Chapter 12
Low-Density Parity-Check (LDPC) Codes

E. Paolini

Abstract In this chapter, low-density parity-check (LDPC) codes, a class of powerful iteratively decodable error correcting codes, are introduced. The chapter first reviews some basic concepts and results in information theory such as Shannon's channel capacity and channel coding theorem. It then overviews the flash memory channel model. Next, it addresses binary LDPC codes describing both their structure and efficient implementation, and their belief propagation and reduced-complexity decoding algorithms. Non-binary LDPC codes and their belief propagation decoding algorithm are also addressed. Finally simulation results are provided.

12.1 Shannon Limit

12.1.1 Entropy and Mutual Information

Let X be a discrete random variable taking its values in a set $\mathcal{X}$, according to some probability mass function (pmf) $p(x) = \Pr\{X = x\}$. The entropy of X is defined as

$$H(X) = -\sum_{x} p(x) \log_2 p(x).$$

Intuitively, the entropy $H(X)$ may be thought as the uncertainty associated with the random variable. For example, a deterministic variable is characterized by a zero entropy while, for a given positive integer M, the random variable with the largest entropy among all discrete random variables whose support set $\mathcal{X}$ has cardinality M is the uniform one, i.e., $p(x) = 1/M$ for all $x \in \mathcal{X}$. In this latter case we obtain $H(X) = \log_2 M$.

E. Paolini (✉)
DEI, University of Bologna, Bologna, Italy
e-mail: e.paolini@unibo.it

R. Micheloni et al. (eds.), *Inside Solid State Drives (SSDs)*,
Springer Series in Advanced Microelectronics 37,
https://doi.org/10.1007/978-981-13-0599-3_12

Consider now a second discrete random variable $Y \in \mathcal{Y}$ characterized by a pmf $p(y)$. Let $p(y|x) = \Pr\{Y = y | X = x\}$ be the pmf of Y conditioned to the event $\{X = x\}$. The entropy of Y given the event $\{X = x\}$ is defined as

$$H(Y|X=x) = -\sum_y p(y|x) \log_2 p(y|x).$$

Next, the conditional entropy $H(Y|X)$ is defined as

$$\begin{aligned} H(Y|X) &= \sum_x p(x) H(Y|X=x) \\ &= -\sum_x \sum_y p(y|x) p(x) \log_2 p(y|x). \end{aligned}$$

Finally, the mutual information $I(X;Y)$ between X and Y is defined as

$$I(X;Y) = \sum_x \sum_y p(y|x) p(x) \log_2 \frac{p(y|x)p(x)}{p(x)p(y)}. \tag{12.1}$$

It can be shown that $I(X;Y) = H(Y) - H(Y|X) = H(X) - H(X|Y)$. As such, $I(X;Y)$ intuitively represents the reduction of uncertainty about X due to the fact that we can observe Y (equivalently, reduction of uncertainty about Y due to the fact that we can observe X). The mutual information is well-defined also for continuous random variables. In this case, $p(x)$, $p(y)$, and $p(y|x)$ are probability density functions (pdfs), and we have

$$I(X;Y) = \int p(y|x) p(x) \log_2 \frac{p(y|x)p(x)}{p(x)p(y)} dxdy. \tag{12.2}$$

Moreover, if X is a discrete random variable and Y is a continuous one, $I(X;Y)$ is defined as

$$I(X;Y) = \sum_x p(x) \int p(y|x) \log_2 \frac{p(y|x)p(x)}{p(x)p(y)} dy. \tag{12.3}$$

12.1.2 System Model and Channel Capacity

The fundamental limit of point-to-point digital communication over a noisy channel was established in 1948 by C. Shannon, who showed that a vanishing error probability can be attained at a finite information rate, provided this rate is smaller than the *capacity* of the noisy channel.

With reference to Fig. 12.1, a source S of information generates messages that must be delivered to a destination D through a noisy channel. The generic message,

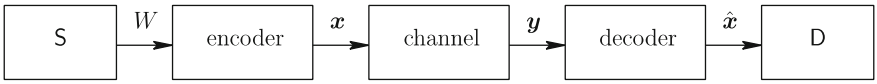

Fig. 12.1 Communication model

denoted by W, is drawn from a set of M possible messages $\{1, 2, \ldots, M\}$, where all messages are a priori equally likely. Prior to transmission over the channel, the message W is encoded through a *channel encoder*, that maps deterministically (and univocally) each message onto a *codeword* $x = [x_0, x_1, \ldots, x_{n-1}]$, i.e., an n-tuple of symbols belonging to some alphabet $\mathcal{X}$. The ratio

$$R = \frac{\log_2 M}{n}$$

is the code rate of the channel code and the code is named an $(n, 2^{nR})$ code. All n codeword symbols are then transmitted sequentially over the channel, resulting in a sequence $y = [y_0, y_1, \ldots, y_{n-1}]$ whose symbols belong to an alphabet $\mathcal{Y}$. A decoding algorithm is then performed by a *channel decoder* to decide which codeword, out of the set of M candidate codewords, had been transmitted over the channel, given the noisy observation y. The codeword $\hat{x}$ returned by the decoder is converted back to the corresponding message $\hat{W}$ that is finally delivered to the destination. As error occurs whenever $W \neq \hat{W}$, i.e., a wrong message is delivered.

A probability of error can be defined for each of the M transmitted messages as follows. The probability of error associated with the j-th message, $j \in \{1, 2, \ldots, M\}$, is denoted by $P_{e,j}$ and is defined as

$$P_{e,j} = \Pr\{\hat{W} \neq W | W = j\}.$$

Furthermore, the maximum probability of error is defined as

$$P_{e,max} = \max_{j \in \{1, 2, \ldots, M\}} P_{e,j} \tag{12.4}$$

and the average probability of error as

$$P_e = \frac{1}{M} \sum_{j=1}^{M} P_{e,j}. \tag{12.5}$$

The channel code along with its decoding algorithm shall be designed in order to make the maximum probability of error over the given channel as small as possible.

Assume that both the input alphabet $\mathcal{X}$ and the output alphabet $\mathcal{Y}$ are discrete. Let $X \in \mathcal{X}$ and $Y \in \mathcal{Y}$ be two discrete random variables, representing the input to the channel and the corresponding output. Moreover, assume that the channel is fully defined by the transition probabilities $p(y|x) = \Pr\{Y = y | X = x\}$. In this case,

the channel is called a discrete memory-less channel (DMC). The capacity of a DMC is defined as

$$C = \max_{p(x)} I(X;Y) \tag{12.6}$$

i.e., as the maximum amount of uncertainty we can remove from the input symbol (which cannot be observed directly) by observing the output symbol, where the maximum is taken over all possible pmfs for the input symbol. The capacity is an intrinsic parameter of the channel, only depending on the cardinalities of $\mathcal{X}$ and $\mathcal{Y}$ and on the transition probabilities $p(y|x)$. It is expressed in terms of information bits (or Shannon) per channel use.

Example 12.1 The DMC depicted in Fig. 12.2 is characterized by $\mathcal{X} = \mathcal{Y} = \{+1, -1\}$ and by $\Pr\{Y = +1|X = +1\} = \Pr\{Y = -1|X = -1\} = 1 - p$, $\Pr\{Y = +1|X = -1\} = \Pr\{Y = -1|X = +1\} = p$. This channel is known as binary symmetric channel (BSC), and p is called the error (or crossover) probability. Every binary symbol input to the channel is received in error with probability p and is correctly received with probability $1 - p$. The capacity of the BSC is achieved for $\Pr\{X = +1\} = \Pr\{X = -1\} = 1/2$ and is given by[1]

$$C = 1 - [-p \log_2 p - (1-p) \log_2 (1-p)]. \tag{12.7}$$

As we shall see later, the BSC is a possible channel model for SLC Flash memories. Assuming $p \leq 1/2$, its capacity is maximum for $p = 0$, where we have $C = 1$ (every binary symbol outcoming from the channel is reliable) and is minimum for $p = 1/2$, where we have $C = 0$ (no uncertainty is removed from X by observing Y).

The concept of capacity, so far introduced for a DMC, can be extended to time-discrete memory-less channels whose input symbol is either a discrete or a continuous random variable and whose output symbol is a continuous one. The capacity is still defined by (12.6), where the mutual information is now given by (12.2) if X is continuous, and by (12.3) if X is discrete. As opposed to the DMC case, however, additional constraints to the optimization problem may be introduced (for example, an upper bound on the average transmitted power). The reason is that the solution to the unconstrained optimization problem may correspond to an input variable X for which the channel is essentially noiseless.

Additive noise channels represent an important class of such channels. Here, the output symbol is obtained as $Y = X + Z$, where Z is a continuous random variable, namely, an additive noise. If Z is independent of X and is normally distributed with zero mean and variance σ^2,

[1]The capacity of the BSC only depends on the crossover probability and not on the values assumed by X and Y.

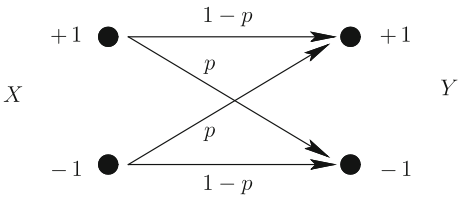

Fig. 12.2 Binary symmetric channel (BSC) model

$$p(z) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{z^2}{2\sigma^2}},$$

then the corresponding channel is called an additive Gaussian channel.

Example 12.2 Consider the additive Gaussian channel depicted in Fig. 12.3, and assume that $\boldsymbol{X}$ is a Bernoulli (i.e., discrete with a binary alphabet) random variable. Without any further constraint, it is possible to achieve the capacity $C = 1$ (corresponding to a noiseless channel) *regardless* of σ^2 by letting $X \in \{-A, +A\}$, where $A > 0$ is a real, choosing $\Pr\{X = -A\} = \Pr\{X = +A\} = 1/2$, and letting $A \to \infty$. On the other hand, if the maximization problem is constrained to $(1/n)\sum_{i=0}^{n-1} x_i^2 \le E_s$ for any transmitted codeword, then the maximum is attained for $X \in \left\{-\sqrt{E_s}, +\sqrt{E_s}\right\}$ and $\Pr\left\{X = -\sqrt{E_s}\right\} = \Pr\left\{X = +\sqrt{E_s}\right\} = 1/2$. In this case (12.3) yields

$$C = -\int p(y) \log_2\left(p(y)\sqrt{2\pi e\sigma^2}\right) dy,$$

where

$$p(y) = \frac{1}{\sqrt{8\pi\sigma^2}} \left(e^{-\frac{\left(y-\sqrt{E_s}\right)^2}{2\sigma^2}} + e^{-\frac{\left(y+\sqrt{E_s}\right)^2}{2\sigma^2}} \right).$$

and where the capacity, that does not admit a closed-form expression, must be computed via numerical integration. This channel model is known as the binary-input additive white Gaussian noise (Bi-AWGN) channel. It is possible to show that its capacity is a function of parameter E_s/N_0, where $N_0 = 2\sigma^2$. In general, the larger E_s/N_0 the higher C. Moreover, $C \to 1$ as $E_s/N_0 \to \infty$.

Example 12.3 Consider a channel $X \to Y' \to Y$ composed of the cascade of a Bi-AWGN channel and a one-bit quantizer, returning $Y = +1$ if $Y' > 0$ and $Y = -1$ otherwise (if $Y' = 0$, $+1$ or -1 is returned with equal probability). It is readily shown that this channel is equivalent to a BSC whose crossover probability p is

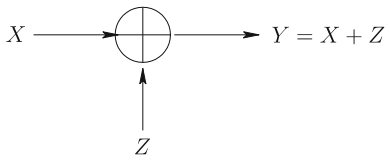

Fig. 12.3 Binary-input additive white Gaussian noise channel model

$$p = \frac{1}{2}\mathrm{erfc}\left(\sqrt{\frac{E_s}{N_0}}\right) \tag{12.8}$$

where

$$\mathrm{erfc}(x) = \frac{2}{\sqrt{\pi}}\int_x^{\infty} e^{-\theta^2} d\theta.$$

Again, the capacity is a monotonically increasing function of parameter E_s/N_0, and again $C \to 1$ as $E_s/N_0 \to \infty$. For the same value of E_s/N_0, the capacity of the output-quantized Bi-AWGN channel is always smaller than the capacity of the corresponding unquantized channel.

With reference to the last two examples, if Y is allowed to assume $q > 2$ different quantized values (which corresponds to adopting $\lceil \log_2 q \rceil$ quantization bits), the capacity of the obtained channel is upper bounded by that of the unquantized Bi-AWGN channel and is lower bounded by that of the one-bit quantized channel. (Note that the $q - 1$ quantization thresholds shall be properly designed.) In general, the higher q the larger the capacity.

12.1.3 The Channel Coding Theorem

Adopting the formulation in [1], which makes use of the maximum error probability defined in (12.4), Shannon's channel coding theorem can be stated as follows. "For every rate $R < C$ there exists a sequence of $(n, 2^{nR})$ codes for which $\lim_{n\to\infty} P_{e,max}(n) = 0$. Conversely, if $\lim_{n\to\infty} P_{e,max}(n) = 0$ for a sequence of $(n, 2^{nR})$ codes, then $R \leq C$." Note that $\lim_{n\to\infty} P_{e,max}(n) = 0$ implies $\lim_{n\to\infty} P_e(n) = 0$, where $P_e(n)$ is the average error probability defined in (12.5).

Essentially, Shannon's channel coding theorem states that communication over a noisy channel is possible with an arbitrarily small maximum error rate if and only if the code rate of the employed channel code does not exceed the channel capacity. On the other hand, from the proof of the converse, it is possible to show that, when $R > C$, the average probability of error probability is bounded away from zero. Specifically, we have

$$P_e(n) \geq 1 - \frac{C}{R} - \frac{1}{nR} \tag{12.9}$$

$$\rightarrow 1 - \frac{C}{R} \tag{12.10}$$

in the limit where $n \rightarrow \infty$. Inequality (12.9) defines a *non-achievable region* for the considered communication channel. No channel code of length n exists whose average probability of error over the considered channel is smaller than the right-hand side of (12.9). For $n \rightarrow \infty$, the non-achievable region is identified by (12.10). For a channel parametrized by some parameter γ (e.g., the crossover probability p for a BSC, or E_s/N_0 for the Bi-AWGN channel or its output-quantized version), the non-achievable region can be reported in the $P_e(n)$ versus γ plane for a specific code rate R, as illustrated in the following example.

Example 12.4 In Fig. 12.4 the non-achievable region is depicted for both the unquantized Bi-AWGN channel and its one-bit output-quantized version, for code rate $R = 9/10$ and infinite codeword length. Specifically, for fixed $R = 9/10$ the right-hand side of (12.10) is plotted as a function of E_b/N_0 (in logarithmic scale), where $E_b = RE_s$. If E_s is interpreted as the energy per transmitted binary symbol, E_b can be regarded as the energy per information bit. The dashed curve identifies a non-achievable region over the unquantized Bi-AWGN channel (i.e., no $(E_b/N_0, P_e)$ point inside the corresponding area is achievable), while the solid one a non-achievable region over its one-bit output-quantized version. That the unquantized non-achievable region is contained in the quantized one is coherent with the fact that the capacity of the Bi-AWGN channel is larger than the capacity of its output-quantized version, for the same value of E_s/N_0. In general, if $q > 2$

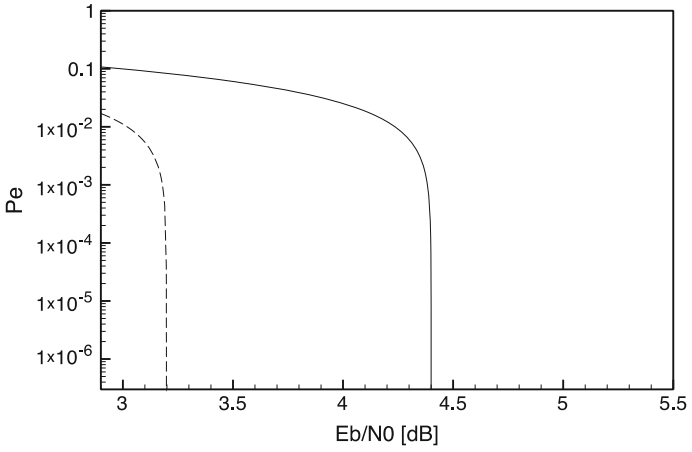

Fig. 12.4 Plot of the Shannon limit for code rate $R = 9/10$, over the Bi-AWGN channel and over the BSC obtained via one-bit quantization of the output of the Bi-AWGN channel

quantization levels are allowed, the corresponding non-achievable region is identified by a curve falling between the two plotted curves. This serves to illustrate how soft information at the decoder can be exploited to improve the system performance. The smallest value of E_b/N_0 for which communication is possible with a vanishing error probability at the given rate $R = 9/10$ over the Bi-AWGN channel is about 3.198 dB. The corresponding value over the one-bit quantized Bi-AWGN channel is about 4.400 dB.

12.2 Maximum a Posteriori and Maximum Likelihood Decoding of Linear Block Codes

As from Sect. 12.1.2, decoding is essentially a decision problem. Given the observation $\boldsymbol{y}$ from the communication channel, the decoder has to decide which of the M codewords has been most likely transmitted, in order to minimize the maximum probability of error. Optimum decoding is based on *maximum a posteriori* (MAP) decision criterion, and consists of assuming as the transmitted codeword the one maximizing the a posteriori probability:

$$\hat{x} = \text{argmax}_x p(x|y).$$

When the codewords are a priori equally likely, then MAP decoding is equivalent to maximum likelihood (ML) decoding, that returns the codeword

$$\hat{x} = \text{argmax}_x p(y|x).$$

It is readily shown that, over a BSC, ML decoding is equivalent to returning the codeword exhibiting the minimum Hamming distance from the received word $\boldsymbol{y}$. (Recall that the Hamming distance between two sequences is the number of positions at which the corresponding symbols are different.) Moreover, over a Bi-AWGN channel, ML decoding consists of returning the codeword (whose symbols belong to the set $\{-\sqrt{E_s}, +\sqrt{E_s}\}$) exhibiting the minimum Euclidean distance from $\boldsymbol{y}$.

Optimum decoding is unfeasible for most codes (including linear codes), due to the need of computing M metrics, with M prohibitively large. Low-density parity-check codes, introduced in Sect. 12.4, are capable to perform close to the Shannon limit at a manageable complexity.

12.3 NAND Flash Memory Channel Model

In NAND flash memories, the generic memory cell is a floating gate transistor. Writing the cell consists of exploiting Fowler-Nordheim tunneling effect [2] to inject a certain amount of charges into the floating gate in order to program the threshold voltage V_{th} of the transistor. For an MLC memory with b bits per cell, there are 2^b nominal values for threshold voltage V_{th}, each bijectively associated with a word of b bits. (There are two nominal values for V_{th} in the particular case of an SLC memory.) The whole range of possible values of V_{th} is then partitioned into 2^b intervals, each corresponding to a nominal value of the threshold voltage.

Reading a cell is a decision problem consisting of picking one of the 2^b nominal values of V_{th} and forwarding the corresponding binary b-tuple. The value of V_{th}, however, cannot be observed directly. In order to read the cell, a word-line voltage must be applied and the corresponding transistor drain current measured. In this chapter, we refer to the word-line voltage simply as the "read voltage", denoting it by V_{READ}. If for some V_{READ} a sufficiently high drain current is detected then we conclude that $V_{READ} > V_{th}$, otherwise we conclude that $V_{READ} < V_{th}$. In this sense, the application of a specific read voltage value is capable to provide exactly one bit of information. Therefore, in order to read the full content of a cell in an MLC memory the drain current must be analyzed for a sufficiently large number of read voltage values. A single V_{READ} value is sufficient in the SLC case unless we wish to extract some soft information to improve the performance of the adopted error control coding scheme.

In ideal flash memories, after a cell is written the corresponding value of V_{th} is exactly equal to one of the 2^b nominal values. In real memories, however, the actual value of V_{th} may differ, even significantly, from its nominal value due to a number of possible physical impairments. For a thorough description of these impairments we refer the reader, for example, to [3, Chap. 4], [4]. As such, the actual value of V_{th} may fall into a voltage interval whose nominal voltage threshold is different from the one we attempted to set during the write operation. When this happens the forwarded binary b-tuple after a read operation differs from the one that was written into the cell. A bit error generated by an erroneous decision about the interval of voltage values V_{th} belongs to is called a *raw bit error*, and the probability of occurrence of raw bit errors is called the raw bit error probability.

The raw bit error probability may be analyzed by modeling the threshold voltage V_{th} of the generic cell as a continuous random variable whose pdf is here denoted by $p(V_{th})$. It must be pointed out that $p(V_{th})$ is not constant during the memory lifetime, as it is modified by subsequent write and read operations, leading to a progressive degradation of the channel in terms of increasing raw bit error probability. The threshold voltages for two different memory cells are typically assumed to be independent and identically distributed (i.i.d.) random variables. In the following two subsections, the channel model for SLC and MLC flash memories is addressed.

12.3.1 SLC Channel Model

The simplest channel model for an SLC flash memory consists of modeling the threshold voltage V_{th} of the generic cell as the weighted sum (with the same weights) of two independent Gaussian random variables with the same variance σ^2 neglecting that, in principle, Gaussian random variables assume their values over an infinite range. The mean values of the two Gaussian distributions are the two nominal values of the threshold voltage, namely, $V_{th,1}$ and $V_{th,2}$ where we assume $V_{th,1} < V_{th,2}$. Let $X \in \{0, 1\}$ be a Bernoulli random variable with equiprobable values, representing the bit originally written into the memory cell. Moreover, let Y be the symbol read from the cell. Conditionally to X, the threshold voltage V_{th} is a Gaussian random variable with variance σ^2 and whose mean is $V_{th,1}$ if $X = 1$ (erase state) and $V_{th,2}$ if $X = 0$. This is depicted in Fig. 12.5. Overall, we have

$$p(V_{th}) = \frac{1}{2}p(V_{th}|X=1) + \frac{1}{2}p(V_{th}|X=0)$$
$$= \frac{1}{\sqrt{8\pi\sigma^2}}\left(e^{-\frac{(V_{th}-V_{th,1})^2}{2\sigma^2}} + e^{-\frac{(V_{th}-V_{th,2})^2}{2\sigma^2}}\right).$$

If we apply only one read voltage $V_{th,1} < V_{READ,1} < V_{th,2}$ we get information about the actual value of V_{th} being larger or smaller than the applied read voltage value. Hence, if only one read voltage value is used, Y is a Bernoulli random

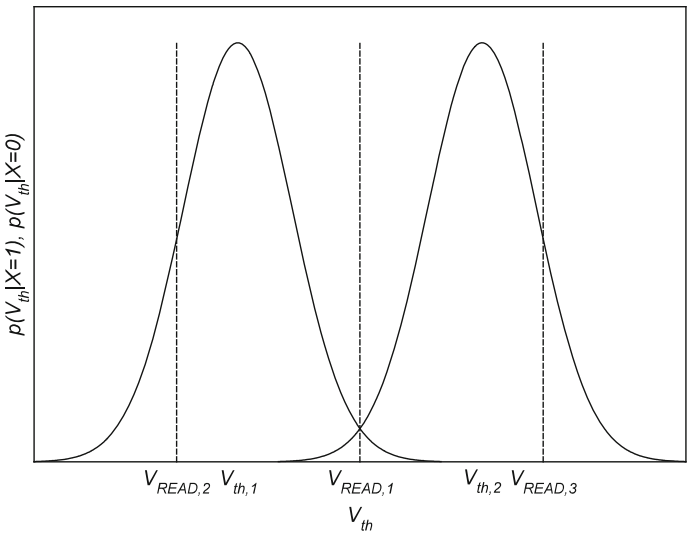

Fig. 12.5 Plot of $p(V_{th}|X=1)$ and $p(V_{th}|X=0)$ for an SLC flash memory where the threshold voltage V_{th} is modeled as the sum of two independent and identically distributed (i.i.d.) Gaussian random variables

variable as well as X. In particular, we have $Y=1$ if $V_{th}<V_{READ}$ is detected, and $Y=0$ otherwise. A raw bit error occurs any time $Y\neq X$, and the raw bit error probability is trivially minimized by setting $V_{READ,1}=(V_{th,1}+V_{th,2})/2$, as depicted in Fig. 12.5. In this situation, the channel is clearly equivalent to the cascade of a Bi-AWGN channel and a one-bit quantizer described in Example 12.2 (i.e., to a BSC), and the raw bit error probability is given by (12.8) where $E_s/N_0=(V_{th,2}-V_{READ,1})^2/2\sigma^2$. At the beginning of the memory life, σ^2 is very small and the memory is almost ideal. Then, σ^2 increases with the memory use, increasing the raw error probability and degrading the channel. A typical value of the raw bit error probability towards the end of the memory life is 10^{-2}.

If an error correcting code is employed to protect the data stored in the flash memory, hard-decision decoding must be necessarily performed if only one V_{READ} value is used as no soft information is available at the decoder. As it will be shown in Sect. 0, however, the availability of soft information at the decoder input represents an essential feature to boost the performance of the coding scheme. In order to provide the decoder with soft information, and consequently to increase its coding gain, more read voltages must be applied sequentially. For example, with reference again to Fig. 12.5 we may employ three read voltage values $V_{READ,1}$, $V_{READ,2}$, and $V_{READ,3}$ and apply two of them for each cell read operation. Specifically, $V_{READ,1}$ is applied at first. if $V_{th}<V_{READ,1}$ then $V_{READ,2}$ is applied to discriminate between $V_{th}<V_{READ,2}$ and $V_{READ,2}<V_{th}<V_{READ,1}$. On the contrary, $V_{READ,3}$ is applied to discriminate between $V_{th}>V_{READ,3}$ and $V_{READ,1}<V_{th}<V_{READ,3}$. In this case the output symbol Y is a discrete random variable assuming the four possible values in the set $\{Y_1,Y_2,Y_3,Y_4\}$ and the channel may be represented as the DMC depicted in Fig. 12.6.

Each arrow in the depicted DMC is associated with a transition probability $p(y|x)$, where the transition probabilities depend on the choice of the read voltages $V_{READ,2}$ and $V_{READ,3}$. A "natural" approach to choose them consists of maximizing the mutual information between the random variables X and Y under the setting $\Pr(X=0)=\Pr(X=1)=1/2$. This approach, proposed in [5], may be easily

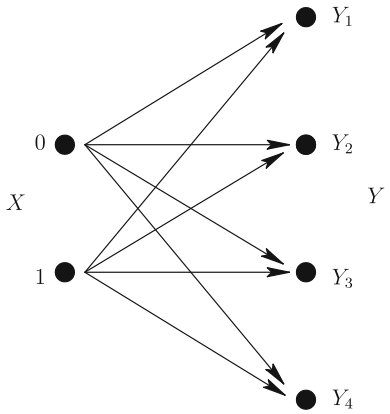

Fig. 12.6 Equivalent channel model for an SLC flash memory where the threshold voltage is modeled as the sum of two i.i.d. Gaussian random variables and where three read voltage values are employed. Each read operation involves two read voltages

extended to any number of read voltages. It may also be easily extended to different choices of the pdf $p(V_{th})$, and therefore to MLC Flash memories.

12.3.2 MLC Channel Model

While the channel model for SLC Flash memories is rather well-established, the development of an MLC channel model is still a subject of research and measurement campaigns, and several models may be found in the literature. These models typically assume the random variable V_{th} to be the weighted sum (with the same weights) of 2^b independent random variables, each corresponding to a nominal value of the threshold voltage. Among these models, the one described next has been adopted in several works [6]. Letting $\boldsymbol{X}$ denote the binary b-tuple that was written in the cell, the pdf $p(V_{th}|X_1 = 11\ldots1)$ associated with the lowest nominal threshold voltage value $V_{th,1}$ (erase state) is modeled as Gaussian with mean $V_{th,1}$ and variance σ_0^2, while the pdf $p(V_{th}|X_i)$ associated with any other nominal value $V_{th,i}$ $(X_i \neq 11\ldots1)$ is characterized by a uniform central region of size ΔV centered in the mean value $V_{th,i}$ and by two Gaussian tails of variance $\sigma^2 < \sigma_0^2$. Formally, for $i \in \{2, 3, \ldots, 2^b\}$ we have

$$p(V_{th}|X_i) = \begin{cases} \frac{1}{\sqrt{2\pi\sigma^2 + \Delta V}} e^{-\frac{(V_{th} - V_{th,i} - \Delta V/2)^2}{2\sigma^2}} & V_{th} > V_{th,1} + \frac{\Delta V}{2} \\ \frac{1}{\sqrt{2\pi\sigma^2 + \Delta V}} & V_{th,1} - \frac{\Delta V}{2} < V_{th} < V_{th,1} + \frac{\Delta V}{2} \\ \frac{1}{\sqrt{2\pi\sigma^2 + \Delta V}} e^{-\frac{(V_{th} - V_{th,i} + \Delta V/2)^2}{2\sigma^2}} & V_{th} < V_{th,1} - \frac{\Delta V}{2} \end{cases}$$

and

$$p(V_{th}) = \frac{1}{2^b} \sum_{i=1}^{2^b} p(V_{th}|X_i).$$

A pictorial representation of the four conditional pdfs $p(V_{th}|X_i)$, $i \in \{1, 2, 3, 4\}$, for an MLC flash memory with $b = 2$ bits per cell and equally spaced threshold voltages is shown in Fig. 12.7.

In an analogous way as for the SLC case, a read is performed by applying sequentially a certain number of read voltages V_{READ} in order to identify the interval in which the actual value of the threshold voltage belongs. If $N \geq 2^b - 1$ different read voltages are employed, the equivalent communication channel is a DMC with 2^b equiprobable input symbols $\boldsymbol{X}$ and $N + 1$ output symbols $\boldsymbol{Y}$. Again, the larger the number of employed read voltages (i.e., the larger the number of intervals in which the range of possible V_{th} values is partitioned) the more accurate the soft information at the decoder input, the lower the bit error rate after decoding. Again, the values of the N read voltages must be properly designed, for instance, maximizing

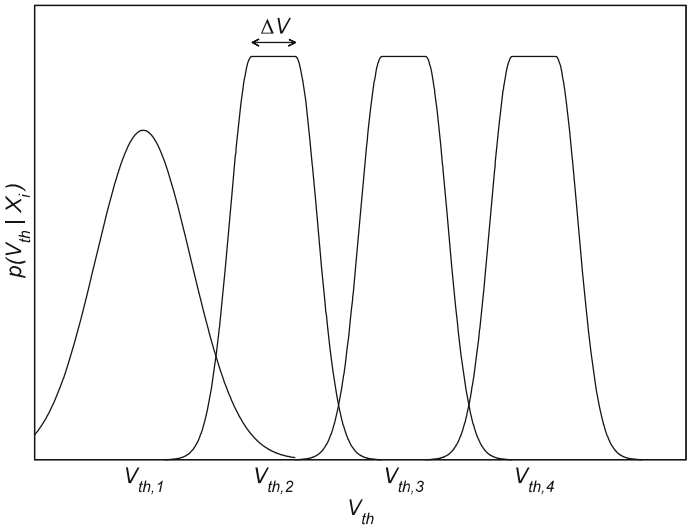

Fig. 12.7 Representation of the four conditional probability density functions $p(V_{th}|X_i)$ of the threshold voltage in an MLC flash memory with $b=2$ bits per cell

the mutual information $I(X;Y)$ under the assumption $\Pr(X=X_i)=2^{-b}$ for all $i\in\{1,2,\ldots,2^b\}$.

12.4 Low-Density Parity-Check Codes

Low-density parity-check (LDPC) codes were introduced by Gallager in [7] and have been almost forgotten for about 30 years. They gained a new interest only after the discovery of turbo codes [8], when it was shown that iterative decoding schemes can attain performances very close to the Shannon limit with a manageable complexity [9, 10].

A binary LDPC code is defined as a binary linear block code whose parity-check matrix $\boldsymbol{H}$ is characterized by a relatively small number of 1 entries, i.e., whose parity-check matrix is sparse. LDPC codes are often represented graphically through a bipartite graph $G=(\mathcal{V}\cup\mathcal{C},\mathcal{E})$ called the Tanner graph [11]. In the Tanner graph there are two different types of nodes, namely, the variable nodes (whose set is $\mathcal{V}$) and the check nodes (whose set is $\mathcal{C}$). The n variable nodes and the m check nodes are associated in a bijective way with the n encoded bits of the generic codeword and with the m parity-check equations, respectively. Each edge $e\in\mathcal{E}$ in the Tanner graph connects a variable node $V\in\mathcal{V}$ with a check node $C\in\mathcal{C}$ if and only if the bit corresponding to V is involved in the parity-check equation corresponding to C. Note that in general not all the m parity-check equations may be linearly independent, so that the actual code rate R of the LDPC code fulfills

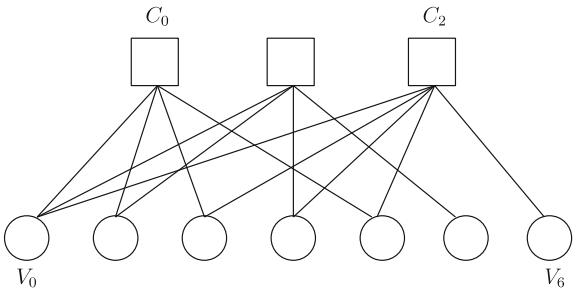

Fig. 12.8 Tanner graph of a $(7,4)$ Hamming code represented by $H = [1110100, 1101010, 10111001]$. There are seven variable nodes variable nodes $\{V_0, \ldots, V_6\}$, one for each encoded bit, and three check nodes, $\{C_0, \ldots, C_2\}$, one for each parity-check equation

$$R \geq \frac{n-m}{n}$$

where equality holds when all m equations are independent. In the Tanner graph of an LDPC code a cycle (or loop) is any closed path starting from a node and ending on the same node. The length of a cycle is the number of edges involved in the cycle. Moreover, the girth g of the Tanner graph is the length of its shortest loop. For reasons that will be clear in the next section, the Tanner graph of an LDPC code should exhibit a large girth. In the Tanner graph, the degree of a variable node or check node is the number of edges incident to it. An LDPC code is said to be regular if all of its variable nodes have the same degree and all of its check nodes have the same degree, and is said to be irregular otherwise.

The representation of LDPC codes in terms of their Tanner graphs is very convenient in order to describe their iterative decoding algorithm, known as belief propagation (BP). In fact, as it will be addressed in Sect. 12.5, BP decoding of LDPC codes may be interpreted as an iterative exchange of messages between the variable nodes and the check nodes along the edges of the Tanner graph. In principle, the Tanner graph can be drawn for any $\boldsymbol{H}$ matrix of any linear block code. As an example, in Fig. 12.8 the Tanner graph is depicted for the $(7,4)$ Hamming code represented by $H = [1110100, 1101010, 10111001]$.

In this section we provide a few details about binary LDPC code design, while LDPC decoding is discussed in the next section. One of the major issues in LDPC coding is represented by efficient encoding, i.e., the efficient computation of the encoded codeword of n bits from a message W represented by a binary k-tuple. Hence, we focus on the design of quasi-cyclic LDPC (QC-LDPC) codes based on circulant matrices, a class of LDPC codes characterized by low-complexity encoding and good performances [12]. In general, a linear block code is said to be quasi-cyclic when there exists some positive integer q such that a cyclic shift by q positions of any codeword results in another codeword. The encoder of

QC-LDPC codes may be implemented very efficiently in hardware using shift register-based circuits [13]. Efficient hardware implementations for the decoder are also available [14].

12.4.1 LDPC Code Ensembles

As opposed to classical algebraic codes, LDPC codes are typically analyzed in terms of average ensemble properties, where an LDPC code ensemble is formed by all LDPC codes having the same codeword length n and nominally the same rate R, and sharing common properties. This approach was introduced by Gallager to analyze his regular LDPC codes [7], and has been successfully adopted to design irregular LDPC codes performing very close to the Shannon limit [15, 16].

An example of LDPC code ensemble is the *unstructured* irregular one [15]. Let n and m be the numbers of variable and check nodes, respectively. Moreover, let Λ_i and P_i be the fractions of variable nodes and check nodes of degree i, respectively. Hence, in the Tanner graph there are $\Lambda_i n$ variable nodes with i sockets and $\mathrm{P}_i m$ check nodes with i sockets and the number of edges is $E = n \sum_{i=2}^{D} i\Lambda_i = m \sum_{i=2}^{H} i\mathrm{P}_i$ where D is the maximum variable node degree and H the maximum check node degree. For given Λ_i, $i = 2, \ldots, D$ and P_i, $i = 2, \ldots, H$,[2] the unstructured $\mathcal{C}(n, \Lambda, \mathrm{P})$ ensemble includes all LDPC codes corresponding to all possible $E!$ edge permutations between the variable node and the check node sockets, according to a uniform probability distribution.

Another example is the *protograph* ensemble [17] (see also the work [18] on LDPC codes from superposition). A protograph is defined as a small Tanner graph and represents the starting point to derive a larger Tanner graph via a "copy-and-permute" procedure. Specifically, the protograph is first copied Q times. Then, the edges of the individual replicas are permuted among the replicas, leading to a larger graph. The edge permutation is performed in such a way that, if an edge e connects a variable node V to a check node C in the protograph, then in the final graph any of the Q replicas of e may connect only a replica of V to a replica of C. Note that, while parallel edges between nodes are allowed in the protograph, they are avoided in the permutation phase. An example of this copy-and-permute procedure is depicted in Fig. 12.9. For a given protograph and a given Q the ensemble is composed of the LDPC codes corresponding to all possible edge permutations fulfilling the described constraints (again, the probability distribution over such permutations is uniform).

[2]For unstructured ensemble, the minimum variable and check nodes are usually set to. The reason for this choice is out of the scope of this chapter.

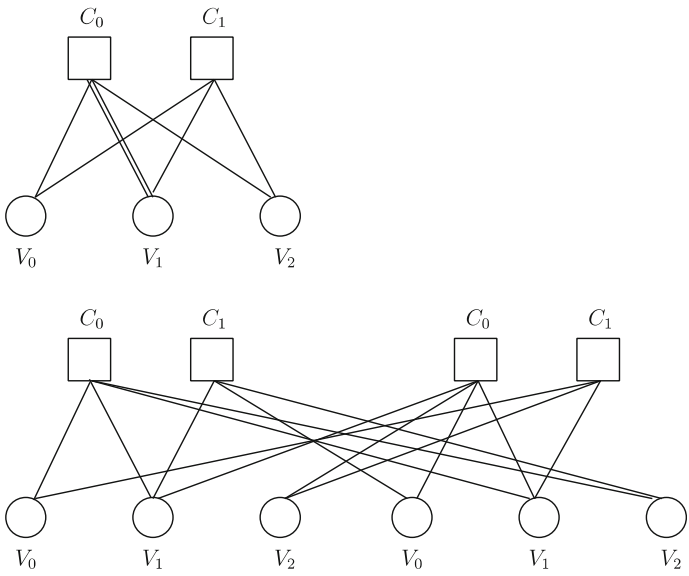

Fig. 12.9 Conceptual example of copy-and-permute protograph procedure

12.4.2 QC-LDPC Codes Construction

A very popular technique to design finite length LDPC codes consists of two subsequent steps. An ensemble of LDPC codes with desired properties is first designed and then a code from the ensemble is picked constructing its Tanner graph according to some graph-lifting algorithm. In the first design phase (ensemble optimization) *asymptotic ensembles* are considered, i.e., ensembles of LDPC codes whose codeword length tends to infinity (examples are the unstructured $\mathcal{C}(\infty, \Lambda, \mathrm{P})$ ensemble and the protograph ensemble defined by a specific finite-length protograph in the limit where $Q \to \infty$). The main parameter characterizing an asymptotic ensemble of LDPC codes under iterative decoding is the *asymptotic decoding threshold* [19, 15]. Letting ℓ be the iteration index and assuming that the communication channel is parameterized by some real parameter θ such that $\theta_1 < \theta_2$ means that the channel corresponding to θ_2 is a degraded version of the channel corresponding to θ_1, the asymptotic threshold θ^* is defined as

$$\theta^* = \sup\left\{\theta \,\text{s.t.}\, P_{e,\ell}^{\infty} \to 0 \text{ as } \ell \to \infty\right\}$$

where $P_{e,\ell}^{\infty}$ is the average error probability under iterative decoding over the asymptotic ensemble (i.e., the expected probability of error for an LDPC code randomly picked in the asymptotic ensemble). For example, over a BSC the parameter θ is the crossover probability p, while over a Bi-AWGN channel it is the noise power σ^2 for given E_s (therefore over the Bi-AWGN channel the threshold

may be expressed as $(E_b/N_0)^*$ where $E_b = RE_s$ and R is the nominal ensemble rate). Note that for the same ensemble, the threshold is different for different message passing decoders. For unstructured ensembles the threshold may be calculated exactly via a procedure called *density evolution* [15] or approximately via a tool known as EXIT chart [20]. For protograph ensembles it may be calculated with good approximation via multi-dimensional EXIT analysis [21]. In Sect. 12.6.2 density evolution is reviewed for unstructured regular LDPC ensembles and for a very simple decoder called the Gallager B decoder.

Once a protograph ensemble with a satisfying threshold over the channel of interest has been designed, a QC-LDPC code can be constructed from the protograph. This step is usually performed by first representing the protograph as a *base matrix* $\boldsymbol{B}$. The number of rows and columns in the base matrix equal the number of check and variable nodes in the protograph, respectively. Moreover, the (j, i)th entry of $\boldsymbol{B}$ is equal to the number of connection between check node C_j and variable node V_i in the protograph. For example, the base matrix corresponding to the protograph depicted in Fig. 12.9 is

$$B = \begin{bmatrix} 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

In order to construct the parity-check matrix $\boldsymbol{H}$ of a QC-LDPC code from $\boldsymbol{B}$, each entry in the base matrix is replaced with a $Q \times Q$ circulant matrix, where a circulant matrix is any square matrix such that every row is obtained from the previous row by a cyclic shift to the right by one position. An entry in $\boldsymbol{B}$ equal to t is replaced by a circulant matrix whose rows and columns all have Hamming weight t. (Null entries in $\boldsymbol{B}$ are replaced by zero $Q \times Q$ square matrices.) If the number of variable nodes in the protograph is n_p then the final LDPC code has length Qn_p. Moreover, it is a QC-LDPC code as the cyclic shift of any codeword by n_p positions results in another codeword. The specific circulant matrices used to replace the entries of the base matrix are chosen according to algorithms aimed at increasing the girth g of the graph, making it suitable to iterative message-passing decoding. It is pointed out that sometimes the parity-check matrix $\boldsymbol{H}$ is obtained by lifting the base matrix in several steps. For example, instead of replacing each entry of $\boldsymbol{B}$ by a $Q \times Q$ matrix (for large Q), $\tilde{Q} \times \tilde{Q}$ circulant matrices may be used at first, with Q being a multiple of $\tilde{Q}$, and then circulant permutation matrices of size $Q/\tilde{Q}$ may replace each entry in the "intermediate" matrix.[3]

[3]The described protograph-based technique is not the only one to construct good QC-LDPC codes. Another possible approach is based on Euclidean and projective finite geometries [22, 23].

12.4.3 Error Floor

Finite length LDPC codes are affected by a phenomenon known as the "error floor" [24, 25]. Considering again a communication channel parameterized by a real parameter θ indicating the level of channel noise, the error floor consists of a sudden reduction in the slope of the LDPC code performance curve when θ becomes lower than some value. For example, over the BSC the error floor appears at sufficiently low values of the error probability p, while over the Bi-AWGN channel it appears at sufficiently high values of E_b/N_0. An example performance curve in term of bit error rate (BER) versus E_b/N_0 exhibiting an error floor is depicted in Fig. 12.10. In NAND Flash memories applications, very pressing requirements are usually imposed on the error floor. More specifically, it is often required that the error floor must not appear above page error rate (i.e., codeword error rate) 10^{-15}.

The error floor of LDPC codes under belief propagation decoding is mainly due to graphical structures in the Tanner graph called *trapping sets* [25]. Given a subset $\mathcal{W}$ of the variable nodes, the subgraph induced by $\mathcal{W}$ is the bipartite graph composed of $\mathcal{W}$, of the subset $\mathcal{U}$ of check nodes connected to $\mathcal{W}$ and of the corresponding edges. By definition, an (a, b) trapping set is any size-a subset $\mathcal{W}$ of the variable nodes, such that there are exactly b check nodes of odd degree (an arbitrary number of check nodes of even degree) in the corresponding induced subgraph. The parameter a is called the size of the trapping set. If there are only degree-1 and degree-2 check nodes in the induced subgraph, then the trapping set is said to be *elementary*. Elementary trapping sets of small size are a major cause of error floor

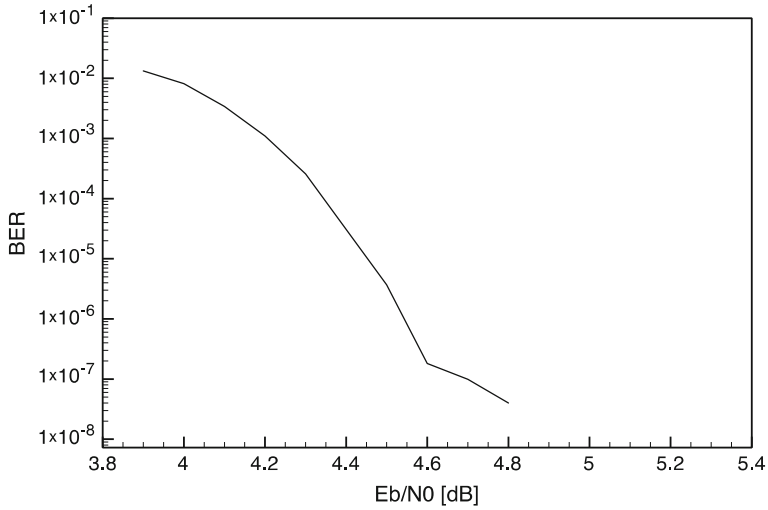

Fig. 12.10 Performance curve (in terms of BER vs. E_b/N_0) exhibiting an error floor at BER $\approx 10^{-7}$ ($E_b/N_0 > 4.6$ dB)

for iteratively decoded LDPC codes. We point out that small weight codewords may also contribute to the error floor together with trapping sets.

The need to construct LDPC codes characterized by very low error floors imposes some modifications to the QC-LDPC code design procedure described in the previous subsection, which becomes more involved. The asymptotic decoding threshold is not the only metric to be taken into account during the ensemble optimization phase, as other asymptotic parameters such as the typical relative minimum distance or smallest trapping set size must be considered [26, 27]. We also point out that reliable error floor analysis at very low error rates of LDPC codes for storage applications still represents an open issue. In fact, Monte Carlo software simulation is not feasible at very low error rates because of prohibitively long simulation times. Approaches proposed in the literature are hardware simulation, importance sampling [6, 28], and estimation techniques [29].

12.5 Belief Propagation (BP) Decoding of LDPC Codes

12.5.1 Introduction

As opposed to MAP and ML decoding algorithms (Sect. 12.2), that are block-wise algorithms, BP is a *bit-wise* decoding algorithm, working iteratively. More specifically, at the end of each decoding iteration a separate decision is taken about each bit in the codeword, and then it is checked whether the currently decoded hard-decision sequence is a codeword or it is not. Letting $y = [y_0, y_1, \ldots, y_{n-1}]$ denote the sequence outcoming from the communication channel, the decision about encoded bit c_i, $i = 0, \ldots, n-1$, is taken according to its a posteriori likelihood ratio (LR), namely,

$$L(c_i|y) = \frac{\Pr(c_i = 0|y)}{\Pr(c_i = 1|y)} \underset{\hat{c}_i = 1}{\overset{\hat{c}_i = 0}{\gtrless}} 1.$$

Unfortunately, the only information available at variable node i at the beginning of the decoding process is the a priori LR

$$L(c_i|y_i) = \frac{\Pr(c_i = 0|y_i)}{\Pr(c_i = 1|y_i)}$$

i.e., the LR conditioned only to the local observation, not the a posteriori LR $L(c_i|y)$ as required. Indeed, the task of the BP decoder consists of calculating the a posteriori LR for each variable node, starting from the individual a priori LRs, exploiting an iterative exchange of information among the nodes of the bipartite graph. In the following description of the BP decoder, we will not make any

assumption on the communication channel, but that the channel is memory-less with binary input and equally likely input values.

12.5.2 Preliminaries

We start with some preliminary material that will be useful to properly describe BP decoding of LDPC codes.

Let us consider a Bernoulli random variable B taking the values 0 and 1 with equal probabilities. As depicted in Fig. 12.11, assume that N random experiments are performed to get information about the value assumed by B and that all these experiments are independent. The outcome of the n-th experiment (n-th observation) is denoted by ω_n, while the vector of N observables by $\boldsymbol{\omega} = [\omega_1, \omega_2, \ldots, \omega_N]$. We define the likelihood ratio (LR) of B conditioned to the observation ω_n as

$$L(B|\omega_n) = \frac{\Pr(B=0|\omega_n)}{\Pr(B=1|\omega_n)} \tag{12.11}$$

and the a posteriori likelihood ratio of B (i.e., conditioned to the whole set of N independent observations), as

$$L(B|\boldsymbol{\omega}) = \frac{\Pr(B=0|\boldsymbol{\omega})}{\Pr(B=1|\boldsymbol{\omega})}. \tag{12.12}$$

We now seek for an expression of the a posteriori LR, $L(B|\boldsymbol{\omega})$, as a function of the individual LRs, each conditioned to a specific observation. By Bayes rule we have

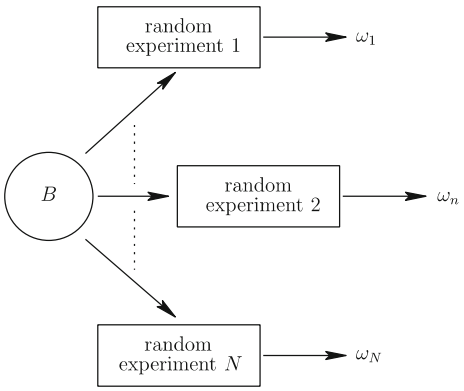

Fig. 12.11 N random experiments are conducted to obtain some information about the value of a Bernoulli random variable B. The observation associated with the n-th random experiment is ω_n

$$\begin{aligned} L(B|\boldsymbol{\omega}) &= \frac{p(\boldsymbol{\omega}|B=0)}{p(\boldsymbol{\omega}|B=1)} \\ &= \prod_{n=1}^{N} \frac{p(\omega_n|B=0)}{p(\omega_n|B=1)} \\ &= \prod_{n=1}^{N} L(B|\omega_n), \end{aligned} \tag{12.13}$$

where the second equality follows from independence of the random experiments.

We also observe that, through (12.12) and the relationship $\Pr(B=0|\boldsymbol{\omega}) + \Pr(B=1|\boldsymbol{\omega}) = 1$, the probabilities $\Pr(B=0|\boldsymbol{\omega})$ and $\Pr(B=1|\boldsymbol{\omega})$ may be expressed as functions of the a posteriori LR as follows:

$$\Pr(B=0|\boldsymbol{\omega}) = \frac{L(B|\boldsymbol{\omega})}{1+L(B|\boldsymbol{\omega})}, \tag{12.14}$$

$$Pr(B=1|\boldsymbol{\omega}) = \frac{1}{1+L(B|\boldsymbol{\omega})}. \tag{12.15}$$

This is sometimes referred to as *soft bit*. Analogous relationships may be derived for $\Pr(B=0|\omega_n)$ and $\Pr(B=1|\omega_n)$.

Next, consider n statistically independent Bernoulli random variables $B_1, B_2, \ldots, B_n$ each taking its value in $\{0, 1\}$. We allow $\Pr(B_k=1) \neq \Pr(B_l=1)$ if $k \neq l$. We ask what is the probability that the n variables sum to 0 (in binary algebra), i.e., the probability that an even number of such random variables take value 1. This problem was solved in [7], where it was shown that

$$\Pr(B_1 + B_2 + \cdots + B_n = 0) = \frac{1 + \prod_{k=1}^{n} (1 - 2\Pr(B_k=1))}{2}. \tag{12.16}$$

Consider now n Bernoulli random variables $B_1, B_2, \ldots, B_n$ fulfilling a parity constraint $B_1 + B_2 + \cdots + B_n = 0$. Moreover, assume that some reliability information is known about variables $B_1, \ldots, B_{i-1}, B_{i+1}, \ldots, B_n$, in terms of LRs $L(B_k)$, $k \in \{1, \ldots, i-1, i+1, \ldots, n\}$ and that $B_1, \ldots, B_{i-1}, B_{i+1}, \ldots, B_n$ are statistically independent. We seek for an expression of the LR $L(B_i)$, conditional on all available information about the other $n-1$ variables. Since $\Pr(B_i=0) = \Pr(B_1 + \cdots B_{i-1} + B_{i+1} + \cdots + B_n = 0)$, through (12.16) we obtain

$$\begin{aligned} &\Pr(B_i=0|L(B_1), \ldots, L(B_{i-1}), L(B_{i+1}), \ldots, L(B_n)) \\ &= \frac{1 + \prod_{k \neq i} (1 - 2\Pr(B_k=1))}{2} \end{aligned}$$

and, consequently,

$$\begin{aligned}&\Pr(B_i=1|L(B_1),\ \ldots,L(B_{i-1}),L(B_{i+1}),\ \ldots,L(B_n))\\&=\frac{1-\prod_{\mathrm{k}\neq\mathrm{i}}(1-2\Pr(B_k=1))}{2}.\end{aligned}$$

Note that each term $\Pr(B_k=1)$ involved in the multiplication may be expressed in terms of the corresponding $L(B_k)$ through (12.15). From the term-by-term ratio between these two latter equations, we obtain

$$\mathrm{L}(B_i|L(B_1),\ \ldots,L(B_{i-1}),L(B_{i+1}),\ \ldots,L(B_n))=\frac{1+\prod_{\mathrm{k}\neq\mathrm{i}}(1-2\Pr(B_k=1))}{1-\prod_{\mathrm{k}\neq\mathrm{i}}(1-2\Pr(B_k=1))}.$$

Through (12.15), after a few calculations this leads to

$$\mathrm{L}(B_i|L(B_1),\ \ldots,L(B_{i-1}),L(B_{i+1}),\ \ldots,L(B_n))=\frac{\prod_{k\neq i}\frac{L(B_k)+1}{L(B_k)-1}+1}{\prod_{k\neq i}\frac{L(B_k)+1}{L(B_k)-1}-1}. \tag{12.17}$$

12.5.3 Algorithm Description

12.5.3.1 Overview

For ease of presentation, in the description of the algorithm we omit the decoding iteration index. We denote by r_i^j the message sent by variable node V_i, $i=0,\ \ldots,n-1$, to check node C_j, $j=0,\ \ldots,m-1$ during the current iteration, and by m_j^i the message sent back by check node C_j, to variable node V_i, during the same iteration. For $i=0,\ \ldots,n-1$, we also denote by w_i the a priori LR for variable node V_i, i.e.,

$$w_i=\frac{\Pr(c_i=0|y_i)}{\Pr(c_i=1|y_i)}.$$

This is illustrated in Fig. 12.12.

Belief-propagation decoding is composed of four steps, namely[4]:

- initialization;
- horizontal step;

[4]The words "horizontal" and "vertical" remind us that the check nodes and the variable nodes are associated with the rows and the columns of the parity-check matrix, respectively.

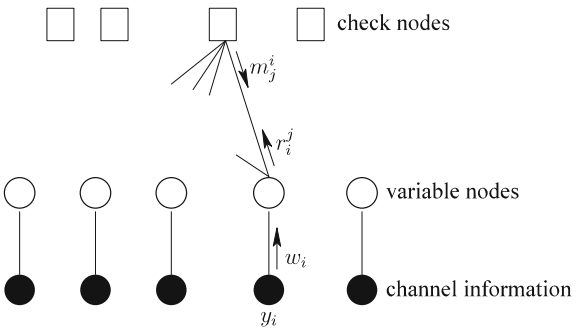

Fig. 12.12 Tanner graph of an LDPC code. The message sent by variable node V_i to check node C_j and the message sent by check node C_j to variable node V_i are denoted by r_i^j and m_j^i, respectively

- vertical step;
- hard decision and stopping criterion step.

Out of them, the initialization step is executed only once, at the beginning of decoding. The other three steps are executed iteratively, until a termination condition is verified or a maximum number of iterations, denoted by I_{max}, is reached. Each decoding iteration is split into two half-iterations. During the first half-iteration (horizontal step), check nodes process messages incoming from their neighboring variable nodes. Then, each check node sends one message along every edge incident on it. Thus, every check node sends one message per iteration to each of its neighboring variable nodes. During the second half-iteration (vertical step) variable nodes process messages incoming from their neighboring check nodes. Similar to the previous half-iteration, at the end of this processing each variable node sends one message along each edge incident on it. Thus, every variable node sends one message per iteration to each of its neighboring check nodes. At the end of the two half-iterations, a hard decision is taken in each variable node, about the value of the corresponding encoded bit.

The message transmitted by check node C_j, $j = 0, \ldots, m-1$, to variable node V_i, $i = 0, \ldots, n-1$, where V_i belongs to the neighborhood of C_j, may be interpreted as the best estimate C_j has about the value of V_i up to the current iteration. This is the estimate of the value of V_i given all information about V_i the check node has got from the variable nodes connected to it *other than* V_i. This is known as *extrinsic information*. Analogously, the message sent back by variable node V_i to check node C_j may be interpreted as the best estimate V_i has about itself up to the current iteration. This is the estimate of its value given all information the variable node has got from the communication channel and from the check nodes connected to it *other than* C_j (extrinsic information). All messages exchanged between variable nodes and check nodes are LRs or, equivalently, soft bits.

At the end of the vertical step, each variable node takes a hard decision about the value of its associated bit, based on the a priori information incoming from the channel and on all estimates incoming from the check nodes connected to it. If the obtained hard-decision binary sequence $\hat{c}$ is a codeword of the LDPC code, i.e., if every check node is connected to an even number of variable nodes whose

current estimate is 1, then a decoding success is declared, decoding is terminated, and $\hat{c}$ is returned as the decoded codeword. Otherwise, a new iteration is started, unless the maximum number of iterations has been reached. In this latter case, no codeword has been found and a decoding failure is declared. LDPC codes decoded via belief propagation are then characterized by two different error events: detected errors and undetected errors. A detected error takes place whenever no codeword is found up to the maximum number of iterations. An undetected error takes place whenever, at some iteration, the hard-decision sequence $\hat{c}$ is a codeword but not the transmitted one. Undetected errors may be extremely dangerous is some contexts, including NAND Flash memories.

12.5.3.2 Initialization

At the beginning, each variable node broadcasts to all its neighboring check nodes the a priori LR received from the communication channel. Hence, we have

$$r_i^j = w_i$$

for all $j \in N(i)$, where $N(i)$ is the set of indexes of check nodes connected to V_i. The expression of w_i depends on the nature of the channel. For example, it is easy to check that over a BSC with error probability p and antipodal mapping $x_i = 1 - 2c_i \in \{-1, +1\}$, we have

$$w_i = \begin{cases} \frac{1-p}{p} & \text{if } y_i = +1 \\ \frac{p}{1-p} & \text{if } y_i = -1. \end{cases} \tag{12.18}$$

As another example, over a Bi-AWGN channel and again antipodal mapping $x_i = 1 - 2c_i$, (meaning E_s normalized to 1) we have

$$w_i = e^{(2/\sigma^2)y_i}. \tag{12.19}$$

Importantly, the initialization step requires a knowledge of the channel. For instance, in the case of a BSC the error probability p must be known, as well as the noise power σ^2 in the Bi-AWGN case.

12.5.3.3 Horizontal Step

For $j = 0, \ldots, m-1$, check node C_j, of degree h_j, sends to each of the h_j variable nodes connected to it its current estimate of the corresponding bit. If variable node V_i is connected to C_j, the message from C_j to V_i is the LR of bit c_i, conditional on the information available at C_j incoming from all its neighboring variable nodes, except the information incoming from V_i. A pictorial representation of this process

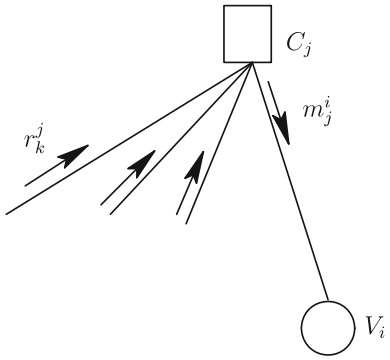

Fig. 12.13 Check node processing of incoming messages during the horizontal step

is provided in Fig. 12.13. Note that two different variable nodes connected to C_j will receive, in general, different messages.

The message m_j^i from C_j to V_i can be calculated exploiting one of the results introduced in Sect. 12.5.2. In fact, each of the h_j incoming messages is the LR of a specific bit on which the check node imposes a parity constraint. Hence, under independence hypothesis, denoting by $N(j)\backslash\{i\}$ the set of indexes of variable nodes connected to C_j except V_i, from (12.17) we immediately obtain

$$m_j^i = \frac{\prod_{k \in N(j)\backslash\{i\}} \frac{r_k^j + 1}{r_k^j - 1} + 1}{\prod_{k \in N(j)\backslash\{i\}} \frac{r_k^j + 1}{r_k^j - 1} - 1}. \tag{12.20}$$

Note that the independence hypothesis is fulfilled only during the first $g/2$ decoding iterations, where g is the girth of the Tanner graph. On the other hand, it represents an approximation during all subsequent iterations.

12.5.3.4 Vertical Step

For $i = 0, \ldots, n-1$, variable node V_i, of degree d_i, sends to each of its d_i neighboring check nodes its current estimate of the associated bit. With reference to Fig. 12.14, the message r_i^j sent to check node C_j is the LR about bit c_i, conditional on the a priori information available from the communication channel and on the information incoming from all check nodes connected to it, except C_j. Again, two different check nodes connected to V_i will receive, in general, different messages.

The message r_i^j that variable node V_i sends to check node C_j connected to it can be easily computed based on the result in Sect. 12.5.2. In fact, each of the d_i messages incoming towards the variable node (including the message w_i incoming

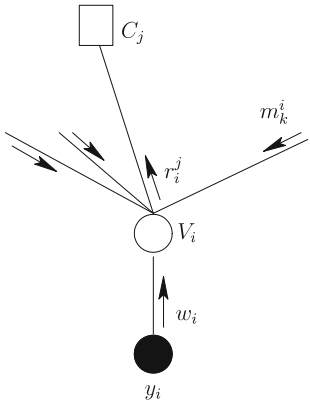

Fig. 12.14 Variable node processing of incoming messages during the vertical step

from the channel), represents the LR of c_i conditioned to some observation. Under the hypothesis of independence for the d_i observations, denoting by $N(i)\backslash\{j\}$ the set of indexes check nodes connected to V_i except check node of index j, we have

$$r_i^j = w_i \prod_{k \in N(i)\backslash\{j\}} m_k^i. \tag{12.21}$$

(Again, the independence hypothesis is valid rigorously only during the first $g/2$ decoding iterations.)

12.5.3.5 Hard Decision and Stopping Criterion

At the last step of each iteration, every variable node takes a decision about its associated encoded bit. This decision is based on all currently available information about the bit, i.e., on the a priori information from the communication channel and on *all* messages incoming from the check nodes. Let m^i denote the list of all messages incoming towards the variable node V_i. Applying again the result developed in Sect. 12.5.2 under the hypothesis of independence of the incoming messages, we may write

$$L\left(c_i | w_i, m^i\right) = w_i \prod_{k \in N(i)} m_k^i. \tag{12.22}$$

(Again, the independence hypothesis is fulfilled rigorously only during the first $g/2$ decoding iterations.) The decision about encoded bit c_i at the end of the generic iteration is then

$$L\left(c_i|w_i, m^i\right) \begin{matrix} \hat{c}_i = 0 \\ \gtrless \\ \hat{c}_i = 1 \end{matrix} 1.$$

If the current hard-decision sequence $\hat{c}$ is a codeword ($\hat{c}H^T = 0$, where $\boldsymbol{H}$ is any parity-check matrix of the code) then the algorithm is terminated and $\hat{c}$ is returned as the decoded codeword. Else, if $\hat{c}$ is not a codeword and the maximum number of iterations I_{max} has been reached, the algorithm is terminated and a failure is reported. Else, a new iteration is started jumping to the horizontal step. Belief propagation decoding of LDPC codes may be summarized as follows.

Belief-Propagation Decoding of LDPC Codes

1: set $I = 1$. For $i = 0, \ldots, n-1$, for $j \in N(i)$, set $r_i^j = w_i$;
2: for $j = 0, \ldots, m-1$
  for $i \in N(j)$ calculate m_j^i according to (11.20);
3: for $i = 0, \ldots, n-1$
  for $j \in N(i)$ calculate r_i^j according to (11.21);
4: for $i = 0, \ldots, n-1$ {
  calculate $L(c_i|w_i, \boldsymbol{m}^i)$ according to (11.22);
  if $L(c_i|w_i, \boldsymbol{m}^i) \geq 1$ then set $\hat{c}_i = 0$;
  else set $\hat{c}_i = 1$;
}
if $\hat{\boldsymbol{c}}\boldsymbol{H}^T = \boldsymbol{0}$ then return $\hat{\boldsymbol{c}}$;
else {
  if $I = I_{max}$ exit;
  else {
    $I = I + 1$;
    goto 2;
    }
}

12.5.4 Log-Domain BP Decoder

The main issue when implementing BP decoding described in Sect. 12.5.3 is represented by the need to handle and combine, through multiplications and divisions, likelihood ratios whose values may differ by several orders of magnitude.

For this reason, a log-domain implementation is usually preferred from an implementation viewpoint. In the log-domain version of BP decoding, log-likelihood ratios (LLRs) of the encoded bits are exchanged between variable and check nodes. Next, we discuss how the above-described BP decoding shall be modified in the log-domain. All logarithms are assumed to be natural logarithms. Moreover, $\mathrm{sgn}(x)$ will denote the sign function, i.e., $\mathrm{sgn}(x) = +1$ if $x \geq 0$ and $\mathrm{sgn}(x) = -1$ otherwise.

The initialization step remains the same, the only difference being that the first message each variable node sends to all its neighboring check nodes is the a priori LLR of the corresponding encoded bit. Neglecting again the iteration index and denoting by R_i^j the message sent from variable node $i \in \{0, \ldots, n-1\}$ to check node $j \in N(i)$, we have

$$R_i^j = W_i,$$

where $W_i = \log w_i$. For instance, assuming antipodal mapping $x_i = 1 - 2c_i$, over a BSC with error probability p we have

$$W_i = \begin{cases} \log \frac{1-p}{p} \, if y_i = +1 \\ \log \frac{p}{1-p} \, if y_i = -1 \end{cases} \tag{12.23}$$

while, over a Bi-AWGN channel,

$$W_i = \frac{2}{\sigma^2} y_i. \tag{12.24}$$

The development of check node message processing (horizontal step) in the log domain is more involved. Denoting $R_i^j = \log r_i^j$ and $M_j^i = \log m_j^i$, from (12.20) we may write

$$\begin{aligned} M_j^i &= \log \frac{\prod_{k \in N(j) \setminus \{i\}} \frac{e^{R_k^j}+1}{e^{R_k^j}-1} + 1}{\prod_{k \in N(j) \setminus \{i\}} \frac{e^{R_k^j}+1}{e^{R_k^j}-1} - 1} \\ &= \log \frac{\prod_{k \in N(j) \setminus \{i\}} \mathrm{sgn}\left(R_k^j\right) \cdot \prod_{k \in N(j) \setminus \{i\}} \frac{e^{\left|R_k^j\right|}+1}{e^{\left|R_k^j\right|}-1} + 1}{\prod_{k \in N(j) \setminus \{i\}} \mathrm{sgn}\left(R_k^j\right) \cdot \prod_{k \in N(j) \setminus \{i\}} \frac{e^{\left|R_k^j\right|}+1}{e^{\left|R_k^j\right|}-1} - 1}, \end{aligned}$$

where we have exploited the fact that any odd function fulfills $f(x) = \mathrm{sgn}(x) f(|x|)$ and the fact that $f(x) = (e^x + 1)/(e^x - 1)$ is odd. The obtained expression of M_j^i can be further developed through the identity $\log((x+1)/(x-1)) = \mathrm{sgn}(x) \cdot \log((|x|+1)/(|x|-1))$ and through the fact that $e^{|R|} \geq 1$. This yields

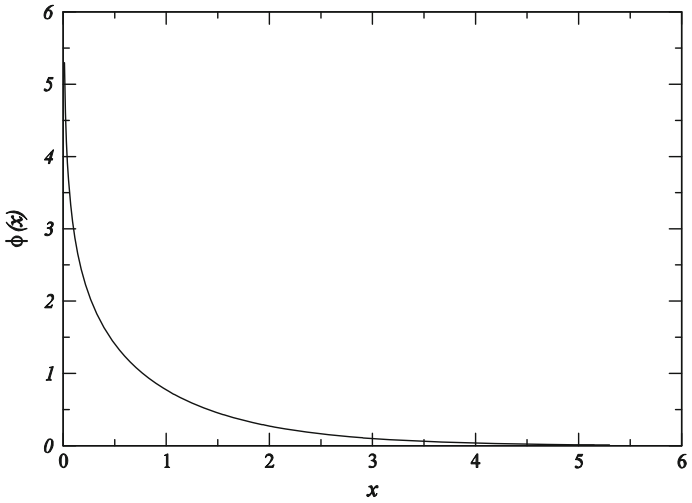

Fig. 12.15 Plot of function $\varphi(x) = -\log(\tanh(x/2))$.

$$
\begin{aligned}
M_j^i &= \prod_{k \in N(j)\setminus\{i\}} \operatorname{sgn}\left(R_k^j\right) \cdot \log \frac{\prod_{k \in N(j)\setminus\{i\}} \frac{e^{\left|R_k^j\right|}+1}{e^{\left|R_k^j\right|}-1} + 1}{\prod_{k \in N(j)\setminus\{i\}} \frac{e^{\left|R_k^j\right|}+1}{e^{\left|R_k^j\right|}-1} - 1} \\
&= \prod_{k \in N(j)\setminus\{i\}} \operatorname{sgn}\left(R_k^j\right) \cdot \log \frac{e^{\sum_{k \in N(j)\setminus\{i\}} \log \frac{e^{\left|R_k^j\right|}+1}{e^{\left|R_k^j\right|}-1}} + 1}{e^{\sum_{k \in N(j)\setminus\{i\}} \log \frac{e^{\left|R_k^j\right|}+1}{e^{\left|R_k^j\right|}-1}} - 1} \\
&= \prod_{k \in N(j)\setminus\{i\}} \operatorname{sgn}\left(R_k^j\right) \cdot \varphi\left(\sum_{k \in N(j)\setminus\{i\}} \varphi\left(\left|R_k^j\right|\right)\right)
\end{aligned}
\tag{12.25}
$$

where, for $x > 0$, we have introduced the nonlinear function

$$
\varphi(x) = \log \frac{e^x + 1}{e^x - 1} = -\log(\tanh(x/2)).
$$

A plot of this function is depicted in Fig. 12.15. Note that the function coincides with its inverse, i.e., $\varphi(\varphi(x)) = x$.

The transposition of the variable node processing (vertical step) to the logarithmic domain is much simpler. In fact, from (12.21) we immediately obtain

$$
R_i^j = W_i + \sum_{k \in N(i)\setminus\{j\}} M_k^i. \tag{12.26}
$$

Analogously, (12.22) shall be updated as

$$\log L\left(c_i|W_i, M^i\right) = W_i + \sum_{k \in N(i)} M_k^i. \tag{12.27}$$

The algorithm may be then summarized as follows.

Log-Domain Belief-Propagation Decoding of LDPC Codes

1: set $I = 1$. For $i = 0, \ldots, n-1$, for $j \in N(i)$, set $R_i^j = W_i$;
2: for $j = 0, \ldots, m-1$
for $i \in N(j)$ calculate M_j^i according to (11.25);
3: for $i = 0, \ldots, n-1$
for $j \in N(i)$ calculate R_i^j according to (11.26);
4: for $i = 0, \ldots, n-1$ {
calculate $\log L(c_i|W_i, \boldsymbol{M}^i)$ according to (11.27);
if $\log L(c_i|W_i, \boldsymbol{M}^i) \geq 0$ then set $\hat{c}_i = 0$;
else set $\hat{c}_i = 1$;
}
if $\hat{\boldsymbol{c}}\boldsymbol{H}^T = \boldsymbol{0}$ then return $\hat{\boldsymbol{c}}$;
else {
if $I = I_{max}$ exit;
else {
$I = I + 1$;
goto 2;
}
}

Although an enhanced numerical stability is achieved operating on log-likelihood ratios, as well as a lower complexity (as, for instance, products in (12.21) and (12.22) are transformed in sums in (12.25) and (12.26), respectively), check node processing in the log-domain imposes the evaluation of the nonlinear function φ. For a single check node C_j of degree h_j, this function should in principle be evaluated $(h_j)^2$ times per iteration (even if techniques to limit the number of φ evaluations exist). The calculation of function φ is typically performed by means of lookup-tables. Note that, however, for small x the graph of $\varphi(x)$ is very steep, thus requiring a very fine (in general, nonuniform) discretization of the corresponding region of the function domain, and that the implementation of $\varphi(x)$ through a lookup table may be quite inconvenient in hardware implementation. For these reasons, extensive work has been carried out to develop either approximations of the log-domain BP decoder or other reduced-complexity decoding schemes.

All of these decoders offer a reduced error correction capability than actual BP. However, they also exhibit a lower decoding complexity and, hence, a higher decoding speed.

12.6 Reduced-Complexity Decoders

So far we have focused on the BP decoder (both in probability domain and log-domain) originally developed by Gallager. Next, we present a few reduced-complexity, implementation-friendly decoders for LDPC codes. It must be pointed out that a large amount of reduced-complexity decoding schemes for LDPC codes have been developed in the last decade [30]. Most of these decoding schemes may be seen as approximations of the BP decoder, in the sense that they are characterized by approximations of the most complex step of BP decoding, namely, the horizontal step (consisting of the calculation of extrinsic messages from the check nodes to the variable nodes). As such, these approximate BP decoding algorithms can be formalized via the same pseudo-code we have adopted for the log-domain BP decoder, with a difference in step 2.

We only present the most famous approximation of the BP decoder, called the Min-Sum (MS) decoder. We then move to describe decoders exhibiting an even lower complexities. More specifically, we present a binary message-passing algorithm known as "Gallager B" (and originally proposed in [7]) and a class of non-message-passing decoders named "flipping algorithms" (the idea of bit flipping appears again in [7]). These very low complexity decoding algorithms (along with some of their modifications, not addressed in this chapter) are of interest in NAND Flash memories at the beginning of the memory life, when the raw bit error probability is extremely low.

12.6.1 Min-Sum Decoder

The MS decoder can be directly developed from the log-domain BP decoder as follows. From Fig. 12.15 observe that the graph of function $\varphi(x)$ is very steep for small values of x. Then, when x assumes small values, a small perturbation in terms of x determines a large deviation in terms of $\varphi(x)$. For this reason, if at least one of the magnitudes $|R_k^j|$ in the summation appearing in (12.25) is sufficiently small, the corresponding value of $\varphi(|R_k^j|)$ dominates the other summands. Hence, we can write

$$
\begin{aligned}
M_j^i &= \prod_{k\in N(j)\setminus\{i\}} \operatorname{sgn}\left(R_k^j\right)\cdot\varphi\left(\sum_{k\in N(j)\setminus\{i\}}\varphi(|R_k^j|)\right)\\
&\approx \prod_{k\in N(j)\setminus\{i\}} \operatorname{sgn}\left(R_k^j\right)\cdot\varphi\left(\max_{k\in N(j)\setminus\{i\}}\varphi(|R_k^j|)\right)\\
&= \prod_{k\in N(j)\setminus\{i\}} \operatorname{sgn}\left(R_k^j\right)\cdot\min_{k\in N(j)\setminus\{i\}}|R_k^j|
\end{aligned}
\tag{12.28}
$$

where the last equality follows from $\varphi(x)$ being self-invertible (i.e., $\varphi(\varphi(x))=x$) and monotonically decreasing. The MS decoding algorithm is summarized next.

Min-Sum Decoding of LDPC Codes

1: set $I=1$. For $i=0,\dots,n-1$, for $j\in N(i)$, set $R_i^j=W_i$;
2: for $j=0,\dots,m-1$
  for $i\in N(j)$ calculate M_j^i according to (11.28);
3: for $i=0,\dots,n-1$
  for $j\in N(i)$ calculate R_i^j according to (11.26);
4: for $i=0,\dots,n-1$ {
  calculate $\log L(c_i|W_i,\boldsymbol{M}^i)$ according to (11.27);
  if $\log L(c_i|W_i,\boldsymbol{M}^i)\geq 0$ then set $\hat{c}_i=0$;
  else set $\hat{c}_i=1$;
}
if $\hat{\boldsymbol{c}}\boldsymbol{H}^T=\boldsymbol{0}$ then return $\hat{\boldsymbol{c}}$;
else {
  if $I=I_{max}$ exit;
  else {
    $I=I+1$;
    goto 2;
    }
}

Several improvements to the MS decoder have been proposed in the literature, to reduce the gap between its performance and that of BP decoding, at the expense of a small increase in terms of computational cost. These refinements are out of the scope of this book. Interested readers may refer, for example, to [31, 32].

12.6.2 Gallager B Decoder

The BP and MS decoders are characterized by real-valued (properly quantized, in hardware implementation) messages exchanged between the variable nodes and the check nodes. Moreover, as previously emphasized, both algorithms remain unchanged over a wide range of communication channels. In contrast, Gallager B decoder, first proposed in [7], is a message-passing decoding algorithm for LDPC codes characterized by binary-valued messages and is specifically tailored for the BSC (i.e., no soft information is available at the decoder input). Although its performance is poor compared with that of BP and MS algorithms over the BSC, it has been proved that it represents the optimum LDPC decoder over the BSC when the extrinsic messages are constrained to be binary.

The algorithm works as follows. Assuming transmission over a BSC with error probability p and input and output alphabets $\mathcal{X}=\mathcal{Y}=\{0,1\}$, for $i=0,\ldots,n-1$ variable node V_i is fed with the corresponding binary symbol $y_i\in\{0,1\}$ received from the channel. (In contrast, to perform BP decoding over the BSC variable node i is initialized according to (12.18) or to its logarithmic version (12.23).) The symbol y_i is broadcasted by variable node V_i to each of its neighboring check nodes. The algorithm is then structured in a similar way as BP or MS, where the horizontal, vertical, and stopping criterion steps are specified as follows.

During the horizontal step, for $j=0,\ldots,m-1$ the message propagating from check node C_j to variable node V_i, $i\in N(j)$, is simply the modulo-2 summation of all binary messages incoming from variable nodes connected to C_j but the message incoming from V_i. Hence, we can write

$$m_j^i=\sum_{k\in N(j)\setminus\{i\}} r_k^j \tag{12.29}$$

where the summation is modulo-2. (Note that $r_i^j=y_i$ for all $i=0,\ldots,n-1$ at the first iteration.) During the vertical step, for $i=0,\ldots,n-1$ the message from variable node V_i to check node C_j, $j\in N(i)$, is equal to the modulo-2 complement of y_i if the number of incoming extrinsic messages different from y_i is above some threshold, and is equal to y_i otherwise. Letting

$$X_j^i=\left|\left\{m_k^i\neq y_i\, s.t.\, k\in N(i)\setminus\{j\}\right\}\right|$$

and $T^{(i)}$ be the number of such extrinsic messages and the threshold at the current iteration, respectively, and letting $C(y_i)$ be the modulo-2 complement of y_i, we have

$$r_i^j=\begin{cases} C(y_i) & \text{if } X_j^i\geq T^{(i)} \\ y_i & \text{otherwise.}\end{cases} \tag{12.30}$$

At the end of each decoding iteration, for each variable node V_i the decision about the current value of the local bit $\hat{c}_i$ is taken according to a majority policy.

More specifically, if the variable node degree d_i is even, then $\hat{c}_i$ is set equal to the value assumed by the majority of the incoming messages m_j^i and of y_i. On the other hand, if the variable node degree is odd, then $\hat{c}_i$ is set equal to the value assumed by the majority of the incoming messages m_j^i (y_i is not considered).

Gallager B Decoding of LDPC Codes

1: set $I = 1$. For $i = 0, \ldots, n-1$, for $j \in N(i)$, set $r_i^j = y_i$;
2: for $j = 0, \ldots, m-1$
  for $i \in N(j)$ calculate m_j^i according to (11.29);
3: for $i = 0, \ldots, n-1$
  for $j \in N(i)$ calculate r_i^j according to (11.30);
4: for $i = 0, \ldots, n-1$ {
  if $d_i \bmod 2 = 0$ then set $\hat{c}_i$ to the value assumed by the majority of the incoming messages m_j^i and of y_i ;
  else set $\hat{c}_i$ to the value assumed by the majority of the incoming messages m_j^i ;
}
if $\hat{\boldsymbol{c}}\boldsymbol{H}^T = \boldsymbol{0}$ then return $\hat{\boldsymbol{c}}$;
else {
  if $I = I_{max}$ exit;
  else {
    $I = I + 1$;
    goto 2;
    }
}

Appropriate values for the threshold $T^{(i)}$ range between $\lfloor (d_i - 1)/2 \rfloor$ and d_i, as the number of incoming extrinsic messages enforcing an outgoing message different from $\tilde{y}_i$ must be sufficiently high. Note that in principle, for irregular codes the value of the threshold may be different for two different variable nodes, even during the same iteration. Also note that, for the same variable node, the value of the threshold may not remain constant with the iteration index, as it may be adjusted dynamically. In [7] it was shown that for a regular (d, h) LDPC code, the optimum value of the threshold (the same for all variable nodes at the same iteration) is the smallest integer T for which the inequality

$$\frac{1-p}{p} \leq \left(\frac{1 + (1-2\varepsilon)^{h-1}}{1 - (1-2\varepsilon)^{h-1}} \right)^{2T-d+1} \tag{12.31}$$

is fulfilled, where p is the BSC error probability and ε is the extrinsic error probability. This latter parameter represents the average probability that an edge in the Tanner graph carries an error message from the variable node set to the check node set at the considered iteration, and varies over iterations. In the asymptotic setting where the Tanner graph is assumed to be cycle-free, the update equation for ε for regular LDPC codes is [7]

$$\begin{aligned}\varepsilon_{\ell+1} = p - p\sum_{z=T_\ell}^{d-1}\binom{d-1}{z}\left[\frac{1+(1-2\varepsilon_\ell)^{h-1}}{2}\right]^z\left[\frac{1-(1-2\varepsilon_\ell)^{h-1}}{2}\right]^{d-1-z} \\ + (1-p)\sum_{z=T_\ell}^{d-1}\binom{d-1}{z}\left[\frac{1-(1-2\varepsilon_\ell)^{h-1}}{2}\right]^z\left[\frac{1+(1-2\varepsilon_\ell)^{h-1}}{2}\right]^{d-1-z}\end{aligned} \tag{12.32}$$

where $\ell \geq 0$ is the iteration index and where $\varepsilon_0 = p$.

Example 12.5 Equation (12.32) represents density evolution recursion for Gallager B decoding of regular unstructured (d, h) LDPC code ensembles. The asymptotic decoding threshold p^* for this ensemble under Gallager B decoding is then the sup of the set of all $p > 0$ such that $\lim_{\ell\to\infty}\varepsilon_\ell = 0$. For given d and h, whether or not some p is above or below threshold can be easily checked by running the recursion (with starting point $\varepsilon_0 = p$), adapting the value of T_ℓ at each iteration according to (12.31) for the current value of ε_ℓ. For example, for $d = 4$ and $h = 40$ (which corresponds to a rate $R = 9/10$ ensemble) we obtain a threshold $p^* = 0.0041$. Through (12.8) and $E_s = RE_b$, this corresponds to a threshold $(E_b/N_o)^* = 5.892$ dB, about 1.5 dB away from the Shannon limit relevant to the one-bit quantized Bi-AWGN channel.

12.6.3 Flipping Algorithms

Flipping algorithms are a class of low-complexity, iterative decoding algorithms for LDPC codes over the BSC different from message-passing ones. The decoding strategy consists of flipping, at the end of each decoding iteration, the current value of a subset of variable nodes for which a certain flipping condition is fulfilled. If the obtained binary sequence is a codeword, decoding is stopped and the codeword is returned. Otherwise, a new iteration is started. The process continues until a codeword is found or a maximum number of iterations is reached. Different flipping algorithms are characterized by different criteria to identify the variable nodes to be flipped.

A popular flipping algorithm, hereafter referred to simply as bit-flipping (BF) algorithm, consists of flipping at each iteration those variable nodes for which

the number u of unsatisfied check nodes is maximum. A BSC with input and output alphabets $\mathcal{X} = \mathcal{Y} = \{0, 1\}$ is assumed.

Bit-Flipping Decoding of LDPC Codes

1: set $I = 1$. For $i = \{0, \ldots, n-1\}$, set $\hat{c}_i = y_i$;
2: if $\hat{\boldsymbol{c}}\boldsymbol{H}^T = 0$ then return $\hat{\boldsymbol{c}}$;
3: for $i = 0, \ldots, n-1$
    calculate u_i ;
4: calculate u_{max} ;
5: for each i such that $u_i = u_{max}$ set $\hat{c}_i = \left(\hat{c}_i + 1\right) mod 2$;
5: if $\hat{\boldsymbol{c}}\boldsymbol{H}^T = 0$ then return $\hat{\boldsymbol{c}}$;
  else {
    if $I = I_{max}$ exit;
    else {
      $I = I + 1$;
      goto 2;
      }
  }

12.7 Non-binary LDPC Codes

The so far introduced LDPC codes are binary, in that the code represents an Rn-dimensional subspace of the vector space $\mathrm{GF}(2)^n$, where R is the code rate, n is the codeword length, and $\mathrm{GF}(2)$ is the Galois field of order 2. More specifically, the LDPC code is the (Rn-dimensional) null space of an $m \times n$ sparse parity-check matrix $\boldsymbol{H}$. All n encoded vectors belong to $\mathrm{GF}(2)$ as well as all of the elements of $\boldsymbol{H}$. If row vector $\boldsymbol{a}$ belongs to $\mathrm{GF}(2)^n$, then the syndrome of $\boldsymbol{a}$ is $s = aH^T \in \mathrm{GF}(2)^m$, where all operations are performed in $\mathrm{GF}(2)$. Vector $\boldsymbol{a}$ is a codeword if and only if its syndrome is null.

Like other classes of linear block codes, also LDPC codes may be constructed on Galois fields of order $q > 2$ [33]. In this case the code can be represented by a sparse parity-check matrix $\boldsymbol{H}$ on $\mathrm{GF}(q)$ i.e., a matrix whose elements $h_{j,i}$, $j \in \{0, 1, \ldots, m-1\}$ and $i \in \{0, 1, \ldots, n-1\}$, belong to $\mathrm{GF}(q)$ and with a relatively small number of nonzero elements. The code is an Rn-dimensional subspace of the vector space $\mathrm{GF}(q)^n$, where R is still the code rate and the codeword length n is expressed in Galois field symbols. Letting row vector $\boldsymbol{a}$ belong to $\mathrm{GF}(q)^n$, the syndrome of $\boldsymbol{a}$ is still $s = aH^T \in \mathrm{GF}(q)^m$, where now all operations are performed in

GF(q). Still, $\boldsymbol{a}$ is a codeword if and only if its syndrome is null. Hereafter we focus on LDPC codes constructed on extension fields GF(q) with $q = 2^p$ for integer $p > 2$. We denote by α a primitive element of GF(q). We use the terminology non-binary LDPC (NB-LDPC) code to refer to an LDPC code constructed on the Galois field GF(q).

12.7.1 NB-LDPC Code Ensembles

As binary LDPC codes, also NB-LDPC ones admit a graphical representation through a Tanner graph $G = (\mathcal{V} \cup \mathcal{C}, \mathcal{E})$. Again, $\mathcal{V} = \{V_0, V_1, \ldots, V_{n-1}\}$ is the set of variable nodes, $\mathcal{C} = \{C_0, C_1, \ldots, C_{m-1}\}$ is the set of check nodes, and $\mathcal{E}$ is the set of edges. The number of edges, equal to the number of non-zero entries of $\boldsymbol{H}$, is still denoted by E. The n variable nodes and the m check nodes are still bijectively associated with the n codeword symbols and with the m parity-check equations, respectively; each encoded symbol now belongs to GF(q) and each parity-check equation is a linear equation in GF(q). In the Tanner graph, variable node $V_i \in \mathcal{V}$ is connected to check node $C_j \in \mathcal{C}$ by an edge if and only if $h_{j,i} \in \text{GF}(q) \backslash \{0\}$, i.e., if and only if the element of $\boldsymbol{H}$ in row $j \in \{0, 1, \ldots, m-1\}$ and column $i \in \{0, 1, \ldots, n-1\}$ is non-zero. Equivalently, V_i is connected to C_j if and only if the non-binary codeword symbol c_i, associated with V_i, is involved in the parity-check equation corresponding to C_j. Edge labeling represents the main difference between the Tanner graphs of binary and non-binary LDPC codes. As opposed to the Tanner graph of a binary LDPC code, in fact, in the Tanner graph of a NB-LDPC code the edge connecting variable node V_i to check node C_j is labeled by the corresponding non-zero element $h_{j,i}$ of the parity-check matrix.

As an example, the Tanner graph of a linear block code with codeword length $n = 5$ and dimension $k = 3$ (where both n and k are measured in field symbols) is shown in Fig. 12.16. The Tanner graph has two check nodes, each imposing a linear constraint on the variable nodes connected to it, and five variable nodes, each

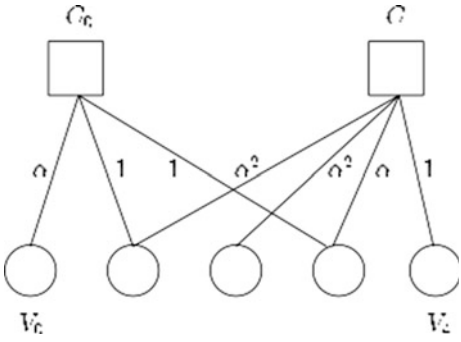

Fig. 12.16 Tanner graph of a non-binary linear block code over GF(4) with codeword length 5 (field symbols) and code rate 3/5. Each edge in the Tanner graph is labeled with a non-zero element of GF(4)

representing a codeword symbol. The parity-check matrix of the corresponding linear block code is

$$H = \begin{bmatrix} \alpha & 1 & 0 & 1 & 0 \\ 0 & \alpha^2 & \alpha^2 & \alpha & 1 \end{bmatrix}.$$

Similarly to their binary counterparts, NB-LDPC codes are usually analyzed in terms of ensemble average. Ensembles of NB-LDPC codes are defined similarly to ensembles of binary LDPC codes, with the difference that edge labeling is also considered in the ensemble definition. For example, the unstructured ensemble of NB-LDPC codes over GF(q) of length n and degree distribution (Λ, P), denoted by $\mathcal{C}_q(n, \Lambda, \mathrm{P})$, includes the LDPC codes constructed GF(q) corresponding to all possible $E!$ edge permutations between the variable node and the check node sockets, according to a uniform probability distribution and, for each such permutation, all possible edges labelings with non-zero elements of GF(q) again according to a uniform probability measure. Ensembles of ultra-sparse NB-LDPC codes (where all variable nodes have degree 2) have attracted an increasing interest in the past decade [34, 35]. Ensembles of protograph-based NB-LDPC codes may also be defined similarly to their binary counterparts, by including edge labeling in the ensemble definition [36, 37].

12.7.2 Iterative Decoding of NB-LDPC Codes

Similarly to binary LDPC codes, NB-LDPC codes may be decoded iteratively via BP decoding. The BP decoder for NB-LDPC codes may be regarded as a generalization of the above-described BP decoder for binary LDPC codes. Hereafter, we provide a description of such a decoder, focusing on its probability-domain implementation. We assume an extension field of order $q = 2^p$ for integer $p > 2$. For the sake of clarity, we divide the algorithm into six steps called initialization, message permutation, horizontal step, message de-permutation, vertical step and hard decision and stopping criterion. Out of these six steps, the first one (initialization) is performed only once, at the beginning of the algorithm, while the others are performed iteratively until a stopping rule is verified.

In the non-binary BP decoder, each message still represents extrinsic information. As opposed to the binary case, in which each message exchanged between a variable node V_i and a check node C_j is (in the log-domain implementation) a scalar value representing a likelihood ratio or log-likelihood ratio, in the non-binary setting each message is a vector of length $q = 2^p$ representing a pmf for the non-binary symbol associated with V_i. For example, for an LDPC code constructed over the Galois field GF(4), the message m_j^i from check node C_j to variable node V_i is a vector with four elements having the form $m_j^i = \left(m_j^i(0), m_j^i(1), m_j^i(\alpha), m_j^i(\alpha^2)\right)$

where $m_j^i(0) = \Pr(V_i = 0)$, $m_j^i(1) = \Pr\{V_i = 1\}$, $m_j^i(\alpha) = \Pr\{V_i = \alpha\}$, and $m_j^i(\alpha^2) = \Pr\{V_i = \alpha^2\}$, each probability being conditioned to the extrinsic information received by the check node along all of its edges but the one towards V_i.

12.7.2.1 Initialization

In the initialization step, each variable node receives a priori information from the channel and simply broadcasts it along all of its edges, towards the check nodes that are connected to it. Hereafter we denote by r^i a priori information for variable node V_i. As well as messages exchanged between variable nodes and check nodes, r_i is a pmf for the non-binary symbol $c_i \in \mathrm{GF}(2^p)$ associated with V_i. The way a priori information r_i is computed depends on the channel.

For example, let us consider transmission of a NB-LDPC code constructed on $\mathrm{GF}(2^p)$ over the Bi-AWGN channel depicted in Fig. 12.16. Let $c = (c_0, c_1, \ldots, c_{n-1})$ be the NB-LDPC codeword, $c_i \in \mathrm{GF}(2^p)$ for $i \in \{0, 1, \ldots, n-1\}$. In this case the generic non-binary codeword symbol c_i is first converted to its binary representation $c_i = (c_{i,0}, c_{i,1}, \ldots, c_{i,p-1})$, $c_{t,j} \in \mathrm{GF}(2)$ for $j \in \{0, 1, \ldots, p-1\}$. Then, the binary representation is mapped onto a word of p antipodal symbols $x_i = 1 - 2c_i$ (meaning E_s normalized to 1), yielding a sequence $x = (x_0, x_1, \ldots, x_{n-1})$ of np channel symbols that are transmitted sequentially over the channel. Letting $y = (y_0, y_1, \ldots, y_{n-1})$ be the corresponding Bi-AWGN channel output, it is easy to verify that, for all $i \in \{0, 1, \ldots, n-1\}$ and for each $\beta \in \mathrm{GF}(2^p)$, we have

$$\Pr(c_i = \beta | y_i) \, \alpha \, \left(2\pi\sigma^2\right)^{-\frac{p}{2}} \exp\left(- \frac{||\mathbf{x_i}(\beta)||^2 + ||\mathbf{y_i}||^2}{2\sigma^2} \right) \exp\left(\frac{< \mathbf{x_i}(\beta), \mathbf{y_i} >}{\sigma^2} \right) \quad (12.33)$$

where $x_i(\beta)$ is the antipodal version of the binary representation of β, where σ^2 is the variance of each noise sample, and where $\langle x_i(\beta), y_i \rangle$ is the inner product between $x_i(\beta)$ and y_i. In this example, a priori information for V_i (coinciding with the message V_i sends to all of its neighboring check nodes during the initialization step) is therefore $r_i = \left(\Pr\{c_i = 0\}, \Pr\{c_i = 1\}, \Pr\{c_i = \alpha\}, \ldots, \Pr\left\{c_i = \alpha^{q-2}\right\}\right)$ where each element of the pmf r^i is computed according to (12.33). Over an SLC or MLC channel model, a priori information shall be appropriately computed, usually based again on the binary representation of each non-binary codeword symbol.

All subsequent steps of the BP decoder for NB-LDPC codes, described next, remain the same regardless of the specific channel model and therefore irrespective of how a priori information is computed.

12.7.2.2 Message Permutation

As previously described, each edge in the Tanner graph of a NB-LDPC code is labeled by the corresponding non-zero element of the parity-check matrix. Considering check node C_j and letting $c_i \in \mathrm{GF}(2^p)$ be the jth codeword symbol, the check node imposes the constraint

$$\sum_{k \in N(j)} h_{j,k} c_k = 0. \tag{12.34}$$

where $h_{j,k} \in \mathrm{GF}(2^p) \backslash \{0\}$ is the element of $\boldsymbol{H}$ in position (j,k). This means that the value of each variable node V_i, connected to C_j, is first multiplied by the corresponding edge label and then is checked by the check node through (12.34). In terms of BP decoding, where message r_i^j is a pmf for symbol $c_i \in \mathrm{GF}(2^p)$, multiplication of by the non-zero edge label simply entails a permutation of the elements of r_i^j. To make a distinction between the message sent by V_i and the message received by C_j (after the permutation), hereafter we denote the former by r_i^j and the latter by $\Pi(r_i^j)$. An example is provided next.

Example 12.6 Let the Galois field order be $q = 4$. Let the edge connecting variable node V_i and check node C_j be labeled by $\alpha \in \mathrm{GF}(4)$, and the message sent by V_i be $r_i^j = (0.4, 0.3, 0.2, 0.1)$. Since in GF(4) we have $0 \cdot \alpha = 0$, $1 \cdot \alpha = \alpha$, $\alpha \cdot \alpha = \alpha^2$, and $\alpha^2 \cdot \alpha = 1$, the effect of the edge label α on the message is a permutation of its elements, leading to the message $\Pi\left(r_i^j\right) = (0.4, 0.1, 0.3, 0.2)$ received by C_j. Each non-zero edge label induces a specific permutation.

12.7.2.3 Horizontal Step

Check node C_j, $j \in \{0, 1, \ldots, m-1\}$, receives one message $\Pi(r_i^j)$, $i \in N(j)$, from each of its neighboring variable nodes and sends back one message m_j^i, $i \in N(j)$, to each of them. To understand how extrinsic information shall be generated at the check node and forwarded to the relevant variable node, we can look at (12.34) that we recast in the form $\sum_{k \in N(j)} z_k = 0$ by defining $z_k = h_{j,k} c_k$. For some $i \in N(j)$, the constraint imposed by the check node is $z_i = -\sum_{k \in N(j) \backslash \{i\}} z_k = \sum_{k \in N(j) \backslash \{i\}} z_k$ where the "$-$" sign can be omitted owing to the fact that $q = 2^p$. We may regard each summand z_k as a random variable taking values in GF(q) and with pmf equal to that of the incoming message $\Pi(r_k^j)$. Under the assumption that all z_k are independent, the pmf of their sum (hence the pmf of z_i) is the convolution of their pmfs. That is, we may write

$$m_j^i = \circledast_{k \in N(j) \setminus \{i\}} \Pi\left(r_k^j\right) \tag{12.35}$$

where $\circledast$ denotes convolution between pmfs.

Since the complexity of convolution scales quadratically with the vector size, a naïve implementation of the horizontal step based on (12.35) leads to a complexity scaling as $\mathcal{O}(q^2)$, such a complexity dominating the overall decoding complexity and becoming problematic even for moderate q. A reduced-complexity but equivalent implementation of the horizontal step is based on applying fast Hadamard transform to both sides of (12.35). Hadamard transform turns vector convolution into element-wise multiplication of the transformed vectors; therefore, letting $\mathcal{H}$ denote the Hadamard transform and recalling that Hadamard transform coincides with its inverse, m_j^i in (12.35) may equivalently be calculated as

$$m_j^i = \mathcal{H}\left(\otimes_{k \in N(j) \setminus \{i\}} \mathcal{H}\left(\Pi\left(r_k^j\right)\right)\right) \tag{12.36}$$

where $\otimes$ denotes element-wise product between two vectors. Using fast Hadamard transform reduces the horizontal step complexity (and more in general the complexity of the whole decoder) to $\mathcal{O}(q \log q)$.

12.7.2.4 Message De-permutation

In the previously described message permutation step, the elements of message r_i^j, sent by variable node V_i to check node C_j, are permuted according to the permutation established by the edge label $h_{j,i}$. The message m_j^i, sent by C_j towards V_i, must undergo the inverse permutation (equivalently, the permutation established by the inverse label $h_{j,k}^{-1}$) before reaching V_i. For the sake of clarity, in order to distinguish the message sent by C_j from the message received by V_i after de-permutation, we keep denoting by m_j^i the former and by $\Pi^{-1}\left(m_j^i\right)$ the latter.

12.7.2.5 Vertical Step

Variable node V_i, $i \in \{0, 1, \ldots, n-1\}$, receives the d_i messages $\Pi^{-1}\left(m_j^i\right)$, $j \in N(i)$, and generates d_i messages r_i^j, $j \in N(i)$, each of which is sent towards a specific edge to the corresponding check node. Each message r_i^j is computed based on a priori information r_i available from the channel and on extrinsic information $\Pi^{-1}\left(m_k^i\right)$, $k \in N(i) \setminus \{j\}$. Specifically, assuming independence between all of the incoming messages (including a priori information), r_i^j is computed as

$$r_i^j = \gamma_j \cdot r_i \otimes \left(\otimes_{k \in N(i) \setminus \{j\}} \Pi^{-1}\left(m_k^i\right) \right) \tag{12.37}$$

where again $\otimes$ denotes element-wise product between two vectors and where the scalar γ_j is a scaling factor whose value makes the sum of the elements of r_i^j equal to 1.

12.7.2.6 Hard Decision and Stopping Criterion

At the end of each BP decoding iteration, a hard decision is made about the value taken by each variable node; this hard decision exploits a posteriori information for the variable node. In probability-domain BP decoding of NB-LDPC codes, a posteriori information is represented by the pmf of the Galois field symbol c_i associated with the variable node given all incoming messages and a priori information. Under independence assumption this is given by

$$r_i^{\mathrm{APP}} = \gamma \cdot r_i \otimes \left[\otimes_{j \in N(i)} \left(\Pi^{-1}\left(m_j^i\right) \right) \right] \tag{12.38}$$

where again γ is a normalization factor. Let $\mathbf{\Pi}^{-1}(m^i)$ be the ordered list of messages received by variable node V_i. Once the a posteriori pmf r_i^{APP} for symbol c_i has been computed, a symbol-wise MAP decision is made, namely,

$$\hat{c}_i = \mathrm{argmax}_{c \in \mathrm{GF}(q)} \Pr\left\{ c | r^i, \mathbf{\Pi}^{-1}(m^i) \right\}. \tag{12.39}$$

In other words, $\hat{c}_i$ is the element of $\mathrm{GF}(q)$ that corresponds to the largest element of the pmf r_i^{APP}. Note that, as the scaling factor γ is the same for all elements of r_i^{APP}, it does not affect the final decision and therefore it can be set to 1 for all $i \in \{0, 1, \ldots, n-1\}$.

Similarly to the binary case, if the current hard-decision sequence $\hat{c} = (\hat{c}_0, \hat{c}_1, \ldots, \hat{c}_{n-1})$ fulfills $\hat{c}H^T = \mathbf{0}$, then the algorithm terminates and $\hat{c}$ is returned as the detected codeword. Else, if $\hat{c}$ is not a codeword and the maximum number of iterations I_{max} has been reached, the algorithm is terminated and a failure is reported. Else, a new iteration is started jumping to the message permutation step.

Belief propagation decoding of LDPC codes over $\mathrm{GF}(q)$, $q = 2^p$, may be summarized as follows.

Belief-Propagation Decoding of NB-LDPC Codes over GF(2^p)

1: set $I = 1$. For $i = 0, \ldots, n-1$, for $j \in N(i)$, set $r_i^j = r_i$;
2: for $i = 0, \ldots, n-1$
    for $j \in N(i)$ permute the elements of r_i^j based on $h_{j,i}$;
3: for $j = 0, \ldots, m-1$
    for $i \in N(j)$ calculate m_j^i according to (11.36);
4: for $i = 0, \ldots, n-1$
    for $j \in N(i)$ permute the elements of m_j^i based on $h_{j,i}^{-1}$;
5: for $i = 0, \ldots, n-1$
    for $j \in N(i)$ calculate r_i^j according to (11.37);
6: for $i = 0, \ldots, n-1$ {
    calculate r_i^{APP} according to (11.38);
    set the value of $\hat{c}_i$ according to (11.39);
}
if $\hat{\boldsymbol{c}}\boldsymbol{H}^T = \boldsymbol{0}$ then return $\hat{\boldsymbol{c}}$;
else {
    if $I = I_{max}$ exit;
    else {
        $I = I + 1$;
        goto 2;
        }
}

12.8 Numerical Example

In this section, we present some numerical results aimed at comparing the performance of binary LDPC and BCH codes, with the purpose to highlight the potential of LDPC codes in Flash memories applications. We assume an SLC memory as the reference channel model. We compare the performance of a regular QC-LDPC code, under several decoding algorithms offering different tradeoffs between performance and complexity, with the performance of a narrowsense binary BCH code with similar parameters, decoded via bounded distance decoding.

The LDPC code is characterized by a length $n_{\mathrm{LDPC}} = 8200$ and a dimension $k_{\mathrm{LDPC}} = 7379$ bits, and therefore by a code rate R very close to $9/10$. Its minimum distance, estimated with the impulse method proposed in [38], is equal to $d_{\mathrm{LDPC}} = 114$. All variable nodes of the LDPC code have degree 4, and all of its check nodes have degree 40. Its 820×8200 parity-check matrix is in block

circulant form, where the generic block is a 205×205 circulant permutation matrix, and has been constructed according to a block circulant version of the progressive edge-growth (PEG) algorithm. The performance of this code has been evaluated via Monte Carlo software simulation, under BP, MS, and BF decoding algorithms. The performance curves under both BP and MS decoding have been obtained under two different settings, namely, soft-decision and hard-decision decoding. These two settings correspond to assuming the Bi-AWGN channel with unquantized output (Example 12.2) and with one-bit quantized output (Example 12.3), respectively, as the channel model. The first setting is equivalent to assuming an SLC memory with an infinite number of reads per bit, while the second one to assuming an SLC memory with one read per bit. The variable nodes are initialized according to (12.19) in the unquantized case and according to (12.18) in the quantized one. In the quantized case, the raw bit error rate of the channel can be obtained from E_b/N_0 according to (12.8), where $E_s/N_0 = RE_b/N_0$. For instance, $E_b/N_0 = 5$ dB corresponds to a raw bit error rate $p = 8.5 \cdot 10^{-3}$. The Shannon limit for the unquantized case and for the one-bit quantized case are also evaluated, for benchmarking purposes.

The competitor BCH code has nominal parameters $n_{\text{BCH}} = 8191$, $k_{\text{BCH}} = 7372$, $t = 63$ (error correction capability), and minimum distance $d_{\text{BCH}} = 127$. Its code rate is approximately equal to 9/10, similar to the code rate of the QC-LDPC code. The codeword error rate (CER) and the bit error rate (BER) of the BCH code under hard decision bounded distance decoding have been evaluated analytically according to the relationships

$$P_e = \sum_{r=t+1}^{n_{\text{BCH}}} \binom{n_{\text{BCH}}}{r} p^r (1-p)^{n_{\text{BCH}}-r} \tag{12.40}$$

and

$$P_b \approx \frac{d_{\text{BCH}}}{k} \cdot P_e \tag{12.41}$$

respectively.

With reference to Fig. 12.17, we see that over the hard-decision channel (SLC with one read) the BCH code exhibits nearly the same performance as the QC-LDPC code decoded via BP and that its performance is even slightly better at low error rates. This is not surprising, as BCH codes are well known to offer very good performances over hard-decision channels, especially at high code rates. As opposed to BCH codes, however, LDPC codes can handle in a very natural way soft information incoming from the communication channel, which allows to attain substantial performance improvements over the error correction capabilities achievable with hard-decision decoding. In our example, when the LDPC decoder is fed with unquantized soft information, its coding gain with respect to that achieved under hard-decision decoding is improved by about 1.6 dB under both BP

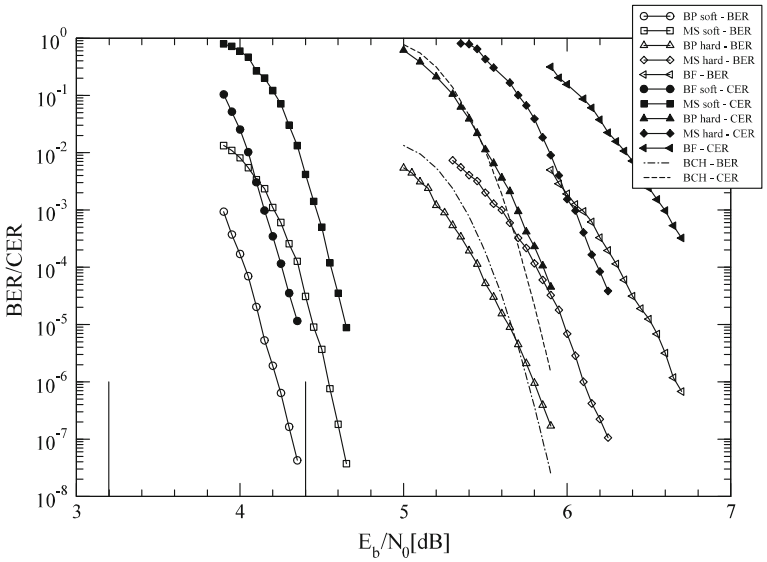

Fig. 12.17 Bit and codeword error rates for an (8191, 7372) QC-LDPC code (under different decoding algorithms) and an (8191, 7372), t = 63 narrowsense binary BCH code under bounded distance decoding, over an SLC flash memory channel. Curves corresponding to filled and empty symbols illustrate the codeword error rates and the bit error rates of the LDPC code, respectively. The dashed and dot-dashed lines illustrate the codeword error rate and the bit error rate of the BCH code, respectively. The two straight solid lines are the Shannon limits for rate R = 9/10 under soft-decision and hard-decision decoding, respectively

and MS decoding algorithms at CER $= 10^{-4}$. Moreover, again at CER $= 10^{-4}$, the LDPC code under unquantized BP decoding performs only 0.8 dB away from the corresponding Shannon limit, in terms of BER.

For the same decoding algorithm (BP or MS), the performance curves of the LDPC code labeled as "soft" and "hard" represent the two extreme cases in which unconstrained soft information is available at the decoder, and no soft information is available. In general, when a finite number of cell reads is performed with different read voltage values, the corresponding performance curve will lie between the two extreme curves: The larger the number of cell reads, the closer the performance curve to the "soft" one. Therefore, LDPC codes can largely outperform BCH codes in Flash memory applications, provided a sufficient amount of soft information is available at the decoder. It is also pointed out that the design of appropriate QC irregular LDPC codes can favor an even larger coding gain with respect to BCH codes.

We also highlight how very simple decoding algorithms of LDPC codes such as BF (or Gallager B) decoding, can be of interest at the beginning of the memory life, i.e., when the raw bit error rate is very small. For example, as from Fig. 12.17, BF decoding could become of interest for values of E_b/N_0 larger of 7.0 dB, corresponding to a raw bit error rate smaller than $1.3 \cdot 10^{-3}$.

Acknowledgements The author wishes to thank R. Micheloni and A. Marelli for their careful proofcheck of this chapter.

References

1. T.M. Cover, J.A. Thomas, *Elements of Information Theory* (Wiley, 1991)
2. R.D. Fowler, L. Nordheim, Electron emission in intense electric fields. Proc. R. S. Lond. **119**, 173–181 (1928)
3. R. Micheloni, L. Crippa, A. Marelli (eds.), *Inside NAND Flash Memories* (Springer, 2010)
4. N. Mielke et al., Bit error rate in NAND Flash memories, in *Proceedings of the 2008 IEEE International Symposium on Reliability Physics*, Phoenix, AZ, USA, April/May 2008, pp. 9–19
5. J. Wang, T. Courtade, H. Shankar, R. Wesel, Soft information for LDPC decoding in flash: mutual-information optimized quantization, in *Proceedings of the 2011 IEEE Global Telecommunication Conference*, Houston, TX, USA, Dec 2011
6. S. Li, T. Zhang, Improving multi-level NAND flash memory storage reliability using concatenated BCH-TCM coding. IEEE Trans. VLSI **18**, 1412–1420 (2010)
7. R.G. Gallager, *Low-Density Parity-Check Codes* (MIT Press, Cambridge, Massachusetts, 1963)
8. C. Berrou, A. Glavieux, P. Thitimajshima, Near Shannon limit error-correcting coding and decoding: turbo-codes, in *Proceedings of the 2003 International Symposium on Communication*, vol. 2, May 1993, pp. 1064–1070
9. T. Richardson, R. Urbanke, The renaissance of Gallager's low-density parity-check codes. IEEE Commun. Mag. **41**, 126–131 (2003)
10. N. Bonello, S. Chen, L. Hanzo, Low-density parity-check codes and their rateless relatives. IEEE Commun. Surv. Tutor. **13**, 3–26 (2011)
11. M. Tanner, A recursive approach to low complexity codes. IEEE Trans. Inf. Theory **27**, 533–547 (1981)
12. M. Fossorier, Quasi-cyclic low-density parity-check codes from circulant permutation matrices. IEEE Trans. Inf. Theory **50**, 1788–1793 (2004)
13. Z. Li, L. Chen, L. Zeng, S. Lin, W. Fong, Efficient encoding of low-density parity-check codes. IEEE Trans. Commun. **54**, 71–81 (2006)
14. M. Mansour, High-performance decoders for regular and irregular repeat-accumulate codes, in *Proceedings of the IEEE 2004 IEEE Global Telecommunications Conference*, Nov/Dec 2004, pp. 2583–2588
15. T. Richardson, M. Shokrollahi, R. Urbanke, Design of capacity-approaching irregular low-density parity-check codes. IEEE Trans. Inf. Theory **47**, 619–637 (2001)
16. S.-Y. Chung, G.D. Forney Jr., T. Richardson, R. Urbanke, On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit. IEEE Commun. Lett. **5**, 58–60 (2001)
17. J. Thorpe, Low-density parity-check (LDPC) codes constructed from protographs, JPL INP, Technical Report, Aug 2003, pp. 42–154
18. J. Xu, L. Chen, L. Zeng, L. Lan, S. Lin, Construction of low-density parity-check codes by superposition. IEEE Trans. Commun. **53**, 243–251 (2005)
19. T. Richardson, R. Urbanke, The capacity of low-density parity-check codes under message-passing decoding. IEEE Trans. Inf. Theory **47**, 599–618 (2001)
20. S. ten Brink, Convergence behavior of iteratively decoded parallel concatenated codes. IEEE Trans. Commun. **49**, 1727–1737 (2001)
21. G. Liva, M. Chiani, Protograph LDPC codes design based on EXIT analysis, in *Proceedings of the 2007 IEEE Global Telecommunications Conference*, Washington, DC, USA, Nov 2007, pp. 3250–3254

22. L. Chen, J. Xu, I. Djurdjevic, S. Lin, Near Shannon limit quasi cyclic low-density parity-check codes. IEEE Trans. Commun. **52**, 1038–1042 (2004)
23. H. Tang, J. Xu, Y. Kou, S. Lin, K. Abdel-Ghaffar, On algebraic construction of Gallager and circulant low density parity-check codes. IEEE Trans. Inf. Theory **50**, 1269–1279 (2004)
24. M. Chiani, A. Ventura, Design and performance evaluation of some high-rate irregular low-density parity-check codes, in *Proceedings of the 2001 Global Telecommunication Conference*, San Antonio, TX, USA, Nov 2001, pp. 990–994
25. T. Richardson, Error floors of LDPC codes, in *Proceedings of the 41st Annual Allerton Conference on Communication, Control and Computing* (2003)
26. S. Abu-Surra, D. Divsalar, W.E. Ryan, Enumerators for protograph-based ensembles of LDPC and generalized LDPC codes. IEEE Trans. Inf. Theory **57**, 858–886 (2011)
27. M. Flanagan, E. Paolini, M. Chiani, M. Fossorier, On the growth rate of the weight distribution of irregular doubly-generalized LDPC codes. IEEE Trans. Inf. Theory **57**, 3721–3737 (2011)
28. D. Cavus, C. Haymes, Low BER performance estimation of LDPC codes via application of importance sampling to trapping sets. IEEE Trans. Commun. **57**, 1886–1888 (2009)
29. L. Dolecek et al., Predicting error floors of structured LDPC codes: Deterministic bounds and estimates. IEEE J. Sel. Areas Commun. **27**, 908–917 (2009)
30. J. Chen, A. Dholakia, E. Eleftheriou, M. Fossorier, X.-Y. Hu, Reduced-complexity decoding of LDPC codes. IEEE Trans. Commun. **53**, 1288–1299 (2005)
31. J. Zhao, F. Zarkeshvari, A. Banihashemi, On implementation of min-sum algorithm and its modifications for decoding low-density parity-check (LDPC) codes. IEEE Trans. Commun. **53**, 549–554 (2005)
32. J. Chen, M. Tanner, C. Jones, Y. Li, Improved min-sum decoding algorithms for irregular LDPC codes, in *Proceedings of the 2005 IEEE International Symposium on Information Theory*, Sept 2005, pp. 449–453
33. M. Davey, D. MacKay, Low-density parity check codes over GF(q). IEEE Commun. Lett. **2** (6), 165–167 (1998)
34. C. Poulliat, M. Fossorier, D. Declercq, Design of regular (2, d_c) -LDPC codes over GF(q) using their binary images. IEEE Trans. Commun. **56**(10), 1626–1635 (2008)
35. G. Liva, E. Paolini, B. Matuz, S. Scalise, M. Chiani, Short turbo codes over high order fields. IEEE Trans. Commun. **61**(6), 2201–2211 (2013)
36. L. Dolecek, D. Divsalar, Y. Sun, B. Amiri, Non-binary protograph-based LDPC codes: enumerators, analysis, and designs. IEEE Trans. Inf. Theory **60**(7), 3913–3941 (2014)
37. E. Paolini, M. Flanagan, Efficient and exact evaluation of the weight spectral shape and typical minimum distance of protograph LDPC Codes. IEEE Commun. Lett. **20**(11), 2141–2144 (2016)
38. X.-Y. Hu, M. Fossorier, E. Eleftheriou, On the computation of the minimum distance of low-density parity-check codes, in *Proceedings of the 2004 International Conference on Communication*, June 2004, pp. 767–771