curves do not intersect with each other, the decoder will reach the top right point in the graph, which corresponds to a level of mutual information equal to 1 – the messages become fully correlated with the code digits and the error probability tends to zero. The threshold is the value of the channel capacity for which the two curves intersect exactly once. Like density evolution, EXIT charts rely on the independence assumption and can make an accurate prediction only for very long codes. Unlike density evolution, they also rely on an assumption for the probability density of the incoming messages at each decoder component. This makes them less accurate than density evolution, and there is usually a loss of up to 0.05 dB for codes designed with EXIT charts as compared to codes designed with density evolution.

The advantage of EXIT charts is that they are easier to apply to a variety of scenarios, like serial or parallel (turbo) concatenation of convolutional codes, or iterative detection plus MIMO multiuser detection [BK03][BKA04] [LSL06]. They provide a visualization of the iterative process, and thereby allow an understanding of effects that are difficult to pinpoint with density evolution, such as decoder-induced error floors or the effect of scheduling on the convergence of doubly iterative processes, to name just a few.

1.4 Repeat Accumulate Codes

It was mentioned before that encoding complexity is a problem with general LDPC codes. This disadvantage can be avoided with structured LDPC codes. Here, we describe repeat accumulate (RA) codes as one possibility to obtain an encoding complexity that grows only linearly with the block size. Repeat accumulate codes can be viewed as LDPC codes or as a serial concatenated coding scheme.

Repeat accumulate codes are characterized by a parity check matrix which can be written in the form $\mathbf{H} = [\mathbf{H}_1 \mathbf{H}_2]$, where $\mathbf{H}_2$ is a banded matrix of the form

$$\mathbf{H}_2 = \begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 \\ 1 & 1 & 0 & & \\ 0 & 1 & 1 & 0 & 0 \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 & 1 \end{bmatrix}.$$

The corresponding generator matrix can be written as

$$\mathbf{G} = \left[\mathbf{I}, \mathbf{H}_1^T \mathbf{H}_2^{-T}\right].$$

Since

$$\mathbf{H}_2^{-T} = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 0 & 1 & 1 & 1 & \cdots & 1 \\ 0 & 0 & 1 & 1 & \cdots & 1 \\ \vdots & & \ddots & \ddots & & \vdots \\ 0 & 0 & \cdots & 0 & 1 & 1 \\ 0 & 0 & \cdots & 0 & 0 & 1 \end{bmatrix}$$

describes an accumulator, the encoder essentially can be represented as shown in Figure 1.23.

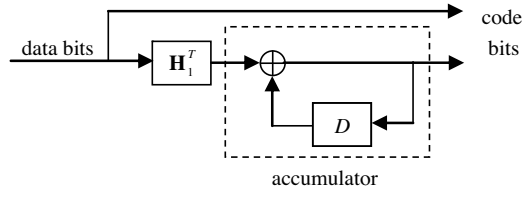

Figure 1.23 Repeat accumulate encoder

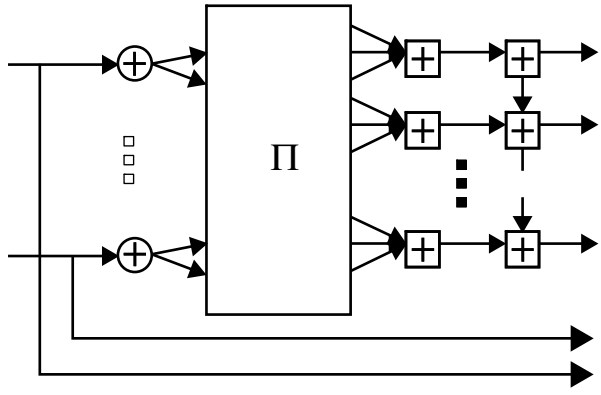

Figure 1.24 Repeat accumulate code: LDPC code-based representation

Another representation is given in Figure 1.24. Here, the repeat accumulate code is represented as an LDPC code plus accumulator. The connections between variable nodes and check nodes according to $\mathbf{H}_1^T$ can be interpreted as an interleaver.

However, the interleaver is not algebraic and has to be designed carefully using methods such as, for example, the progressive edge growth (PEG) algorithm. This is a disadvantage because the interleaver pattern has to be stored for each supported block size, which results in high memory requirements. Given this representation, it is straightforward to obtain the encoder block diagram as depicted in Figure 1.25. An RA code can be viewed as the serial concatenation of a repetition code and a modulo 2 adder plus accumulator.

The structure of RA codes not only allows encoding with a complexity that grows linearly with the block size but also offers several decoding options. The whole RA code can be represented by a Tanner graph on which message passing decoding is performed, as is usually done for LDPC codes (see the left-hand side of Figure 1.26). On the other hand, we can also exploit the trellis structure of the inner encoder in Figure 1.25 – we use a trellis decoder for the accumulator part and decode the remaining part of the code via message passing on a smaller graph (see the right-hand

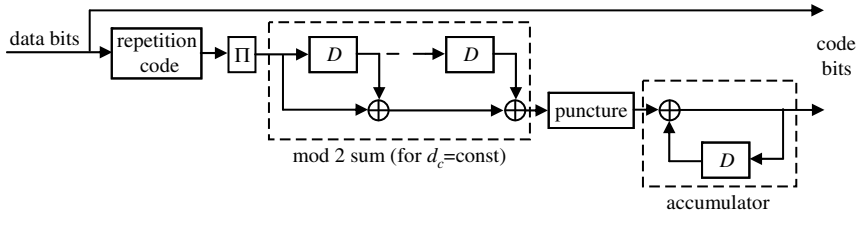

Figure 1.25 Repeat accumulate encoder: repetition code-based representation

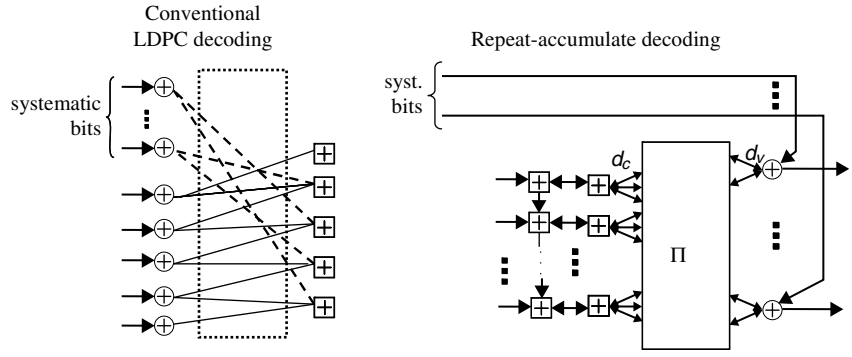

Figure 1.26 Decoding options for repeat accumulate codes

side of Figure 1.26). This can speed up convergence – fewer iterations are necessary compared to full message passing decoding of the complete code.

In Figure 1.27, we compare the BLER performance of PCCC, SCCC and RA codes with QPSK modulation in an AWGN channel. The block size is 1000 bits. Different code rates are obtained by regular puncturing. We used the UMTS turbo code with memory 3 constituent codes as PCCC and the SCCC with memory 2 constituent codes as described above. The RA code was optimized as described in [BK03]. We perform eight iterations for both PCCC and SCCC whereas 20 (broken lines) or 30 (solid lines) iterations, respectively, are performed for the RA code.

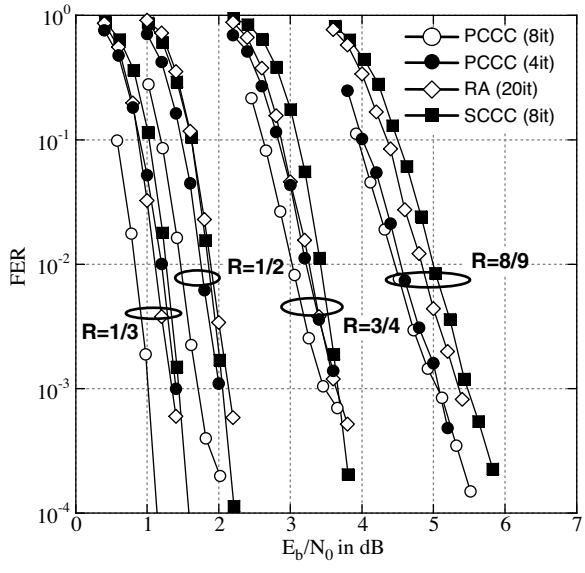

Figure 1.27 Block error rate performance of PCCC, SCCC and RA codes

Table 1.3 Complexity of iterative decoding

	PCCC	SCCC	RA
Additions (weight 1)	198	128.5	120
Comparisons (weight 1)	60	35	140
Multiplications (weight 10)	2	2.5	
Total	278	187	260

It can be observed that PCCC outperforms the other codes for all code rates. However, the error floor of PCCC starts at a higher BER, at least for some code rates. At a BLER of 10^{-2}, PCCC outperforms SCCC by 0.4–0.6 dB depending on the code rate. The degradation of RA codes compared to PCCC is in the range 0.2–0.5 dB.

The decoding complexity in terms of number of operations per iteration per info bit is summarized in Table 1.3. SCCC shows the lowest complexity due to the use of low-memory constituent codes. RA codes have only slightly lower decoding complexity than PCCC. However, it has to be taken into consideration that RA codes require many more iterations for comparable performance, for example 30 iterations whereas PCCC needs only 8 iterations. Consequently, RA codes have the highest decoding complexity of the three schemes.

1.5 Binary versus Nonbinary

Binary LDPC codes can be generalized to nonbinary LDPC codes (NB-LDPC). The parity-check equations are written using symbols in the Galois field of order q, denoted GF(q), where $q = 2$ is the particular binary case. The parity-check matrix defining the code has only a few nonzero coordinates, which belong to GF(q), and a single parity equation involving d_c codeword symbols then has the form:

$$\sum_{i=1}^{d_c} h_{ji} \cdot c_i = 0 \tag{1.8}$$

where $\{h_{ji}\}$ are the nonzero values of the j-th row of $\mathbf{H}$.

In terms of algebraic properties and error-correcting capabilities there is not much difference between nonbinary and binary codes, and there is a valid question about whether it is useful to consider NB-LDPC codes. If we leave aside the better behavior of nonbinary codes for correcting bursts of errors, the principal reason for using NB-LDPC codes lies in the fact that the practical decoder is suboptimal, which is the case for the belief propagation (BP) decoder, or its reduced complexity derivatives. In particular, it is useful to consider nonbinary LDPC codes when the nonbinary decoder is much closer to optimal maximum likelihood decoding (MLD) than its binary counterpart.

Let us discuss some general issues that assist in understanding NB-LDPC code advantages. It is well known nowadays that the drawbacks of belief propagation decoding of binary LDPC codes come from the dependence of the messages in the Tanner-graph representation of the code. The dependence comes from very specific topological structures in the Tanner graph of the code, for example cycles, stopping or trapping sets. The poor behavior of the BP decoder on these topological structures is enhanced if the log-likelihood ratio (LLR) messages that initialize the decoder are already correlated by the channel.

In the following two examples, the use of NB-LDPC codes helps to bypass correlation effects of the messages.

Short/Moderate Length Codes

The Tanner graph of the NB-LDPC code is much sparser than that of a binary code with the same parameters. This has been pointed out by several authors [DM98] [MD99][HE04][PFD06b]. As a consequence, the higher girth of NB-LDPC graphs helps to avoid the short cycles and also mitigates the effect of stopping or trapping sets, making the BP decoder closer to MLD. Actually, when $q \geq 256$, the best error rate results on binary input channels are obtained with the lowest possible variable node degree, that is $d_v = 2$. These codes have been named *cycle codes* in the literature, or *ultra-sparse* LDPC codes [DM98][MD99]. For example, the girth of a binary irregular LDPC code with length $N = 848$ bits and rate $R = 1/2$ is at most $g_b = 6$ for the good degree distributions, while the girth of a NB-LDPC code with the same parameters is $g_{nb} = 14$ when a good graph construction is used [HE04].

High-Order Modulation (M-QAM)

For binary LDPC-coded modulations, the output of the Bayesian maximum a posteriori demapper gives correlated probability weights, which means that the initialization of the BP decoder will experience correlated messages even without any short cycles. Of course, there are several ways of fighting this effect, by using an interleaver (BICM-LDPC), or using multilevel coding. However, if the LDPC code is built in a field with an order equal to or higher than the modulation order, the nonbinary LDPC decoder is initialized with uncorrelated vector messages, which helps the BP decoder to be closer to MLD. This way, the code operates in the modulation signal set, like in the trellis-coded modulations. The application of NB-LDPC codes to high-order modulations has been proved to be very efficient, both with analytical approaches and in simulations [SF02][DCG04][BB06].

So, if one accepts increasing the decoding complexity of the receiver, it is possible to expect a significant performance gain in the cases described above. We will give some evidence of the advantages of NB-LDPC codes. Before describing a few interesting simulation results in detail, we will discuss briefly the most recent results regarding the optimization of NB-LDPC codes.

Because of the very low density of NB-LDPC graphs, there is not much room left to optimize irregularity profiles, as is done for the binary irregular LDPC codes. Some authors have generalized the methods based on density evolution used in the optimization of binary LDPC codes. All convenient optimization methods are based on a Gaussian approximation of the densities, also referred to as EXIT charts for LDPC codes [LFK03][BT05][BB06]. However, the irregularity profiles obtained only apply to very long codeword lengths. For short-length codes, better results are obtained with quasi-cyclic nonbinary LDPC codes [SZL + 06] or ultra-sparse LDPC codes with large girths whose coefficients are chosen appropriately [DM98][PFD06a][KGP06] [PFD06b].

1.6 Performance Results of Nonbinary LDPC Codes

1.6.1 Small Codeword Lengths

Figures 1.28 and 1.29 show two examples of the interest of NB-LDPC codes at small codeword length on the BI-AWGN channel. In each figure, NB-LDPC codes opti-

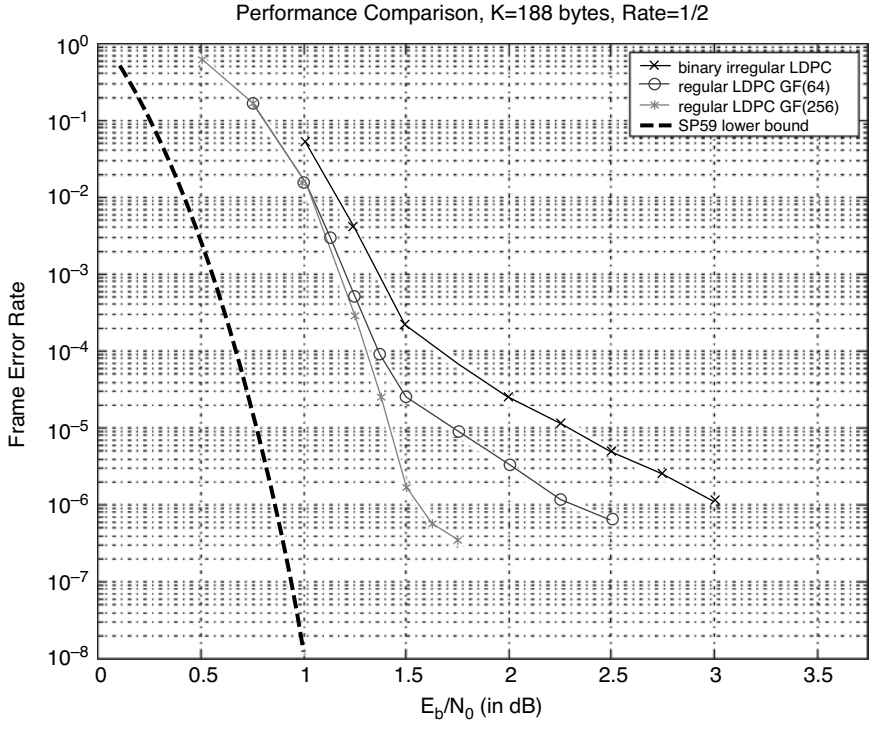

Figure 1.28 Performance comparison of binary versus NB-LDPC codes. Code parameters are $N = 3008$ coded bits and rate $R = 1/2$

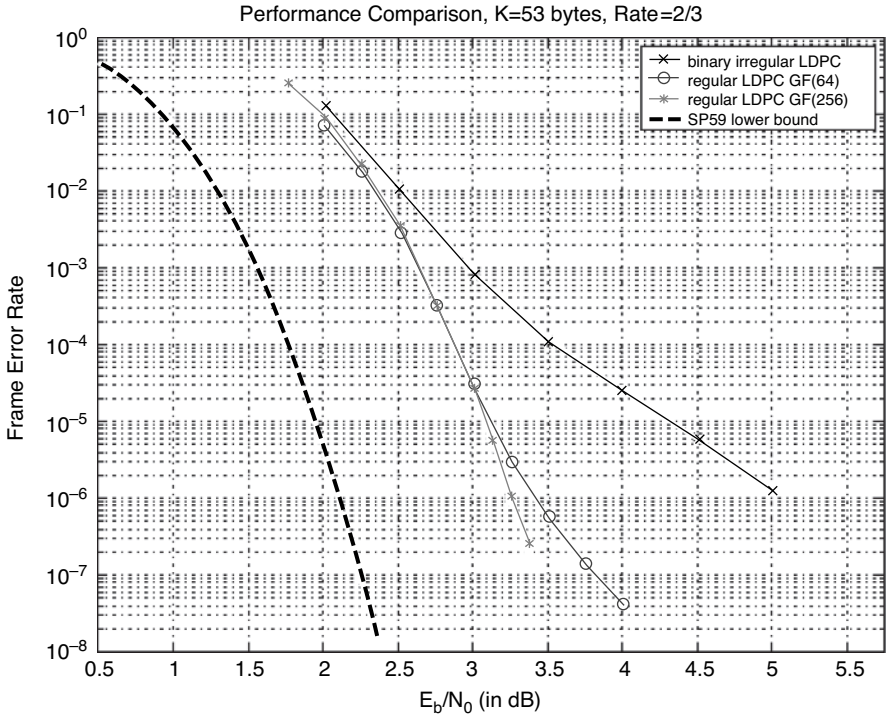

Figure 1.29 Performance comparison of binary versus NB-LDPC codes. Code parameters are $N = 564$ coded bits and rate $R = 2/3$

mized using the method proposed in [PFD06a] are simulated together with an irregular binary LDPC code with the same parameters (size and rate). The binary code irregularity is taken from [CRU01] and the parity matrix is build with the PEG algorithm. One can see that most of the performance gain is obtained by going from GF(2) to GF(64), and that GF(256) codes are only interesting if one wants to lower the error floor region. Note that regardless of the decoding complexity, these results are the best presented in the literature, obtained with iteratively decoded codes. The gap with the theoretical limit is quite low, especially if we consider that the sphere-packing bound has not been corrected with the shaping loss in the drawn curves.

1.6.2 High-Order Modulations

Figures 1.30 and 1.31 are taken from [DCG04]. The curves show the values of (E_b/N_0) at which the BER is equal to 10^{-5} for various regular ultra-sparse LDPC codes in GF(256), for which $d_v = 2$. Several values of the check node degree have been considered, $d_c = \{4, 6, 8, 12, 16\}$, to obtain the rates $R = 1 - 2/d_c$. The codeword

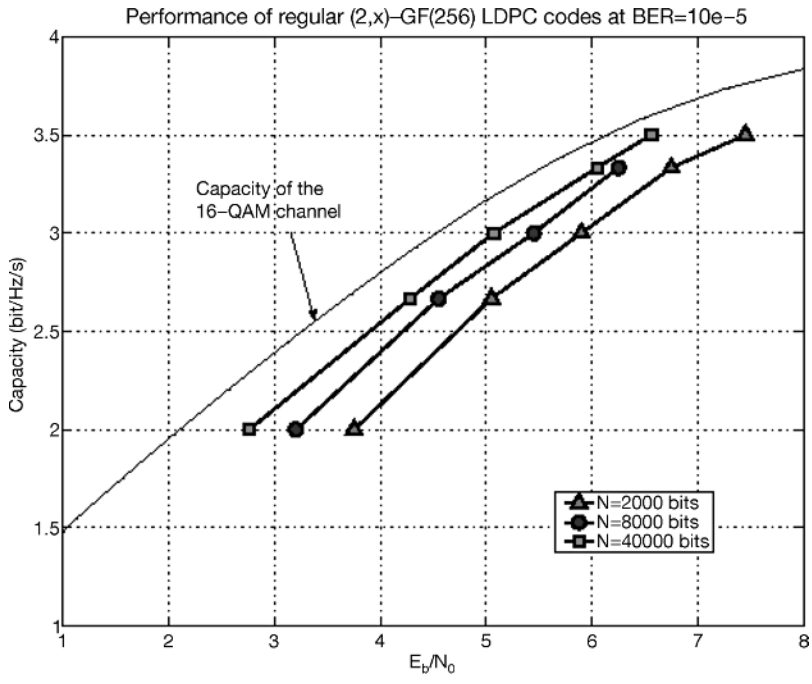

Figure 1.30 Performance of regular GF(256)-LDPC codes for the 16-*QAM* channel at BER $= 10^{-5}$

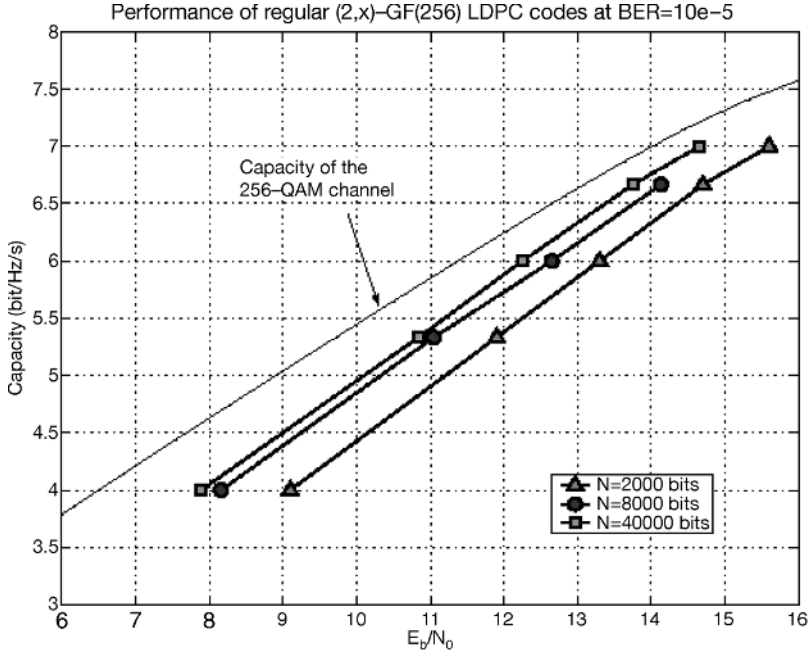

Figure 1.31 Performance of regular GF(256)-LDPC codes for the 256-*QAM* channel at BER $= 10^{-5}$

lengths are $N = 2000$ equivalent bits to $N = 40\,000$ equivalent bits. The SNR loss is measured with respect to the capacity of the AWGN channel *with QAM inputs*. We can see from these figures that, for large codeword lengths, at rate $R = 1/2$, the SNR loss is 0.5 dB for the (16-*QAM*)-AWGN channel and 1.2 dB for the (256-*QAM*)-AWGN channel, which is comparable to the best coding schemes presented in the literature, and at rate $R = 0.833$, the SNR loss is 0.3 dB for the (16-*QAM*)-AWGN channel and 0.7 dB for the (256-*QAM*)-AWGN channel, which is a very good level of performance and shows that when high-order modulation is used, NB-LDPC codes are very good candidates.

This is confirmed for small codeword lengths ($N = 2000$) because the performance loss from $N = 40\,000$ to $N = 2000$ is not very important, as opposed to other coding schemes that suffer greatly when the codeword length is too small (BICM, multilevel coding).

1.6.3 Brief Presentation of NB-LDPC Decoders

The performance improvement of NB-LDPC codes is achieved at the expense of increased decoding complexity. As in all practical coding schemes, an important feature is the complexity/performance tradeoff: it is very important to try to reduce the decoding complexity of NB-LDPC codes, especially for high-order fields GF(q) with $q \geq 64$.

The base decoder of NB-LDPC codes is the BP decoder over the factor graph representation of the code. It differs from the binary BP decoder mainly in that, for GF(q) LDPC codes, the messages from variable nodes to check nodes and from check nodes to variable nodes are defined by q probability weights, or $q - 1$ log-density ratios (LDR). As a result, the complexity of NB-LDPC decoders scales as $O(q^2)$ per check node [WSM04], which prohibits the use of codes built in high-order fields. Computing the check node in the Fourier domain reduces the complexity to $O(q.\log(q))$ per check node [DM98][BD03], but adapting the Fourier domain decoder to practical implementation is tedious due to complicated operators like exponentials or real multiplications.

In [SC03], the authors present a log-domain BP decoder combined with a FFT at the check node input. However, combining log-values and FFT requires a lot of exponential and logarithm computations, which may not be very practical. To overcome this issue, the authors propose the use of a lookup table (LUT) to perform the required operations. Although simple, this approach is of limited interest for codes over high-order fields because the number of LUT accesses grows in $q.\log_2(q)$ for a single message. As a result, for high-order fields, unless the LUT has a prohibitively large size, the performance loss induced by the LUT approximation is quite large. In [SC03], the authors present simulation results for LDPC codes over fields up to GF(16), in which case the LUT approach remains manageable.

Recently, suboptimum decoders based on generalization of the min-sum decoder have been developed [WSM04][WSM04b][DF05][DF06]. The algorithm

that proposes the best complexity/performance tradeoff is the one in [DF05], for which the complexity scales as $O(n_m.q)$ with $n_m \ll q$ and a very small performance degradation compared to the BP decoder. This algorithm is called the extended min-sum (EMS) decoder. However, the complexity of the EMS decoder is still too large to compete with current binary hardware implementations of LDPC codes. New reduced complexity NB-LDPC decoders and parallel hardware models still need to be investigated.

1.7 Three-Dimensional (3D) Turbo Codes

Turbo codes (TCs) are mainly used today in Automatic ReQuest (ARQ) systems, which do not usually require very low error rates. Targeted frame error rates (FER) from 10^{-2} to 10^{-5} are typical for this kind of communication system. However, in future system generations lower FER, down to 10^{-8}, it may be necessary to open the way to real-time and more demanding applications, such as TV broadcasting or videoconferencing. The minimum Hamming distance $d_{\min}$ of a turbo code may not be sufficient to offer such an error correction at the required signal-to-noise ratio. For the current commercial applications of TCs (3G, DVB-RCS, WiMax), commonly based on eight-state component encoders, there are several ways to increase $d_{\min}$ and thereby improve the performance at very low error rates. For instance, one might use stronger component codes, for example 16-state instead of eight-state components [Ber03], at the price of doubling the decoding complexity. Devising more appropriate internal permutations [BSD + 04][CG03] is an appealing alternative to improve $d_{\min}$, because it does not incur any complexity penalty. Unfortunately, designing such powerful permutations is not an easy task and there are limits to the $d_{\min}$ and multiplicity values, and thus to the performance improvements that can be achieved. Another way to improve $d_{\min}$, which has been widely explored in the literature, is to concatenate the component encoders in series rather than in parallel [BDM + 05]. Indeed, thanks to the message passing (turbo) principle, it has become simple today to imagine various coding structures, by concatenating simple component codes, provided that they have a corresponding soft-input/soft-output (SISO) decoder of reasonable complexity. Basically, there are two kinds of concatenation: serial and parallel. Serial concatenation yields higher minimum distances compared to parallel concatenation (turbo codes) but shows a penalty in the convergence threshold, which might be unacceptable for several applications. Hybrid structures, like those proposed in [LNG04][GBK04], are also possible, combining the features of the two concatenations. Finally, multiple concatenations using an increased number of component encoders can be used to eliminate low-weight codewords and so improve the distance properties of the code.

In the following, we address the latter alternative to improve the minimum distance of turbo codes and introduce a three-dimensional turbo code (3D-TC) [BGO07]. The

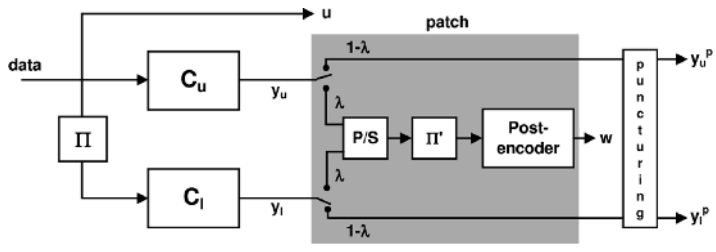

Figure 1.32 The three-dimensional turbo code

encoder structure is depicted in Figure 1.32. The 3D-TC is inspired by the proposals in [LNG04][GBK04] and calls for both parallel and serial concatenation in an original approach: the 3D-TC is simply derived from the classical TC by adding a *partial* rate-1 third dimension. A rate-1 post-encoder is concatenated at the output of the standard turbo code encoder, encoding only a fraction λ of the parity bits stemming from each encoder. The 3D-TC is a very versatile code and provides very low error rates for a wide range of block lengths and coding rates. As will be shown later, it significantly improves performance in the so-called error floor region with respect to the eight-state classical turbo codes, at the expense of a very small increase in complexity (less than 10%). It also compares favorably to more complex codes, such as 16-state turbo codes and the LDPC code of the DVB-S2 standard.

1.7.1 The Encoding Structure

A block diagram of the 3D-TC is depicted in Figure 1.32. The information data sequence $\mathbf{u}$ of length k bits is encoded by a rate-1/3 turbo code consisting of the parallel concatenation of two recursive convolutional codes. We denote by C_u the upper encoder and C_l the lower encoder. The corresponding parity sequences are denoted $\mathbf{y}_u$ and $\mathbf{y}_l$, respectively.

A fraction λ $(0 \leq \lambda \leq 1)$ of the parity bits $\mathbf{y} = \{\mathbf{y}_u, \mathbf{y}_l\}$ stemming from each component encoder are post-encoded by a rate-1, third encoder. We shall refer to λ as the permeability rate. The bits to be post-encoded are chosen in a nonsingular basis. For instance, if $\lambda = 1/4$ the permeability pattern is $\{1000\}$ for both the upper and the lower encoders, i.e., every fourth bit in $\mathbf{y}_u$ and $\mathbf{y}_l$ is post-encoded. The input sequence of the post-encoder is made of alternate $\mathbf{y}_u$ and $\mathbf{y}_l$ (surviving) parity bits. The number of parity bits that are fed to the post-encoder is given by

$$P = 2\lambda k. \tag{1.9}$$

The fraction $1 - \lambda$ of parity bits, which is not encoded, is sent directly to the channel or punctured to achieve the desired code rate. The output of the post-encoder is denoted as $\mathbf{w}$, while $\mathbf{y}_u^p$ and $\mathbf{y}_l^p$ are the punctured versions of $\mathbf{y}_u$ and $\mathbf{y}_l$, respectively. Finally, $\mathbf{u}$, $\mathbf{w}$, $\mathbf{y}_u^p$ and $\mathbf{y}_l^p$ are multiplexed to form the coded sequence $\mathbf{y}$ of length n bits.

Clearly, the overall code rate is given by

$$R = \frac{1}{1 + 2\lambda + 2(1-\lambda)\rho} \tag{1.10}$$

where ρ ($0 \leq \rho \leq 1$) is the fraction of surviving bits in $\mathbf{y}_u$ and $\mathbf{y}_l$ after puncturing. Note that, given λ, and without puncturing information bits, the highest achievable code rate is

$$R = \frac{1}{1 + 2\lambda}. \tag{1.11}$$

For the examples given in this section, we will consider very simple regular or quasi-regular puncturing patterns. For example, if rate-1/2 is sought and $\lambda = 1/4$, then, according to (1.10), $\rho = 1/3$ and the puncturing pattern $\{100\}$ will be applied to $\mathbf{y}_u$ and $\mathbf{y}_l$

The material added to a standard turbo encoder, which will be referred to as the *patch* because it is placed just behind a pre-existing turbo encoder, is composed of:

- a parallel-to-serial (P/S) multiplexer, which takes alternately the parity bits $\mathbf{y}_u$ and $\mathbf{y}_l$, to be encoded, and groups them into a single block of P bits;
- a permutation denoted Π', which permutes the parity bits before feeding them to the post-encoder;
- a rate-1 post-encoder, working on a fraction λ of the parity bits of each component encoder.

This structure combines the features of parallel and serially concatenated codes. In principle, λ can be tuned according to system requirements. Increasing λ turns the code into more serial, while the case $\lambda = 0$ corresponds to the standard parallel turbo code.

The post-encoding principle described above can be applied to any turbo code in a straightforward manner. However, in the sequel, the coding and decoding strategy will be developed on the basis of the double-binary turbo code used in the DVB-RCS standard.

Turbo Code with Double-Binary Components

In another version of the 3D-TC, the same post-encoding principle is applied to the double-binary turbo encoder of the DVB-RCS standard [ETSI00]. The information sequence $\mathbf{u}$ of length k is now grouped into pairs of bits and encoded by a turbo code built from the parallel concatenation of two eight-state recursive systematic convolutional (RSC) codes, with generator polynomials 15 (recursivity), 13 (redundancy), and 7 (second input). Note that now the internal permutation Π deals with messages of $N = k/2$ symbols. An intra-symbol permutation is also adopted to improve the minimum distance of the turbo code [DB05]. The code rate is 1/2. The post-encoding

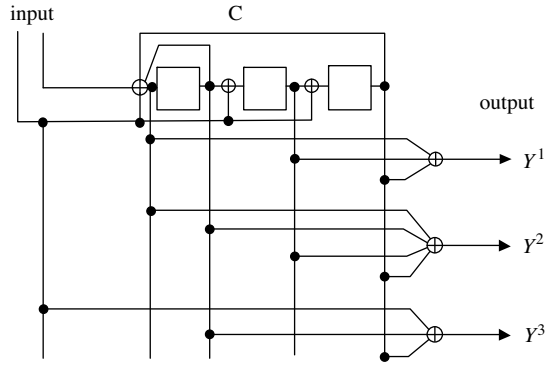

Figure 1.33 The component double-binary encoder with three outputs

principle is the same as previously, but the number of post-encoded bits is now

$$P = 2\lambda N = \lambda k \tag{1.12}$$

while the rate of the 3D-TC is

$$R = \frac{1}{1+\lambda+(1-\lambda)\rho}, \tag{1.13}$$

and the highest achievable code rate (for a given λ and no puncturing of the information bits) is

$$R_{\max} = \frac{1}{1+\lambda}. \tag{1.14}$$

Note that higher code rates might be achieved by puncturing systematic bits. On the other hand, the lowest code rate is given by the rate of the double-binary TC, i.e., $R \leq 1/2$. If lower rates are sought, higher-rate component double-binary encoders must be considered. For instance, for overall rate 1/4 the component encoders need to generate two extra parity bits. Figure 1.33 shows the block diagram of the best eight-state encoder, which provides three outputs.

1.7.2 Code Optimization

Given the parent turbo code (the DVB-RCS turbo code here) and the interleaving laws for Π and Π', the performance of the 3D-TC depends on the post-encoder and the permeability rate λ, which must be properly optimized.

The choice of the permeability rate

The choice of λ is a matter of tradeoff between the convergence loss and the required $d_{\min}$. Convergence designates the zone of the error rate versus signal-to-noise ratio

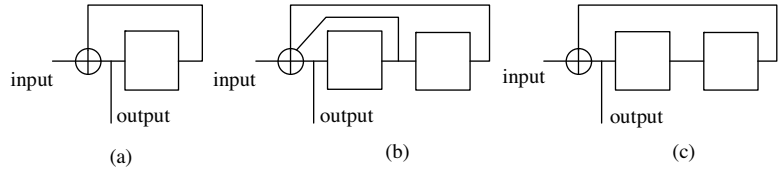

Figure 1.34 Possible candidates, with memory 1 or 2, to become the rate-1 post-encoder

E_b/N_0 curve where the error rate begins to decrease noticeably. Choosing a large value of λ penalizes the decoder from the convergence point of view. This results from the decoder associated with the post-encoder, which does not benefit from any redundant information at the first iteration and therefore multiplies the errors during the first processing. Let us assume for instance that the post-encoder is the well-known *accumulate* code (convolutional code with memory 1), depicted in Figure 1.34(a). The associated decoder (the pre-decoder), without any extra information, doubles the errors at its input. From (1.10), the fraction θ of the codeword bits that are post-encoded bits is

$$\theta = \frac{P}{n} = \lambda R. \tag{1.15}$$

The fraction θ_q of the data processed by the component decoder of each code C_q ($q = u$, l) that is processed by the pre-decoder is

$$\theta_q = \frac{\lambda R}{1+R}. \tag{1.16}$$

Then, if p is the probability of error at the channel output, the average probability of error p' at each decoder intrinsic input is

$$p' = 2\theta_q p + (1-\theta_q)p = (1+\theta_q)p. \tag{1.17}$$

From (1.16) we have:

$$p' = \left(\frac{1+(1+\lambda)R}{1+R}\right)p. \tag{1.18}$$

In other words, the probability of error at each decoder intrinsic input is raised by a factor

$$\frac{1+(1+\lambda)R}{1+R}$$

inducing a loss in convergence.

The strategy for choosing the value of λ arises directly from (1.18):

1. From a given acceptable convergence loss and from the curve $p(E_b/N_0)$ (for instance, erfc(x) for a Gaussian channel), infer the value of $\frac{p'}{p} = \frac{1+(1+\lambda)R}{1+R}$.
2. For a given coding rate, deduce the value of λ.
3. If the resulting MHD is not sufficient, increase p' and go to 1.

The Choice of the Post-encoder
The post-encoder has to meet the following requirements:

1. Its decoder must be simple, adding little complexity to the classical turbo decoder, while being able to handle soft-in and soft-out information.
2. In order to prevent the decoder suffering from any side-effects, because very low error rates are sought, the post-code has to be a homogeneous block code.
3. At the first iteration (without any redundant input information), the pre-decoder associated with the rate-1 post-encoder must not exhibit too much error amplification.

Possible candidates, low-memory RSC codes, which satisfy condition 1, are given in Figure 1.34. Condition 2 is compatible with the use of circular RSC (CRSC) codes having memory 2. Circular convolutional codes (also called tail-biting codes) are such that any state of the encoder register is allowed as the initial state and the encoding always starts and ends in the same state. This makes the convolutional code a perfect block code and prevents it from any side-effects. Moreover, no rate loss is induced by terminating the code. Circular CCs have already been adopted in the DVB-RCS turbo code. Note that the code with memory 1 (the accumulate code, Figure 1.34(a)) cannot be made circular using standard circular termination, and has to be discarded. Code (b) in Figure 1.34 can easily be made circular, provided that the number of bits to be encoded is not a multiple of 3. On the other hand, at the first step of the iterative process, its decoder will (roughly) triple the number of errors of its input. Finally, code (c) has a corresponding decoder which only doubles the number of errors at the first step, but it cannot be made circular directly. However, a simple trick will allow us to use this code as a CRSC code, as explained below.

Circular (Tail-Biting) Encoding
Let $\mathbf{s}_i$ and $\mathbf{d}_i$ be the state of the encoder register and the encoder data input, respectively, at discrete time i. The encoder state at time $i+1$ is given by the following equation:

$$\mathbf{s}_{i+1} = \mathbf{G}\mathbf{s}_i + \mathbf{d}_i \tag{1.19}$$

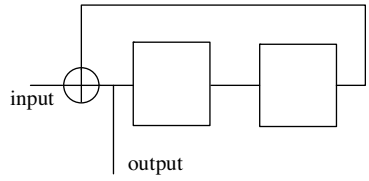

Figure 1.35 Linear feedback register with memory 2 and recursivity polynomial 7 (in octal)

where $\mathbf{G}$ is the generator matrix of the linear feedback register (LFR). For instance, considering the LFR in Figure 1.35, we have:

$$\mathbf{s}_i = \begin{bmatrix} \mathbf{s}_{1,i} \\ \mathbf{s}_{2,i} \end{bmatrix}; \ \mathbf{d}_i = \begin{bmatrix} \mathbf{d}_i \\ 0 \end{bmatrix}; \ \mathbf{G} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \tag{1.20}$$

More generally, for a memory v register, vectors $\mathbf{s}_i$ and $\mathbf{d}_i$ have v components and $\mathbf{G}$ is of size $v \times v$. After the encoding of data sequence $\{\mathbf{d}_i\}$, of length P, the final state $\mathbf{s}_P$can be expressed as a function of the initial state $\mathbf{s}_0$ and $\{\mathbf{d}_i\}$:

$$\mathbf{s}_P = \mathbf{G}^P \mathbf{s}_0 + \sum_{j=1}^{P} \mathbf{G}^{P-j} \mathbf{d}_{j-1}. \tag{1.21}$$

If it is possible to find a circulation state, denoted $\mathbf{s}^c$, such that $\mathbf{s}^c = \mathbf{s}_0 = \mathbf{s}_P$, this is given by:

$$\mathbf{s}^c = [\mathbf{I} + \mathbf{G}^P]^{-1} \sum_{j=1}^{P} \mathbf{G}^{P-j} \mathbf{d}_{j-1} \tag{1.22}$$

where $\mathbf{I}$ is the $v \times v$ identity matrix.

Note that $\mathbf{s}^c$ exists if $\mathbf{I} + \mathbf{G}^P$ is invertible. This condition is never satisfied for some matrices $\mathbf{G}$, whatever the value of P. This is the case for the encoders in Figure 1.34(a, c), which have $\mathbf{G} = [1]$ and $\mathbf{G} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, respectively. For other matrices, $\mathbf{I} + \mathbf{G}^P$ is invertible if P is not a multiple of the period L of the LFR, defined by $\mathbf{G}^L = \mathbf{I}$. For instance, the LFR in Figure 1.34(b) has $L = 3$. Therefore, $\mathbf{I} + \mathbf{G}^P$ is not invertible for $P = 3t$, with t an integer. In such cases, the encoder cannot directly be made circular.

Before the encoding of $\{\mathbf{d}_i\}$, the knowledge of $\mathbf{s}^c$ requires a preliminary step. The encoder is first set up in the zero state and then fed by the data sequence. The final state is denoted $\mathbf{s}_0^p$. From (1.21), we have

$$\mathbf{s}_P^0 = \sum_{j=1}^{P} \mathbf{G}^{P-j} \mathbf{d}_{j-1} \tag{1.23}$$

and, from (1.17), $\mathbf{s}^c$ can be related to $\mathbf{s}_0^p$ by

$$\mathbf{s}^c = \left[\mathbf{I} + \mathbf{G}^P\right]^{-1} \mathbf{s}_P^0. \tag{1.24}$$

Finally, the encoder being initialized in the circulation state, the encoding process can really start to provide the redundant sequence.

State Mapping Encoding
State mapping encoding may be introduced for cases where standard circular (tail-biting) encoding is not possible (i.e., $\mathbf{I} + \mathbf{G}^P$ is not invertible). The core of this encoding is a mapping that maps the final state $\mathbf{s}_P$ to the state $\mathbf{s}'_P = \mathbf{A}\mathbf{s}_P$ using a state-mapping matrix $\mathbf{A}$. Mapping the final state given by (1.21) yields the equation

$$\mathbf{s}_P^0 = \mathbf{A}\mathbf{G}^P \mathbf{s}_0 + \mathbf{A} \sum_{j=1}^{P} \mathbf{G}^{P-j} \mathbf{d}_{j-1}. \tag{1.25}$$

A mapping state $\mathbf{s}^m$ with $\mathbf{s}^m = \mathbf{s}_0 = \mathbf{s}'_P$ always exists, and is given by

$$\mathbf{s}^m = \mathbf{B} \sum_{j=1}^{P} \mathbf{G}^{P-j} \mathbf{d}_{j-1} = \mathbf{B}\mathbf{s}_P^0 \tag{1.26}$$

with

$$\mathbf{B} = \left[\mathbf{I} + \mathbf{A}\mathbf{G}^P\right]^{-1} \mathbf{A}. \tag{1.27}$$

In other words, if the encoding starts in the state $\mathbf{s}^m$, the encoding will end in the state $\mathbf{s}^e$ with $\mathbf{s}^m = \mathbf{A}\mathbf{s}^e$. The encoding procedure can be summarized in the following steps:

1. Set up the encoder in the zero state. Feed it with $\{\mathbf{d}_i\}$ and take the final state $\mathbf{s}_P^0$.
2. Calculate $\mathbf{s}^m$ through (1.26) and (1.27).
3. Encode $\{\mathbf{d}_i\}$ starting from $\mathbf{s}^m$. If needed, map the final state $\mathbf{s}^e$ using $\mathbf{A}$, in order to verify that the result is $\mathbf{s}^m$ (i.e., $\mathbf{s}^m = \mathbf{A}\mathbf{s}^e$).

The encoder of Figure 1.34(c), with generator polynomial 5, can be encoded using:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \quad \text{if } P \text{ is odd}$$
$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \quad \text{if } P \text{ is even} \tag{1.28}$$

Table 1.4 Corresponding values of $\mathbf{s}_P$ and $\mathbf{s}'_P$ and of $\mathbf{s}^0_P$ and $\mathbf{s}^m$

P odd		P even		P odd		P even	
s_P	s'_P	s_P	s'_P	s^0_P	s^m	s^0_P	s^m
0	0	0	0	0	0	0	0
1	3	1	2	1	1	1	3
2	2	2	3	2	3	2	1
3	1	3	1	3	2	3	2

The decoding process has to take into account the mapping described above. This is done by an exchange of metrics after having processed the last address $i = P - 1$, during the forward recursion, and after having processed the first address $i = 0$, during the backward recursion, when the MAP algorithm, or a simplified version, is employed. Table 1.4 provides the values of $\mathbf{s}'_P$ obtained through the mapping of $\mathbf{s}_P$. The table also provides the values of $\mathbf{s}^m$ for each $\mathbf{s}^0_P$ using (1.26) and (1.27). We can observe that only 2 (if P is odd) or 3 (if P is even) metrics need to be swapped during the decoding process, at the extremity of the block, which represents a very small additional complexity for the four-state decoder.

1.7.3 Decoding the 3D Turbo Code

The decoding of the 3D-TC calls for the classical turbo principle. The decoder is shown in Figure 1.36. The decoder consists of three SISO decoders: two eight-state SISO

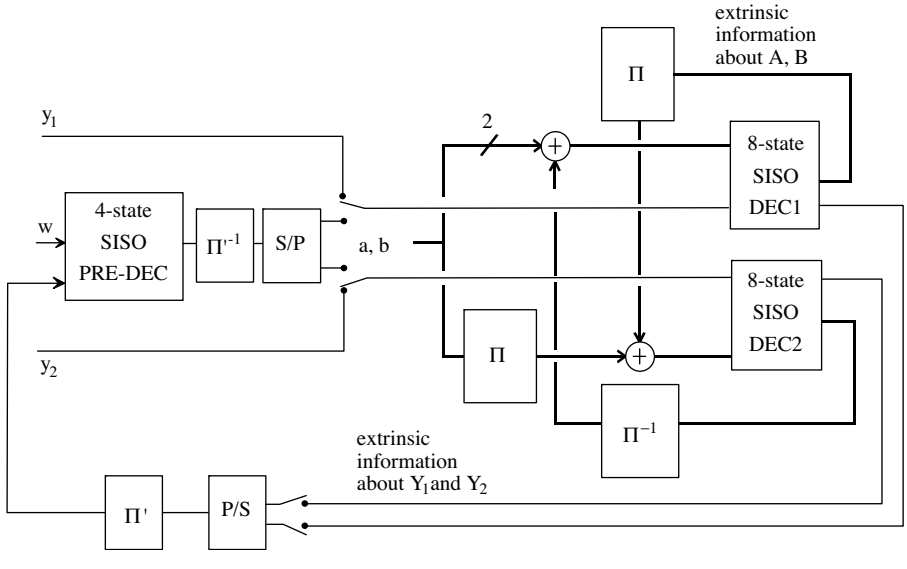

Figure 1.36 Linear block diagram of the 3D turbo decoder

decoders matched to the upper and lower encoder of the double-binary turbo code, denoted $\mathrm{DEC_u}$ and $\mathrm{DEC_l}$, respectively, and a four-state SISO decoder (the pre-decoder) to decode the post-encoder. As usual TCs, $\mathrm{DEC_u}$ and $\mathrm{DEC_l}$ exchange extrinsic information on the systematic symbols of the received codeword. They must also provide the four-state SISO pre-decoder with extrinsic information on the post-encoded parity bits. In turn, the pre-decoder feeds $\mathrm{DEC_u}$ and $\mathrm{DEC_l}$ with extrinsic information on these parity bits.

Because $\mathrm{DEC_u}$ and $\mathrm{DEC_l}$ are quaternary eight-state decoders processing $N = k/2$ pairs of bits and the pre-decoder is a binary four-state decoder processing $P = \lambda k$ data, the relative computational complexity added by the latter is very small. For instance, with $\lambda = 1/4$ (the largest value considered in this chapter), the additional complexity is roughly 6%. However, some extra functions must be added to the classical turbo decoder, the main one being the calculation of the extrinsic information on parity bits to be fed to the pre-decoder. Overall, the additional complexity, compared to the classical turbo decoder, is less than 10% for $\lambda = 1/4$.

1.7.4 Simulation Results

In Figures 1.37 and 1.38, we report frame error rate results for two typical block sizes, 188 and 57 bytes, respectively, and coding rates 1/4, 1/2 and 4/5. In all these simulations $\lambda = 1/4$ and a maximum of eight iterations is assumed. Note that, since $\lambda = 1/4$, no puncturing of information bits is assumed, the maximum achievable coding rate is $R_{\max} = 4/5$ (see 1.14). All simulations assume a permutation Π of the ARP type [BSD + 04] and a regular permutation for Π' [BGO07].

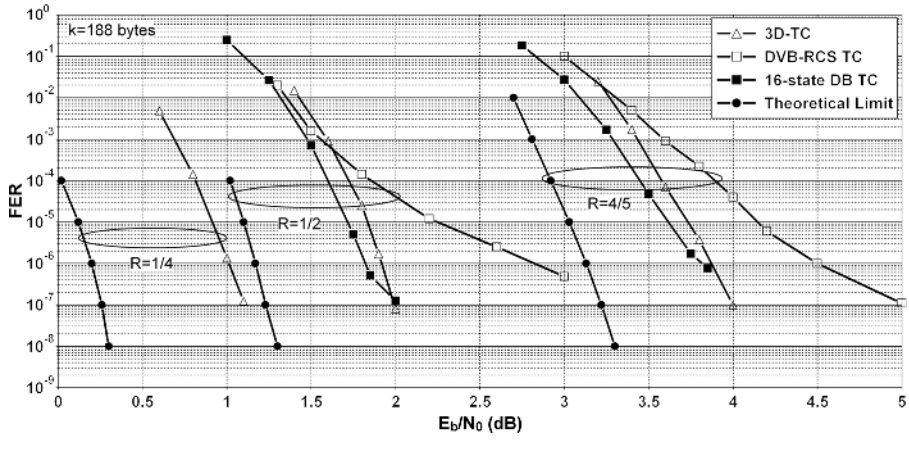

Figure 1.37 Frame error rate performance of the 3D-TC with $\lambda = 1/4$ for $k = 188$ bytes and several rates

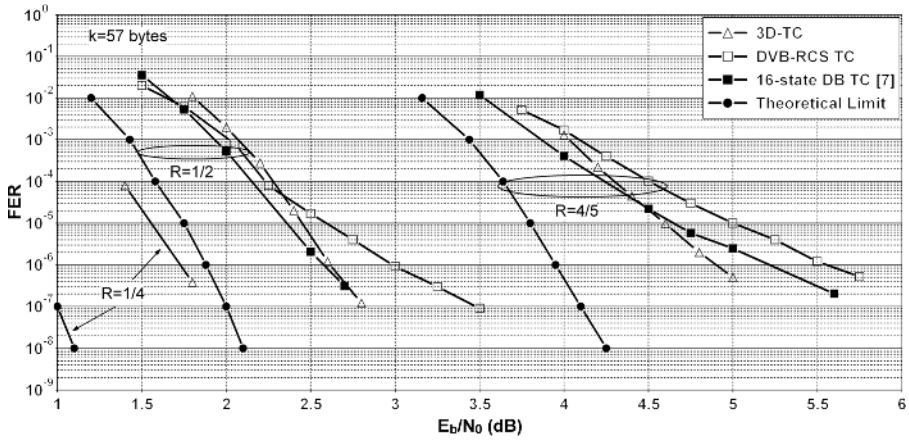

Figure 1.38 Frame error rate performance of the 3D-TC with $\lambda = 1/4$ for $k = 57$ bytes and several rates

The 3D-TC code shows excellent performance for both short and medium block sizes. In particular, for information block size 188 bytes only 0.8 dB loss is observed with respect to Gallager's random coding bound at FER$\sim 10^{-7}$ for all code rates. For comparison purpose, the performance of the original DVB-RCS TC is also reported for rates 1/2 and 4/5. For rate 1/2 the 3D-TC shows a small convergence loss with respect to the DVB-RCS TC, which is explained by the reasoning in Section 1.7.2. On the other hand, the error floor is significantly lowered. The largest gain is obtained for 188 bytes and $R = 1/2$ (about 1.4 dB at FER $= 10^{-7}$). For rate 4/5, the convergence loss is reduced while a significant improvement for low error rates is also observed.

We also report in Figures 1.37 and 1.38 the performance of the 16-state double-binary TC described in [DB05] for rates 1/2 and 4/5. The performance of the proposed 3D-TC is comparable to that of the more complex 16-state TC. For a block length of 188 bytes, the 3D-TC loses 0.1 dB in convergence with respect to the 16-state double-binary turbo code. However, the 3D-TC outperforms the 16-state TC in the error floor. Similar behavior is observed in [BGO07] for a block length of 57 bytes.

The 3D-TC also shows very good performance for large block lengths. In Figure 1.39 the bit error rate performance of the 3D-TC is compared with that of the DVB-S2 standard LDPC code [ETSI05] for coding rates 1/2 and 8/9 and a coded block length of 8000 bytes. The performance of the LDPC code was obtained from an FPGA, and is it very close to simulated performance. Fifty decoding iterations are assumed. Here, $\lambda = 1/8$ and 12 iterations are assumed for the 3D-TC. Similar performances were observed for the two codes.

Finally, Figure 1.40 compares the performance with respect to the eight-state TC adopted in the 3GPP2 standard. An information block length of 12 288 bits and eight iterations is assumed for the two codes. Very similar performance is observed in the

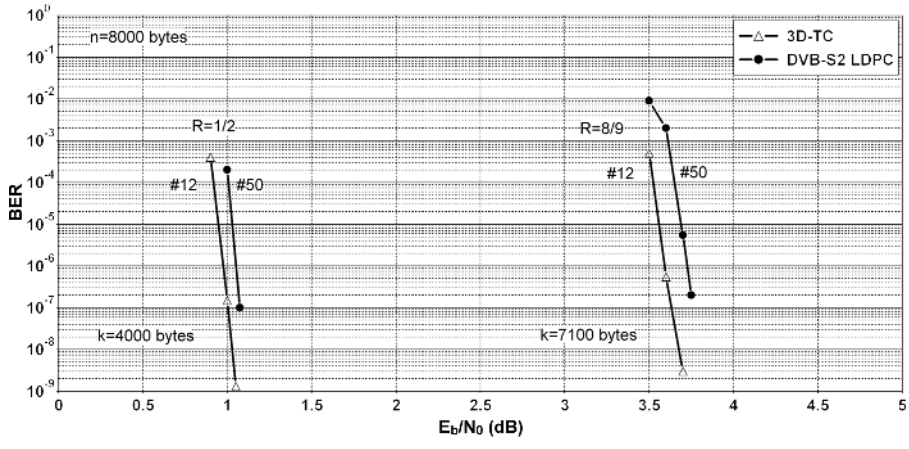

Figure 1.39 Block error rate performance of the 3D-TC with $\lambda = 1/8$ for $n = 8000$ bytes and comparison with the DVB-S2 LDPC code

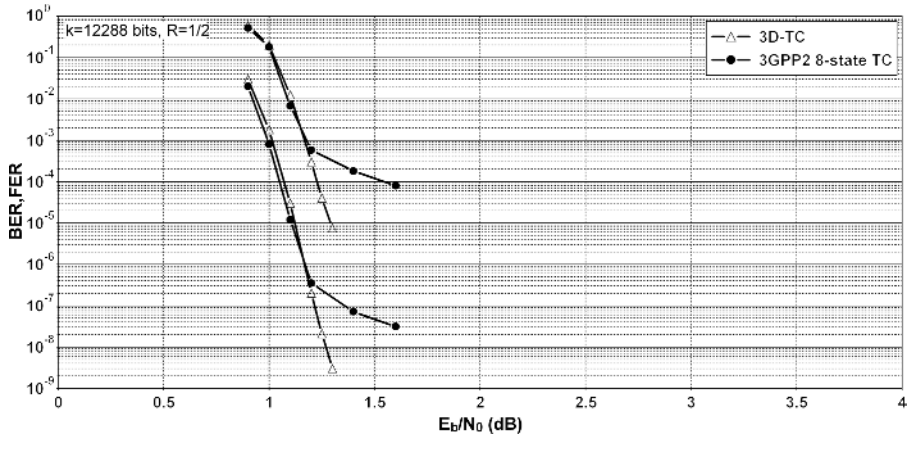

Figure 1.40 Block error rate and FER performance of the 3D-TC with $\lambda = 1/8$ for $k = 12\,288$ bits, $R = 1/2$ and comparison with the 3GPP2 turbo code

waterfall region. However, the 3D-TC significantly improves the 3GPP2 code in the error floor region. No flattening is observed at FER $= 10^{-5}$.

1.8 Conclusions

In this section, a modified turbo code combining the features of parallel and serial concatenation in order to obtain increased Hamming minimum distances with respect to classical turbo codes has been discussed. The simulation results corroborate

the interest of this approach. Frame error rates down to 10^{-7} are obtained near the theoretical limits without the use of any outer block code, such as BCH or Reed–Solomon codes. This characteristic makes the 3D turbo code very versatile from the standpoint of block size and coding rate. Furthermore, the component decoding algorithm (max-log-MAP) is simple and does not require knowledge of the channel noise variance. Finally, the internal permutations of the 3D-TC are based on very simple models enabling large degrees of parallelism, if needed.

References

[ADT05] K. Andrews, S. Dolinar, and J. Thorpe (2005) "Encoders for block-circulant LDPC codes," *Proceedings of IEEE Information Symposium on Information Theory*, pp. 2300–2304.

[AKB04] A. Ashikhmin, G. Kramer, and S. ten Brink (2004) "Extrinsic information transfer functions: model and erasure channel properties," *IEEE Trans. Inf. Theory*, vol. 50, no. 11, pp. 2657–2673.

[B+05] S. Benedetto, R. Garello, G. Montorsi, C. Berrou, C. Douillard, D. Giancristofaro, A. Ginesi, L. Giugno, and M. Luise (2005) "MHOMS: High speed ACM modem for satellite applications," *IEEE Wireless Commun.*, vol. 12, no. 2, pp. 66–77.

[BB06] A. Bennatan and D. Burshtein (2006) "Design and analysis of nonbinary LDPC codes for arbitrary discrete-memoryless channels," *IEEE Trans. Inf. Theory*, vol. 52, no. 2, pp. 549–583.

[BD03] L. Barnault and D. Declercq (2003) "Fast decoding algorithm for LDPC over GF(2q)," *Proceedings of Information Theory Workshop (ITW'03), IEEE*, pp. 70–73, Paris, France, March.

[BDM+05] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara (2005) "Serial concatenation of interleaved codes: Performance analysis, design and iterative decoding," *IEEE Trans. Inf. Theory*, vol. 44, pp. 909–926.

[Ber03] C. Berrou (2003) "The ten-year-old turbo codes are entering into service," *Proc. IEEE Commun. Mag.*, pp. 110–116.

[BGO07] C. Berrou, A. Graell i Amat, Y. Ould Cheikh Mouhamedou, C. Douillard, and Y. Saouter (2007) "Adding a rate-1 third dimension to turbo codes," *Proceedings of IEEE Information Theory Workshop (ITW'07),* Tahoe City, USA, pp. 156–161, September.

[BH02] A. Blanksby and C.J. Howland (2002) "A 690-mW 1-Gb/s, rate-1/2 low-density parity-check code decoder," *IEEE J. Solid-State Circuits*, vol. 37, no. 3, pp. 404–412.

[BK03] S. ten Brink and G. Kramer (2003) "Design of repeat-accumulate codes for iterative detection and decoding," *IEEE Trans. Signal Process.*, vol. 51, no. 11, pp. 2764–2772.

[BKA04] S. ten Brink, G. Kramer, and A. Ashikhmin (2004) "Design of low-density parity-check codes for modulation and detection," *IEEE Trans. Commun.*, vol. 52, no. 4, pp. 670–678.

[BSD+04] C. Berrou, Y. Saouter, C. Douillard, S. Kerouédan, and M. Jézéquel (2004) "Designing good permutations for turbo codes: towards a single model," *Proceedings of IEEE International Conference on Communications (ICC'04),* Paris, France, pp. 341–345, June.

[BT05] G. Byers and F. Takawira (2005) "EXIT charts for non-binary LDPC codes", *Proceedings of International Conference on Communications (ICC'05), IEEE*, vol. 1, pp. 652–657, May.

[CG03] S. Crozier and P. Guinand (2003) "Distance upper bounds and true minimum distance results for turbo-codes designed with DRP interleavers," *Proceedings of 3rd International Symposium on Turbo Codes*, pp. 169–172, September.

[CRU01] S.Y. Chung, T. Richardson, and R. Urbanke (2001) "Analysis of sum–product decoding of LDPC codes using a Gaussian approximation," *IEEE Trans. Inf. Theory*, vol. 47, pp. 657–670.

[DB05] C. Douillard and C. Berrou (2005) "Turbo codes with rate-m/(m + 1) constituent convolutional codes," *Proc. IEEE Trans. Commun.*, vol. 53, no. 10, pp. 1630–1638.

[DCG04] D. Declercq, M. Colas, and G. Gelle (2004) "Regular GF(2q)-LDPC coded modulations for higher order QAM-AWGN channels," *Proceedings of International Symposium on Information Theory and its Applications (ISITA'04),* IEEE, Parma, Italy, October.

[DF05] D. Declercq and M. Fossorier (2005) "Extended MinSum algorithm for decoding LDPC codes over GF (q)," *Proceedings of International Symposium on Information Theory (ISIT'05),* IEEE, Adelaide, Australia, September.

[DF06] D. Declercq and M. Fossorier (2006) "Decoding algorithms for nonbinary LDPC codes over GF(q)," *Proc. IEEE Trans. Commun.*, vol. 55, no. 4, pp. 633–643.

[DM98] M. Davey and D.J.C. MacKay (1998) "Low density parity check codes over GF(q)," *IEEE Commun. Lett.*, vol. 2, pp. 165–167.

[DP95] D. Divsalar and F. Pollara (1995) "Turbo codes for PCS applications," *Proceedings of International Conference on Communications,* Seattle, WA, June.

[ETSI00] European Telecommunications Standards Institute (ETSI) (2000) "Digital Video Broadcasting (DVB), Interaction channel for satellite distribution systems," ETSI, EN 301 790, V1.2.2, pp. 21–24, December.

[ETSI05] European Telecommunications Standards Institute (ETSI) (2005) "Digital Video Broadcasting (DVB), Second generation framing structure for broadband satellite applications," ETSI, EN 302 307, V1.1.1.

[Fos04] M.P. Fossorier (2004) "Quasi-cyclic low-density parity-check codes from circulant permutation matrices," *IEEE Trans. Inf. Theory*, vol. IT-50, pp. 1788–1793.

[Gal62] R.G. Gallager (1962) "Low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 8, pp. 21–28.

[Gal63] R.G. Gallager (1963) *Low-Density Parity-Check Codes*, Cambridge, MA: MIT Press.

[GBK04] H. Gonzalez, C. Berrou, and S. Kerouédan (2004) "Serial/parallel turbo codes for low error rates," *Proceedings of IEEE International Conference on Communications (ICC'04),* Paris, France, pp. 346–350, June.

[HE04] X.-Y. Hu and E. Eleftheriou (2004) "Binary representation of cycle Tanner-graph GF(2q) codes," *Proceedings of International Conference on Communications (ICC'04),* IEEE, vol. 1, pp. 528–532, Paris, France, June.

[HEA01] X. Hu, E. Eleftheriou, and D. Arnold (2001) "Progressive edge-growth Tanner graphs," *Proceedings of 2001 Global Telecommunications Conference (Globecom'01),* San Antonio, TX.

[Hu02] X.-Y. Hu (2002) "Low-delay low-complexity error-correcting codes on sparse graphs," PhD Thesis, EPFL.

[Jim06] A. Jiménez-Feltström (2006) "Iteratively decodable convolutional codes: analysis and implementation aspects," PhD thesis, Lund University.

[JZ99] A. Jiménez-Feltström and K.Sh. Zigangirov (1999) "Time-varying periodic convolutional codes with low-density parity-check matrix," *IEEE Trans. Inf. Theory*, vol. 45, no. 6, pp. 2181–2191.

[KGP06] D. Kimura, F. Guilloud, and R. Pyndiah (2006) "Construction of parity-check matrices for non-binary LDPC codes," *Proceedings of 4th International Symposium on Turbo Codes and Related Topics,* Munich, April.

[LFK03] G. Li, I. Fair, and W. Krzymien (2003) "Analysis of nonbinary LDPC codes using Gaussian approximation," *Proceedings of International Symposium on Information Theory (ISIT'03),* IEEE, Yokohama, Japan, July.

[LNG04] J. Li, K.R. Narayanan, and C.N. Georghiades (2004) "Product accumulate codes: a class of codes with near-capacity performance and low decoding complexity," *IEEE Trans. Inf. Theory*, vol. 50, no. 1, pp. 31–46.

[LSL06] G. Lechner, J. Sayir, and I. Land (2006) "Optimization of LDPC codes for receiver frontends," *2006 IEEE International Symposium on Information Theory,* Seattle, USA, July 9–14.

[MacBib] D.J.C. MacKay,Online database of low-density parity-check codes, www.inference.phy.cam.ac.uk/mackay/codes/data.html.

[MD99] D.J.C. MacKay and M. Davey (1999) "Evaluation of Gallager codes for short block length and high rate applications," *Proceedings of IMA Workshop on Codes, Systems and Graphical Models.*

[Nim04] A. Nimbalker, T.K. Blankenship, B. Classon, T.E. Fuja, and D.J. Costello, Jr. (2004) "Contention-free interleavers", *Proceedings of IEEE International Symposium on Information Theory,* Chicago, IL, p. 54.

[PFD06a] C. Poulliat, M. Fossorier, and D. Declercq (2006) "Using binary image of nonbinary LDPC codes to improve overall performance," *Proceedings of International Symposium on Turbo Codes and Related Topics,* Munich, April.

[PFD06b] C. Poulliat, M. Fossorier, and D. Declercq (2006) "Design of non binary LDPC codes using their binary images: algebraic properties," *Proceedings of International Symposium on Information Theory (ISIT'06),* IEEE, Seattle, WA, pp. 93–97, July.

[RSU01] T.J. Richardson, M.A. Shokrollahi, and R.L. Urbanke (2001) "Design of capacity-approaching irregular low density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 619–637.

[RT05] J. Ryu and O.Y. Takeshita,"On quadratic inverses for quadratic permutation polynomials over integer rings," submitted to *IEEE Trans. Inf. Theory*.

[RU01] T.J. Richardson and R.L. Urbanke (2001) "Efficient encoding of low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 638–656.

[RU03] T. Richardson and R. Urbanke (2003) "The renaissance of Gallager's low-density parity-check codes," *IEEE Commun. Mag.*, vol. 41, no. 8, pp. 126–131.

[SC03] H. Song and J.R. Cruz (2003) "Reduced-complexity decoding of Q-ary LDPC codes for magnetic recording," *IEEE Trans. Magn.*, vol. 39, pp. 1081–1087.

[SF02] D. Sridhara and T.E. Fuja (2002) "Low density parity check codes over groups and rings," *Proceedings of ITW'02,* Bangalore, India, pp. 163–166, October.

[ST05] J. Sun and O.Y. Takeshita (2005) "Interleavers for turbo codes using permutation polynomials over integer rings," *IEEE Trans. Inf. Theory*, vol. IT-51, pp. 101–119.

[SV04] R. Smarandache and P.O. Vontobel (2004) "On regular quasi-cyclic LDPC codes from binomials," *Proceedings of IEEE International Symposium on Information Theory,* Chicago, IL, June.

[SZL+06] S. Song, L. Zeng, S. Lin, and K. Abdel-Ghaffar (2006) "Algebraic constructions of nonbinary quasi-cyclic LDPC codes," *Proceedings of International Symposium on Information Theory (ISIT'06),* IEEE, Seattle, WA, pp. 1303–1308, July.

[Tak05] O.Y. Takeshita (2006) "On maximum contention-free interleavers and permutation polynomials over integer rings," *IEEE Trans. Inf. Theory*, vol. 52, no. 3, pp. 1249–1253, March.

[Tan00] R.M. Tanner (2000) "A [155, 64, 20] sparse graph (LDPC) code," presented at the IEEE International Symposium on Information Theory, Sorrento, Italy, June.

[Tan04] R.M. Tanner, D. Sridhara, A. Sridharan, T.E. Fuja, and D.J. Costello, Jr. (2004) "LDPC block and convolutional codes based on circulant matrices," *IEEE Trans. Inf. Theory*, vol. IT-50, pp. 2966–2984.

[Tan88] R.M. Tanner (1988) "A transform theory for a class of group-invariant codes," *IEEE Trans. Inf. Theory*, vol. IT-34, pp. 752–775.

[TBM04] A. Tarable, S. Benedetto, and G. Montorsi (2004) "Mapping interleaving laws to parallel turbo and LDPC decoder architectures," *IEEE Trans. Inf. Theory*, vol. IT-50, pp. 2002–2009.

[WSM04] H. Wymeersch, H. Steendam, and M. Moeneclaey (2004) "Log-domain decoding of LDPC codes over GF(q)," *Proceedings of International Conference on Communications (ICC'04),* IEEE, Paris, France, pp. 772–776, June.

[WSM04b] H. Wymeersch, H. Steendam, and M. Moeneclaey (2004) "Computational complexity and quantization effects of decoding algorithms of LDPC codes over GF(q)," *Proceedings of ICASSP-2004,* Montreal, Canada, pp. 772–776, May.