Chapter 3

Low-Density Parity-Check Codes

3.1 Introduction

In this chapter, we provide the reader with an overview on low-density parity-check (LDPC) codes. The concepts outlined in this chapter will then be used througout the remainder of the book.

This chapter is structured as follows. In Section 3.2 a basic description of LPDC codes, together with their graphical representation, is given. In Section 3.3, LDPC codes are described through a statistical approach, which allows to derive significant insights into the behavior of these codes in a tractable manner. In Section 3.4, possible decoding algorithms are presented. Section 3.5 presents LDPC code design techniques based on the use of the statistical description introduced in Section 3.3. Finally, encoding techniques are presented in Section 3.6 and conclusions are drawn in Section 3.7.

3.2 Description of LDPC Codes

LDPC codes were first introduced by R. Gallager in his Ph.D. thesis [9]. In their first instance, LDPC codes are linear block codes characterized by a *sparse* parity-check matrix H whose columns have a fixed number d_v of non-zero elements and whose rows have a fixed number d_c of non-zero elements. The following matrix gives an example of a possible LDPC code parity check

M. Franceschini, G. Ferrari and R. Raheli, *LDPC Coded Modulations,*
DOI: 10.1007/978-3-540-69457-1_3, © Springer-Verlag Berlin Heidelberg 2009

matrix with $d_v = 3$ and $d_c = 6$:

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} . \quad (3.1)$$

In the current literature such codes are referred to as *regular* (d_v, d_c) LDPC codes. Let N be the number of columns of the parity check matrix, i.e., N is the length of the codeword. Let $M < N$ be the number of rows of the parity check matrix and assume that the parity check matrix H is maximum rank, i.e., $\mathrm{Rank}(H) = M$. Considering a column vector $\boldsymbol{x}$ of N elements, the number of degrees of freedom in the solution of the parity check equation

$$H\boldsymbol{x} = 0 \quad (3.2)$$

i.e., the number of linearly independent columns of H, is equal to $K = N - M$, which also corresponds to the number of information bits in a codeword. Of course, the sum of all ones in the rows equals the sum of all ones in the columns. Therefore

$$Nd_v = Md_c = (N - K)d_c$$

which yields

$$\frac{K}{N} = 1 - \frac{d_v}{d_c} . \quad (3.3)$$

The term $R = K/N$ in (3.3) is the so called *code rate*, i.e. the average number of information bits per codeword binary symbol. Therefore, a regular (d_v, d_c) LDPC code has code rate $R = 1 - d_v/d_c$.

It is possible to associate H with a bipartite graph in one-to-one correspondence. Such a graph contains two kinds of nodes: each node of the first kind, denoted as *variable node*, is associated with a column of H; each node of the second kind, denoted as *check node*, is associated with a row of H. The bipartite graph associated with the parity check matrix is constructed as follows:

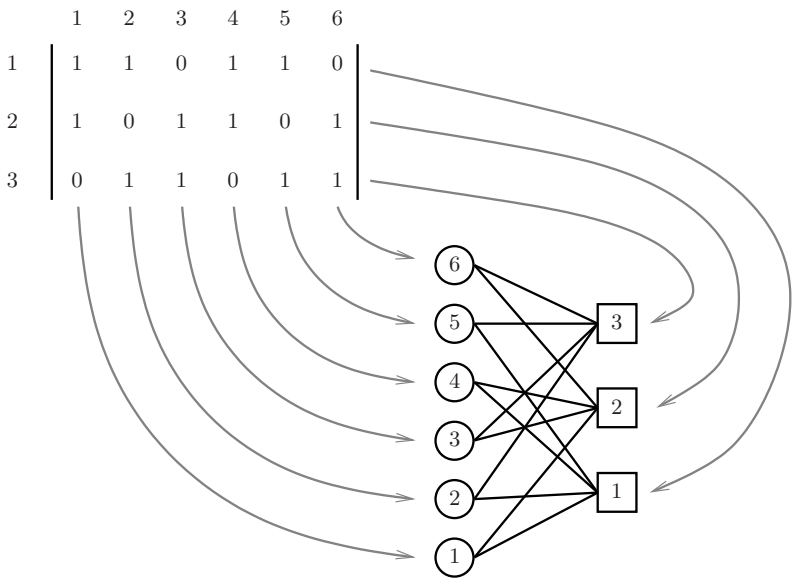

Figure 3.1: Pictorial exemplification of the graph construction for a regular $(2, 4)$ LDPC code.

1. allocate an array of N variable nodes, each one in correspondence to a column of H;

2. allocate an array of M check nodes, each one in correspondence to a row of H;

3. for each nonzero entry of H, connect with a branch the variable and check nodes corresponding to the entry column and row, respectively.

In Figure 3.1, a pictorial exemplification of the construction of the bipartite graph for a very simple LDPC code is shown, characterized by $d_v = 2$ and $d_c = 4$.

In the parity check equation (3.2), each column of the H matrix is multiplied by a corresponding binary symbol in the codeword. Therefore, each variable node is associated with a binary symbol in the codeword. The bit position is represented by the position of the column associated with the variable node. On the other hand, each check node is associated with a parity check equation specified by the row corresponding to that particular check node. A node is said to have "degree i" if i branches depart from it and connect to i different nodes of the other kind.

The above described bipartite graphs for LDPC codes are instances of the so called Tanner graphs for linear block codes. Tanner graphs were introduced by M. Tanner in [31], and may be used to describe the constraints that a codeword must fulfill in order to belong to a particular linear block code. One may observe that, substituting any row in the parity check matrix H with a linear combination of the row itself and any other set of rows does not alter the set of codewords fulfilling the parity check equation (3.2), i.e., the obtained matrix is a parity check matrix for the same code. In other words, every linear block code admits several parity check matrices, or, equivalently, several Tanner graph representations. Nevertheless, in general, the above described linear combination method does not preserve sparseness of the parity check matrix.

It is interesting to note that, although it is possible to construct a parity check matrix for a regular (d_v, d_c) code when d_v is an *even* number, this matrix will not have maximum rank, since there exists at least one linear combination of rows equal to the all-0 vector. In fact, consider the vector $\boldsymbol{r} = (r_1, \ldots, r_N)$ obtained by summing all row vectors in H. The j-th element of $\boldsymbol{r}$ is

$$\begin{aligned} r_j &= \sum_{i=1}^{M} h_{ij} \mod 2 \\ &= d_v \mod 2 \\ &= 0 \end{aligned}$$

where $\{h_{ij}\}$ denote the entries of H. Therefore, it is possible to remove a row from the parity check matrix without modifying the set of codewords and increasing the code rate. As a consequence, there cannot exist regular (d_v, d_c) LDPC codes with even d_v and rate $1 - d_v/d_c$.

3.3 Statistical Description of LDPC Codes

The notation (d_v, d_c) for regular LDPC codes describes a peculiar property, i.e., that the code admits at least a parity check matrix that has exactly d_v ones in each column and d_c ones in each row. There can be more than one actual code that has this property, even if the codeword length N is fixed. Therefore, this notation identifies a class or *ensemble* of codes. Several LDPC codes performance analysis techniques refer to code ensembles, in the sense that the analysis characterizes the expected performance when an actual code is randomly selected within an ensemble.

Although the regularity assumption greatly simplifies performance analysis, it imposes unnecessary constraints on the structure of the parity check

matrix. As a consequence, in [10,13,32] *irregular* LDPC codes have been proposed, which allow for a different number of non-zero elements in each row and column.

The bipartite graph construction does not rely on code regularity and therefore applies to irregular LDPC codes as well. In the irregular case, the number of branches connected to the various nodes may change from node to node, regardless the node kind. In other words, variable (or check) nodes are not constrained to have equal degree.

To give a description of irregular LDPC codes ensembles, in [10, 13, 32], the concept of *degree distribution* is introduced.[1] The degree distributions of an LDPC code is specified by a pair of polynomials

$$(\lambda(x), \rho(x)) \triangleq \left(\sum_{i=1}^{\infty} \lambda_i x^{i-1}, \sum_{j=1}^{\infty} \rho_j x^{j-1} \right)$$

where the coefficient λ_i is the fraction of graph branches connected to degree-i *variable* nodes and ρ_j is the fraction of graph branches connected degree-j *check* nodes [13]. The polynomial $\rho(x)$ is referred to as the *check node degree distribution* and $\lambda(x)$ is referred to as *variable node degree distribution*. The coefficients $\{\rho_j\}$ and $\{\lambda_i\}$ must satisfy the following constraints [13]:

$$\begin{array}{rcll} 0 \leq & \rho_j & \leq & 1 \qquad j \geq 1 \\ 0 \leq & \lambda_i & \leq & 1 \qquad i \geq 1 \\ \sum_{j=1}^{\infty} \rho_j & & = & 1 \\ \sum_{i=1}^{\infty} \lambda_i & & = & 1 \end{array} \tag{3.4}$$

where the third and the fourth relation arise because the "sum of all fractions of edges" must be equal to one.

If a graph has l branches, i.e., the corresponding parity check matrix has l nonzero entries, the number v_i of degree-i variable nodes is

$$v_i = \frac{l\lambda_i}{i}$$

and the number c_j of degree-j check nodes is

$$c_j = \frac{l\rho_j}{j}\,.$$

[1] Note that the concept of degree distribution is borrowed from random graph theory, where it is characterized by a slightly different definition.

Therefore the number N of variable nodes is given by

$$N = \sum_i v_i = l \sum_i \frac{\lambda_i}{i} = l \int_0^1 \lambda(x)\,\mathrm{d}x$$

where the integral notation is sometimes used in the literature for conciseness. The number M of check nodes is given by

$$M = \sum_j c_j = l \sum_j \frac{\rho_j}{j} = l \int_0^1 \rho(x)\,\mathrm{d}x\,.$$

The code rate $R = K/N$, therefore, is as follows:

$$\begin{aligned} R &= \frac{K}{N} = \frac{N-M}{N} = 1 - \frac{M}{N} \\ &= 1 - \frac{l \sum_j \frac{\rho_j}{j}}{l \sum_i \frac{\lambda_i}{i}} = 1 - \frac{\int_0^1 \rho(x)\,\mathrm{d}x}{\int_0^1 \lambda(x)\,\mathrm{d}x} \end{aligned}$$

One can observe that, given the code rate R, the degree distribution coefficients must satisfy the following linear constraint:

$$\sum_{j=1}^{\infty} \frac{\rho_j}{j} = (1-R) \sum_{i=1}^{\infty} \frac{\lambda_i}{i}\,. \tag{3.5}$$

Irregular LDPC codes are among the most powerful binary codes known today. In [13], it is shown how to design the degree distributions of powerful irregular LDPC codes and in [33] it is shown that carefully designed irregular LDPC codes can practically achieve performance as close as 0.0045 dB to the Shannon limit of the additive white Gaussian noise (AWGN) channel. Irregular LDPC codes characterized by performance close to the capacity limit have been obtained for a variety of binary input memoryless channels, among which the binary erasure channel (BEC) and the binary symmetric channel (BSC).

3.4 Decoding Algorithms for LDPC Codes

3.4.1 Sum-Product Algorithm

Maximum a posteriori probability (MAP) decoding, either per-symbol or per-codeword, of a generic linear block code is, in general, a formidable task.

However, in [9] Gallager introduces three suboptimal iterative decoding algorithms for LDPC codes, which exploit the sparseness of the parity check matrix of the code. Two of them have very low complexity and are based on hard decisions and a bit-flipping technique, whereas the third one is a more accurate algorithm which is based on the iterative exchange of real-valued reliabilities of the codeword bits. These algorithms are widely known as Gallager A,B, and C algorithms, after [34].

All these algorithms have the appealing property of being based on the Tanner graph representation of the LDPC code. In particular, they are characterized by the fact that the nodes, both variable and check, act as processors exchanging real-valued messages on the code graph. All the processing is done *locally*, i.e., for each node it is based only on the available messages. The messages represent reliability values for the codeword bits; in particular they represent an estimate of the probability that each particular codeword bit is equal to "1". In [12, 35], the author presents a reinvention of LDPC codes, and the relevant iterative decoding algorithm. He also highlights how the iterative decoding algorithm can be seen as a particular instance of the *belief propagation* (BP) algorithm [36]. In [37], the authors present a general graph-based algorithm, i.e., the *sum-product* (SP) algorithm, which can be useful for computing the marginalization of complex probability density functions. The authors show how BP can be seen as an instance of the SP algorithm and that the SP algorithm achieves optimality if the code graph has no *cycles*. In other words, the Gallager C decoding algorithms computes the exact *a posteriori* probability of each codeword symbol—thus enabling MAP symbol detection—if the code graph has the shape of a tree. This fact had been argued in [9] as well.

The Gallager C algorithm is now introduced with emphasis on its implementation in the logarithmic domain. Given a binary random variable X, taking values in the set $\{0, 1\}$, its likelihood ratio λ_X is defined as

$$\lambda_X = \frac{P\{X = 0\}}{P\{X = 1\}} \tag{3.6}$$

and the corresponding log-likelihood ratio is

$$\Lambda_X = \log \lambda_X \, .$$

At each iteration in the decoding algorithm: (i) first, each variable node computes an output message for each connected edge and, (ii) then, each check node computes an output message for each connected edge. The order of computation of the messages is usually referred to as *schedule*. This is only the

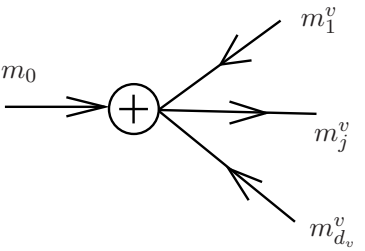

Figure 3.2: Variable node: the quantities involved in the computation of the j-th output message are shown.

most commonly adopted schedule; other schedules exist and can be found in the literature [38–40].

In Figure 3.2, a degree-d_v variable node is shown and the input and output messages are explicitly indicated. One can observe that there is an input, connected to the node and whose value is labeled with m_0, which represents an input reliability value associated with the bit corresponding to the variable node, expressed in the log-likelihood domain. This input reliability value is usually computed on the basis of the observation of the channel.

The decoding algorithm can be derived by reasoning in the probability domain and then by casting the result in the log-likelihood ratio (LLR) domain as follows. The message at the output of a variable node is the probability that the corresponding codeword bit X is equal to "1" given a set of independent observations regarding the bit. Assume to have d_v independent observations $\xi_1, \ldots, \xi_{d_v}$. Let the likelihood ratios of the probability of the observations be

$$\left(\frac{p(\xi_1|X=0)}{p(\xi_1|X=1)}, \ldots, \frac{p(\xi_{d_v}|X=0)}{p(\xi_{d_v}|X=1)} \right) = (\lambda_1, \ldots, \lambda_{d_v}) \, .$$

The probability of X being equal to 0 given the observations is

$$P(X=0|\xi_1, \ldots, \xi_{d_v}) = \frac{p(\xi_1, \ldots, \xi_{d_v}|X=0)P(X=0)}{p(\xi_1, \ldots, \xi_{d_v})}$$

and the corresponding likelihood ratio is

$$\begin{aligned}\lambda &= \frac{P(X=0|\xi_1,\ldots,\xi_{d_v})}{P(X=1|\xi_1,\ldots,\xi_{d_v})} \\ &= \frac{p(\xi_1,\ldots,\xi_{d_v}|X=0)P(X=0)}{p(\xi_1,\ldots,\xi_{d_v}|X=1)P(X=1)} \\ &= \frac{\prod_{i=1}^{d_v} p(\xi_i|X=0)}{\prod_{i=1}^{d_v} p(\xi_i|X=1)} \frac{P(X=0)}{P(X=1)} \\ &= \frac{P(X=0)}{P(X=1)} \prod_{i=1}^{d_v} \lambda_i \end{aligned} \tag{3.7}$$

which, assuming $P(X=0) = P(X=1) = 1/2$, in the LLR domain becomes[2]

$$\Lambda = \sum_{i=1}^{d_v} \Lambda_i \tag{3.8}$$

where $\Lambda_i = \log \lambda_i$.

The variable node decoding algorithm can be formulated in the logarithmic likelihood domain as follows [9]. Each degree-d_v variable node, as shown in Figure 3.2, computes d_v output messages as follows:

$$m_j^v = m_0 + \sum_{\substack{i=1 \\ i\neq j}}^{d_v} m_i^v \tag{3.9}$$

where m_j^v is the j-th output message and m_i^v is the i-th input message coming from a check node. In other words, the variable node treats all the messages at its input as independent observations: $d_v - 1$ are from the check nodes and one, m_0, corresponds to the likelihood ratio associated with an external observation. This external observation can be a sample from the channel, or it might correspond to an *a priori* information on the corresponding bit. The message m_0 enables to account for an a priori information and will be used in Chapter 5 as a feedback input in a more general decoding scheme.

One can observe that the sum explicitly excludes the message coming from the edge whose output message is being computed. This is in agreement with

[2]We remark that the *a priori* probability $P(X=0)$, and in particular its corresponding likelihood ratio $P(X=0)/P(X=1)$ plays the same role in (3.7) as any other likelihood ratio observation. In other words, it could be formally treated as an observation and embedded into the product.

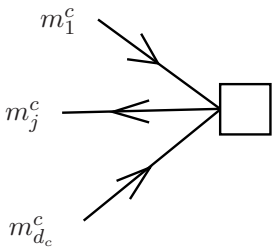

Figure 3.3: Check node: the quantities involved in the computation of the j-th output message are shown.

the use of the so-called *extrinsic information* in iterative detection, and, in particular, in turbo decoding [8].

In Figure 3.3, a generic degree-d_c check node is shown. A check node represents a constraint on the codeword bits associated to the variable nodes connected to it. This constraint is expressed by the corresponding parity check matrix row: *the modulo-2 sum of the codeword bits connected to the check node is equal to 0.* In this case, the associated problem is the following. Given probabilities

$$(P(X_1 = 1), \ldots, P(X_{d_c-1} = 1)) = (p_1, \ldots, p_{d_c-1})$$

associated with $d_c - 1$ bits of the d_c bits connected to the parity check node, compute the probability P_0 that their sum modulo-2 is equal to 0 (we know that the sum of *all* d_c bits modulo-2 is equal to 0). Assuming that all the observations leading to the computation of $P(X_1 = 1), \ldots, P(X_{d_c-1} = 1)$ are independent, then

$$\begin{aligned} P_0 &= \sum_{\boldsymbol{x}:\text{sum is even}} P(\boldsymbol{x}) \\ &= \sum_{\boldsymbol{x}:\text{sum is even}} \prod_{j=1}^{d_c-1} P(x_j) \end{aligned}$$

where "$\boldsymbol{x}$: sum is even" denotes all bit vectors $\boldsymbol{x}$ of length $d_c - 1$ whose sum is an even number, i.e., $\boldsymbol{x}$ contains an even number of 1's. The computation of this quantity can be performed following the guidelines in [9]. First, consider the following polynomial in t

$$q(t) = \alpha_0 + \alpha_1 t + \ldots + \alpha_{d_c-1} t^{d_c-1} = \prod_{j=1}^{d_c-1} (1 - p_j + p_j t)\,.$$

Observe that the coefficient α_i is given by

$$\alpha_i = \sum_{k_1<k_2<\ldots<k_i} p_{k_1} \ldots p_{k_i} \prod_{j\neq k_1,\ldots,k_i} (1-p_j)\,.$$

In other words, α_i is the probability of having i ones and $d_c - 1 - i$ zeros among the $d_c - 1$ bits. Observe now that $q(1)$ is the sum of all coefficients and $q(-1)$ is the sum of all coefficients where all odd coefficients have changed sign. Therefore

$$q(1) + q(-1) = \sum_{j=0}^{d_c-1} (1 + (-1)^j)\alpha_j = \sum_{k=0}^{\lfloor (d_c-1)/2 \rfloor} 2\alpha_{2k}$$

is equal to two times the sum of all even coefficients, i.e., twice the probability of having an even number of 1's. As a consequence,

$$P_0 = \frac{q(1)+q(-1)}{2} = \frac{1+\prod_{j=1}^{d_c-1} 1-2p_j}{2}.$$

The corresponding LLR is

$$\begin{aligned}
\Lambda &= \log \frac{P_0}{1-P_0} \\
&= \log \frac{1+\prod_{j=1}^{d_c-1}(1-2p_j)}{1-\prod_{j=1}^{d_c-1} 1-2p_j} \\
&= \log \frac{1+\prod_{j=1}^{d_c-1}(1-\frac{2}{e^{\Lambda_j}+1})}{1-\prod_{j=1}^{d_c-1} 1-\frac{2}{e^{\Lambda_j}+1}} \\
&= \log \frac{1+\prod_{j=1}^{d_c-1}\tanh(\Lambda_j/2)}{1-\prod_{j=1}^{d_c-1}\tanh(\Lambda_j/2)} \\
&= 2\text{atanh} \prod_{j=1}^{d_c-1} \tanh(\Lambda_j/2)
\end{aligned}$$

where

$$\Lambda_j = \log \frac{1-p_j}{p_j}\,.$$

Recalling the check node in Figure 3.3, in order to compute the generic output message, each check node performs the following computation:

$$m_j^c = 2\ \text{atanh} \prod_{\substack{i=1 \\ i\neq j}}^{d_c} \tanh \frac{m_i^c}{2} \tag{3.10}$$

where m_j^c is the j-th output message and m_i^c is the i-th input message coming from the variable nodes. As in the variable node case, the message coming from the j-th edge is not used for the computation of the outgoing message in the j-th edge. The messages can be interpreted as LLRs of the bits associated with the variable nodes towards/from which the message is directed.

At the end of the decoding process, each variable node computes an output reliability value as follows:

$$m^v = m_0 + \sum_{i=1}^{d_v} m_i^v \tag{3.11}$$

where d_v is the degree of the node. In other words, the output reliability value of a codeword bit is the sum of all messages directed towards the corresponding variable node. As for the forward-backward (FB) algorithm, this step may be referred to as *completion*—see Chapter 2 for more details on the FB algorithm. From (3.6), an LLR referring to a binary random variable can be straightforwardly used to compute a MAP estimate of the random variable. In fact, if the sign of the LLRs is positive the probability of the random variable being equal to 0 is larger than the probability of the random variable being equal to 1. Vice versa, if the sign of the LLRs is negative the probability of the random variable being equal to 0 is smaller than the probability of the random variable being equal to 1. Thus, the signs of the LLRs in (3.11) are all is needed to obtain decisions on the codeword bits.

Summarizing, the Gallager C decoding algorithm comprises the following steps.

1. Compute all the reliability values for the symbols in the codeword. These values correspond, for each variable node, to the m_0 value in Figure 3.2.
2. Initialize to 0 all the messages coming from the check nodes.
3. Compute the variable nodes' output messages using (3.9).
4. Transfer the messages to the check nodes.
5. Compute the check nodes' output messages using (3.10).
6. Transfer the messages to the variable nodes.
7. Verify that a *stopping criterion*—described in the following paragraph—is met. If not, go to step 2.
8. Compute the final reliability values using (3.11).

A stopping criterion may be based on several possible events. The two most common are (i) the codeword bits that would be obtained after the completion step form a valid codeword, and (ii) a given maximum number of iterations is reached. It has been observed [41] that the Gallager C algorithm for LDPC codes has the interesting property of exhibiting particularly low probability of not detecting an erroneous decoding result, i.e., when the decoding process fails, the decoder is mostly aware of the failure. In fact, unlike turbo codes, the BP algorithm for LDPC codes operates on codeword bits rather than information bits. This is easily recognized by observing that the information bits (i.e., the information payload in the codeword) and the parity bits are dealt with in the same way. Neither convergence to the optimum codeword nor convergence to a codeword at all is guaranteed. It seems apparent that, whenever a decoding error occurs, the resulting decided bit sequence is not a codeword, in the sense that usually it does not satisfy (3.2).

In the above described algorithm, the structure of message passing is rigid: first, every variable node computes its messages; then, *all* the messages are passed to the check nodes, which in turn compute all the messages at their outputs. All the obtained messages are then sent back to the variable nodes. This scheduling is optimum when applied to a tree-shaped bipartite graph, i.e., a graph without cycles. If the graph has cycles, the algorithm becomes suboptimal and there may be some benefit in adopting other scheduling schemes. Another motivation for using other scheduling patterns, rather than the standard Gallager C algorithm, is to improve the computational and implementation efficiencies of circuits devoted to LDPC decoding.

In the following, we will refer to the set of variable node processors as *variable node detector* (VND) and to the set of check node processors as *check node detector* (CND). The LDPC decoding process can be seen as an iterative exchange of vector messages, referred to as *message sets*, between VND and CND.

3.4.2 Min-Sum Algorithm

The check node operation (3.10) in the log-domain relies on the computation of complex nonlinear functions, i.e., atanh$(\cdot)$ and tanh$(\cdot)$. The computation of (3.10) may be formulated in a recursive form as described in the following. Let $(m_1, \ldots, m_{d_c-1})$ denote the messages in the product of (3.10), where the superscript c has been omitted and consecutive indices have been adopted to

simplify the notation. If we define the following recursion:

$$\begin{aligned} m_1^* &= m_1 \\ m_{i+1}^* &= 2\,\mathrm{atanh}\tanh\frac{m_i^*}{2}\tanh\frac{m_{i+1}}{2} \end{aligned} \tag{3.12}$$

then

$$m_j^c = m_{d_c-1}^* .$$

Note that, for any $x, y \in \mathbb{R}$

$$\begin{aligned} &2\mathrm{atanh}\left(\tanh\frac{x}{2}\tanh\frac{y}{2}\right) \\ &= \mathrm{sgn}(x)\mathrm{sgn}(y)\left(\min\{|x|,|y|\} + \log\frac{1+e^{-|x|-|y|}}{1+e^{-||x|-|y||}}\right) \end{aligned}$$

where the term

$$\log\frac{1+e^{-|x|-|y|}}{1+e^{-||x|-|y||}} \tag{3.13}$$

becomes small whenever $||x| - |y||$ is large. This result can be obtained by observing that, first, both $\tanh(\cdot)$ and $\mathrm{atanh}(\cdot)$ have odd symmetry and, therefore,

$$2\mathrm{atanh}\left(\tanh\frac{x}{2}\tanh\frac{y}{2}\right) = 2\mathrm{sgn}(x)\mathrm{sgn}(y)\mathrm{atanh}\left(\tanh\frac{|x|}{2}\tanh\frac{|y|}{2}\right) .$$

Expanding the right-hand side, one obtains:

$$\begin{aligned} 2\mathrm{atanh}\left(\tanh\tfrac{|x|}{2}\tanh\tfrac{|y|}{2}\right) &= \log\frac{1+\tanh(|x|/2)\tanh(|y|/2)}{1-\tanh(|x|/2)\tanh(|y|/2)} \\ &= \log\frac{1+e^{-|x|-|y|}}{e^{-|x|}+e^{-|y|}} \\ &= -\log\left(e^{-|x|}+e^{-|y|}\right) + \log\left(1+e^{-|x|-|y|}\right) \\ &= \min\{|x|,|y|\} - \log\left(1+e^{-||x|-|y||}\right) \\ &\qquad + \log\left(1+e^{-|x|-|y|}\right) \\ &= \min\{|x|,|y|\} - \log\frac{1+e^{-|x|-|y|}}{1+e^{-||x|-|y||}} \end{aligned}$$

where the well known identity

$$\log\left(e^a+e^b\right) = \max\{a,b\} + \log\left(1+e^{-|a-b|}\right)$$

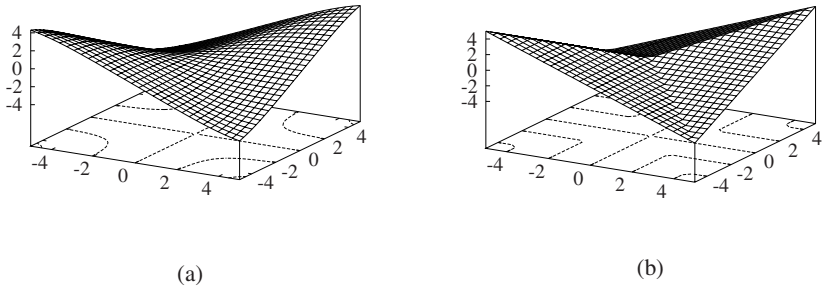

Figure 3.4: Comparison of the check node functions: (a) BP algorithm and (b) Min-Sum algorithm.

has been used. By neglecting the term (3.13) in the computation of the recursion (3.12), one obtains the following approximate check node operation:

$$m_j^c = \prod_{\substack{i=1 \\ i \neq j}}^{d_c} \operatorname{sgn}(m_i^c) \min_{i \neq j} \{m_i^c\}$$

which yields the so-called Min-Sum approximation of the BP algorithm.

In Figure 3.4.2, the check node operation in the LLR domain for a degree-3 check node is shown considering (a) the BP algorithm and (b) the Min-Sum algorithm. The x and y axes represent the two input LLR values and the z axis represents the output message in the LLR domain. One can observe the visual similarity of the two functions. In Figure 3.5, the difference between the exact function and its Min-Sum approximation is shown. One can observe that the highest difference is concentrated in the region where the two message values are close. If the input reliabilities to the variable nodes are characterized by high LLR values, as in the case of a good channel such as an AWGN channel with high SNR, the average difference between the Min-Sum approximation and the BP is expected to be small.

Min-Sum decoding has an interesting property that we will briefly discuss. Observe that, in the case of binary-input AWGN (BIAWGN) channel described in Example 1.1, the input LLR reliability value to the k-th variable node can

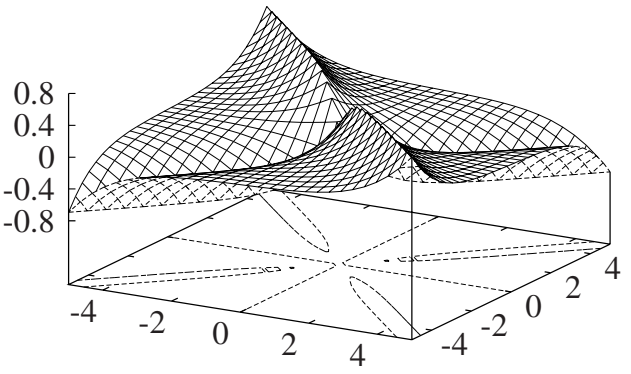

Figure 3.5: Difference between the BP and Min-Sum check node functions.

be computed as follows:

$$m_0 = \log \frac{\frac{1}{\sqrt{2\pi\sigma_w^2}} e^{-\frac{(y_k-1)^2}{2\sigma_w^2}}}{\frac{1}{\sqrt{2\pi\sigma_w^2}} e^{-\frac{(y_k+1)^2}{2\sigma_w^2}}} = \log e^{\frac{4y_k}{2\sigma_w^2}} = \frac{2y_k}{\sigma_w^2}$$

where y_k is the k-th received sample, i.e., signal plus noise, and σ_w^2 is the additive noise variance. In order to perform LDPC decoding with the above described BP algorithm, the only needed channel information consists of the knowledge of σ_w^2, which is used to compute the input LLRs. If in a variable node or check node Min-Sum operation all the messages are multiplied by a factor $\alpha > 0$, the resulting output LLR is multiplied by the same factor α. Therefore, if in the Min-Sum algorithm every input LLR is multiplied by α, the obtained messages, at every iteration and in every graph edge, change only by a constant factor α. In particular, the message sign does not change and, as a consequence, the bit decisions do not change.

By choosing $\alpha = \sigma_w^2/2$, one obtains a decoding algorithm that does not rely on the knowledge of the channel statistics, i.e., knowledge of σ_w^2 is unneeded and the input messages can be computed simply as

$$m_0 = y_k \,.$$

For this reason, the Min-Sum decoder is also known as *universal decoder* [42]. The performance of the Min-Sum decoder can be improved in a number of ways. In particular, the application of a correction factor and an offset in the check node output message may have beneficial effect on the decoder performance [43].

3.4.3 Alternative Decoding Algorithms

Several techniques have been proposed either to achieve better performance than that of BP decoding or to obtain low-complexity decoding. Linear programming (LP) decoding of LDPC codes has been proposed in [44, 45], where it is shown how to formulate the maximum likelihood (ML) decoding of LDPC codes as a LP problem. To make the use of an efficient LP algorithm feasible, the constraints on the solution must be (approximately) relaxed. Several improvements have been proposed to tighten the distance between approximate LP decoding and ML decoding. LP decoders have, in general, better performance than BP decoders. Nevertheless, their use as component blocks of the complex iterative receivers that will be investigated in the next chapters is difficult since they are conceived to perform *per-codeword* ML detection as opposed to *per-bit* MAP detection, which is better suited for iterative detection [46].

Other reduced complexity decoding techniques comprise bit-flipping techniques, which refer to graph-based algorithms with binary valued messages (see, e.g., [9] and [47] and references therein). Message quantization has been also investigated. The basis for performance analysis using quantized message passing algorithm for LDPC decoding may be found in [34].

3.5 Practical LDPC Code Design from Statistical Description

As usual, design techniques rely on performance evaluation techniques. The most effective LDPC code performance evaluation techniques will be discussed in Chapter 4 and are based on asymptotic analysis of LDPC code ensembles defined by their degree distributions. The use of these techniques in code design allows to optimize the degree distributions of the code, i.e., its statistical description. In order to use the code, it is necessary to construct a parity check matrix that satisfies, in addition to the obtained degree distributions, all the desired constraint, such as, for example, the codeword length.

A widely known LDPC construction algorithm is the progressive edge

growth (PEG) algorithm [48, 49], which enables to design good LDPC codes characterized by high minimum cycle length, or *girth*, and based on a given degree distribution.

Most of the results obtained in this book will be based on LDPC codes constructed using the simple random graph construction algorithm described in the following.

The key goal is to obtain a code with, in order of priority:

1. a given codeword length N;
2. a girth larger than or equal to a given value γ;
3. a given rate R;
4. given degree distributions.

In general, it is not possible to satisfy the last two constraints. In fact, since R is generally a real number and the degree distributions have real coefficients, there may be cases where it is not possible to build a finite length code characterized by given code rate R and degree distributions. These two design constraints may need to be somehow relaxed, allowing for small roundings on the degree distributions coefficients.

In order to build a code we first compute, based on N and the variable node degree distribution, the number ℓ of edges in the graph

$$\ell = \frac{N}{\sum_i \frac{\lambda_i}{i}}$$

and the number v_i and c_i of degree-i variable and check nodes, respectively, for each i:

$$\begin{aligned} v_i &= \ell \frac{\lambda_i}{i} \\ c_i &= \ell \frac{\rho_i}{i}. \end{aligned}$$

The obtained values must be properly rounded in order to obtain an integer number of nodes for each degree. In addition, the rounding strategy for v_i must be chosen in order to obtain a total number of variable nodes equal to N, and the rounding of c_i must preserve the number of actual edges ℓ in the variable nodes.

To build the LDPC code graph, first the available variable and check nodes must be arranged in two separate groups. Each degree-i node of a specific kind,

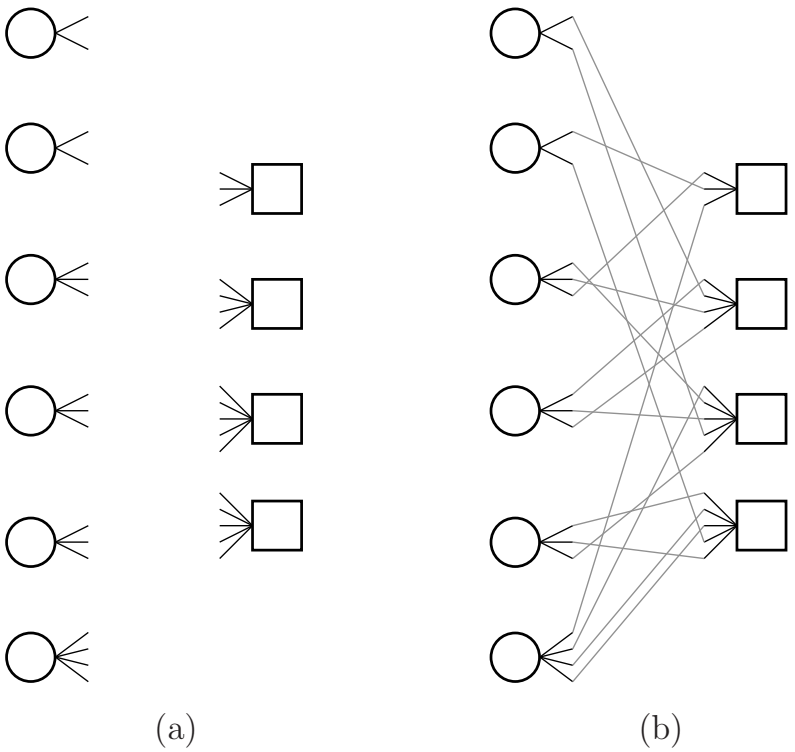

Figure 3.6: Simple code construction: (a) arranged nodes with empty sockets and (b) graph after all empty sockets have been connected.

either check or variable, has i "empty sockets," each ready to be connected to a socket of a node of the other kind. Figure 3.6 (a) shows the arranged nodes ready to be connected. After having prepared the nodes to be connected, the construction algorithm is as follows:

1. start from the first socket in the first variable node;
2. connect the current socket with a random socket in the check nodes;
3. verify that the new connection does not generate a cycle up to a given desired girth γ (by exploring the graph starting from the new edge up to a depth $\gamma - 1$);
4. if no new short cycle is generated pass to the next empty socket in the variable nodes and repeat starting from 2; else, delete the connection

and repeat from 2.

If the algorithm reaches a dead end, i.e., it is not possible to find an empty check node socket that does not generate a cycle of length longer than γ, all the connections are erased and the algorithm restarts from the beginning. If after a given number of trials the algorithm does not provide a complete graph, then the algorithm fails and returns no code at all. In this case, it may be useful to reduce the required girth or to increase the codeword length. Eventually, all empty sockets will be connected, as shown in Figure 3.6 (b).

3.6 Encoding Techniques for LDPC Codes

On the basis of the vector of information bits to be transmitted, encoding is the operation that selects, from the set of all codewords, i.e., the codebook, one corresponding codeword. In general, LDPC code encoding can entail significant complexity due to the random code structure. In the following subsections, some encoding techniques are described that may be applied to any generic LDPC code.

3.6.1 Encoding by Matrix Multiplication

In several contexts involving binary coding, it is useful that the information bits explicitly appear in the associated codeword. This allows, for example, to avoid costly decoding when the communication system is operating on a particularly favorable channel, introducing negligible uncertainty on the received data sequence. Codes satisfying this condition are known as *systematic*. The part of codeword replicating the data bits is known as the *systematic portion* of the codeword. In convolutional encoding, the systematic portion is often interleaved with the non-systematic part (also known as *parity bits*). In this book, we will address systematic LDPC codes whose systematic part consists of the first bits of the codeword.

Since LDPC codes are linear block codes, they can be encoded using the corresponding $N \times K$ generation matrix G. In particular, the codeword $\boldsymbol{x}$ can be obtained as follows:

$$\boldsymbol{x} = G\boldsymbol{a}$$

where $\boldsymbol{a} = (a_1, \ldots, a_K)^T$ is the vector of K information bits to be encoded. Since the code is systematic, the generation matrix must be of the form

$$G = \begin{pmatrix} I_K \\ \cdots \\ P \end{pmatrix}.$$

where P is a $(N-K)\times K$ matrix that generates the parity bits of the codeword. The encoded vector $\boldsymbol{x} = G\boldsymbol{a}$ is in the form

$$(a_1, \ldots, a_K, p_1, \ldots, p_{N-K})^T$$

where the vector of parity bits is

$$\boldsymbol{p} = (p_1, \ldots, p_{N-K})^T = P\boldsymbol{a}\,.$$

The matrix $\tilde{H}$ defined as

$$\tilde{H} = \left(P \vdots I_{N-K}\right)$$

is a parity check matrix for the systematic code generated by G, meaning that a vector $\boldsymbol{x}$ is a codeword if and only if

$$\tilde{H}\boldsymbol{x} = 0\,.$$

This fact can be easily recognized since any bit vector $\boldsymbol{x} = (x_1, \ldots, x_N)$ is a codeword only if the last $N - K$ bits $\hat{\boldsymbol{p}} = (x_{K+1}, \ldots, x_N)$ are equal to the parity bit vector $\boldsymbol{p}$ generated by the first K codeword bits as follows:

$$\boldsymbol{p} = P(x_1, \ldots, x_K)^T\,.$$

It is easily recognized that

$$\tilde{H}\boldsymbol{x} = \left(P \vdots I_{N-K}\right)(x_1, \ldots, x_K, x_{K+1}, \ldots, x_N) = \boldsymbol{p} + \hat{\boldsymbol{p}}$$

which is equal to 0, using boolean algebra, if and only if

$$\boldsymbol{p} = \hat{\boldsymbol{p}}$$

i.e., if and only if $\boldsymbol{x}$ is a codeword.

The generation matrix G is completely defined by its sub-matrix P, which, in turn, can be computed by transforming a given parity check matrix H into its $\tilde{H}$ form, by means of simple row operations (i.e., substitution of a row with the linear combination of itself and other rows).

Given a parity check matrix H, in order to guarantee the existence of a corresponding systematic code, the rightmost $(N - K) \times (N - K)$ sub-matrix of H must be non singular. If this condition does not hold, however, observe that, if H has maximum rank, there exists a subset of $N - K$ columns of H that forms a non-singular matrix. Therefore, it is possible to obtain a parity

check matrix corresponding to a systematic code by applying a permutation of the columns of H that places those columns in the rightmost positions.

Although H is sparse by definition, the sub-matrix P of G is, in general, dense. This means that, discarding the identity matrix part, in order to store G, at least

$$K(N-K) = K^2\left(\frac{1}{R}-1\right)$$

bits of memory are needed. Since a typical LDPC codeword size ranges from 10^3 to 10^4 it means that the memory for storing G for, e.g., a rate 1/2 code (i.e., $N-K=K$), is at least on the order of 10^6 bits.

The matrix multiplication operation, accounting only for the dense part of G, requires $K(N-K) = K^2(1/R-1)$ multiplications (boolean AND operations) and $(K-1)(N-K) = (K^2-K)(1/R-1)$ additions (boolean EXOR operations). The encoding complexity is therefore *quadratic* in the number of information bits per codeword or, given the code rate, in the length of the codeword. It is also an increasing function of the code rate. The $P\boldsymbol{a}$ matrix multiplication can be intrinsically done in parallel, by performing several scalar products. This may be useful if speed or latency are critical issues, although it requires a specific implementation.

3.6.2 Recursive Encoding and Structured Codes

Since BP decoding has linear computational cost in the block length, for long enough codes, encoding may become more computationally expensive than decoding itself. To overcome this problem, several solutions have been proposed. A possible technique is to design highly structured codes. This approach generated a thriving field of research that lead to the invention of several coding techniques. A particularly interesting possibility is to build LDPC codes that are quasi-cyclic, thus enabling low-complexity encoding [50–52]. Another example is given by [53], where turbo-Gallager codes are introduced. They consist of a class of turbo codes which can be effectively decoded by means of a standard LDPC code decoder and, simultaneously, are characterized by the linear complexity encoding property of turbo codes. A family of codes which enable efficient recursive encoding is proposed in [54], with particular emphasis on digital subscriber line applications. Due to their highly structured parity check matrix, the proposed codes enable the use of a very simple circuit for encoding purposes. Besides possible simplification of the encoding procedure, structured codes might have other interesting benefits including

- the possibility of an algorithmic description of the parity check matrix,

thus avoiding the storage of the whole matrix;

- a more efficient interconnection of the processing blocks involved in the LDPC code decoding: an unstructured LDPC code requires the capability of a generic interconnection structure which might lead to unsolvable difficulties in the design of the system hardware;

Another possible technique to obtain efficient encoding is proposed in [55], where it is shown how to exploit the sparseness of the parity check matrix to obtain quasi-linear encoding complexity. The proposed technique exhibits a one-to-one correspondence with the decoding process for an erasure channel using the same code. The procedure is recursive, i.e., all the parity bits but a small fraction are obtained using a recursion, whereas the remaining bits are obtained by means of a matrix multiplication.

It is interesting to note that, if the parity check matrix can be put in lower triangular form in its rightmost part by means of a row permutation, then a BP decoding process, initialized with zero LLRs for the parity bits part and the value $1 - 2a_k$ in the k-th systematic bit, completely recovers all the parity bits after a sufficiently large number of iteration (at most $N-K$). This means that, if the parity check matrix fulfills the previously given condition, the decoder can be used for encoding as well. If the parity check matrix does not fulfill the requirements for exact *encoding by decoding*, then at the end of the encoding process there will be a fraction of parity bits that remain uncertain. In principle, it could be possible to use the partial encoding result to compute a signal to be transmitted. Depending on the strategy used this would result in a performance loss, due to a *suboptimal encoding*.

3.7 Concluding Remarks

In this chapter, an overview of LDPC codes has been given. We remark that, on the basis of the structure and properties of LDPC codes, a huge number of new code families have been proposed in several works. In this book, we focus on generic irregular LDPC codes since we will use an LDPC code as a component block of a more complex system tailored for transmission on specific channels. In the next chapter, an overview of the most important performance analysis techniques for iterative detection schemes will be given, with particular emphasis on LDPC coded schemes.