UNIVERSITY OF SOUTHAMPTON

Generalized Low-Density Parity-Check Codes

by

Fang-Chun Kuo

A thesis submitted in partial fulfillment for the degree of Master of Philosophy

in the

Faculty of Engineering, Science and Mathematics
School of Electronics and Computer Science

August 2006

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Master of Philosophy

by Fang-Chun Kuo

An evolution of Gallager's Low-Density Parity-Check (LDPC) codes was first introduced by Tanner in 1981, namely Generalized Low-Density Parity-Check (GLDPC) codes, and then further developed by Boutros *et al.* as well as by Lentmaier and Zigangirov. It has been shown that Hamming-code based GLDPC codes are asymptotically good in the sense of minimum distance and exhibit an excellent performance over both AWGN and Rayleigh channels. Because of the regular parallel structure of the GLDPC decoder, it is amenable to systolic array based practical integrated circuit (IC) implementations.

This thesis is devoted to the characterization of iterative symbol based hard decision aided decoding algorithm designed for GLDPC codes constructed over $GF(q)$. We proposed a novel symbol-flipping based decoding algorithm, designed for GLDPC codes defined over non-binary Galois fields using RS constituent codes. Seven vote rules were proposed and the suggested optimal voting rule was deemed to be $E = 3$, $V = 0$, and $e = F = 1.5$ where larger values indicates unreliable symbols and smaller values indicates more reliable symbols. It was demonstrated by our simulations that our symbol-flipping decoding algorithm can be successfully used for decoding nonbinary GLDPC codes constructed from RS constituent codes. The simulation results also demonstrated that GLDPC codes defined over $GF(q)$ have the potential of outperforming similar-rate binary constituent codes.

Contents

Abstract i

1 Introduction 1
1.1 Publications Supporting the Thesis 3

List of Figures 1

2 Generalized Low-Density Parity-Check Codes 5
2.1 The Structure of GLDPC Codes 5
2.1.1 Description of the Parity-Check Matrix 5
2.1.1.1 Classic LDPC codes 5
2.1.1.2 GLDPC codes 7
2.1.2 Graphical Concept 9
2.1.2.1 Classic LDPC codes 9
2.1.2.2 GLDPC codes 11
2.2 Coding Rate 12
2.3 A Special Case: GLDPC Codes Having J = 2 Levels 13
2.4 GLDPC Encoding 14
2.5 GLDPC Soft-In/Soft-Out Decoding 17
2.5.1 Introduction 17
2.5.2 The Constituent Decoder Using Log-MAP Algorithm 18
2.5.3 Iterative SISO Decoding Algorithm for $J = 2$-Level GLDPC Codes 19
2.6 Decoding Example of the binary $J = 2$-level GLDPC code 22
2.7 Simulation Results 24
2.7.1 Effect of the Number of Iterations 25
2.7.2 Effect of the SISO Decoding Algorithm 33
2.7.3 Effect of Different Coding Rates 37
2.7.4 Effect of Different Codeword Lengths 40
2.7.5 Shortened Constituent Hamming Codes 44
2.8 Conclusions 47

3 Symbol-Flipping Based Decoding of Nonbinary GLDPC Codes 49
3.1 Introduction 49

3.2 Weighted Bit Flip Voting algorithm for Binary Hamming Code-Based GLDPC codes 51
3.3 The Vote Pairs for Symbol-Flipping Based Non-Binary GLDPC Decoding 53
3.4 The Design of the Vote Weight 55
3.5 Symbol-Flipping Based Non-Binary GLDPC Decoding 60
3.6 Simulation Results 61
3.6.1 Effect of the Number of Iterations 61
3.6.2 Effects of the GLDPC Block-length 65
3.6.3 Effects of the RS constituent Code 69
3.6.4 Effect of Different Galois Fields 72
3.7 Conclusion 74

4 Conclusions **76**

Bibliography **78**

Chapter 1

Introduction

The appearance of Shannon's landmark paper [1] published in 1948 marks the beginning of two classic fields, *Information Theory* and *Coding Theory*. Inspired by Hamming [2], coding theorists have developed numerous coding schemes in order to achieve the performance limit predicted by Shannon while maintaining a reasonable complexity. Two main types of codes namely *block codes* and *convolution codes* [3] have been developed. Following Hamming's single error correcting code [2], the family of Bose-Chaudhuri-Hocquenghem (BCH) codes [4, 5, 6] having multiple error correction capability has been developed. Another classic block coding family is represented by that of Reed Solomon (RS) codes [7], which are the nonbinary codes. Another important coding structure was proposed by Forney who introduced the notion of *Concatenated Codes* [8] and following this innovation further diverse alternative solutions have been proposed, such as generalized concatenated codes [9], product codes [10].

It was the work of Berrou *et al.* [11] on *turbo codes* that rekindled the interests of coding theorists. The success of turbo codes is a benefit of the associated iterative decoding algorithm. However, the idea of iterative decoding is closely linked to the concept of the sum-product algorithm often used for Low-Density Parity-Check (LDPC) codes [12, 13]. LDPC codes constitute a class of linear block codes, which were first proposed by Gallager in his 1960 doctoral dissertation [12] and has scarcely been exploited in practice until their rediscovery by Spielman *et al.* [14] and Mackay *et al.* [15, 16]. One notable exception is the important contribution of Tanner in 1981 [17] in which Tanner generalized LDPC codes and introduced a graphical representation of LDPC codes, now often referred to as Tanner graphs.

Following the important contribution of Wiberg, Loeliger and Kotter [18, 19], in recent years graph theory has also been gaining more interest in the coding community [20, 21, 22, 23, 24, 25, 26, 27]. Furthermore, Kschischang has shown in [22] that a single graphical model based framework that is capable of presenting compound codes such as turbo codes [11], serially concatenated codes [28, 29], Gallager's LDPC codes [12] as well as product codes [10] in a identical manner. It has also been demonstrated in [20] that iterative turbo decoding, Pearl's "belief propagation" algorithm [30] and the sum-product algorithm were found to be similar techniques.

Based on the concept of the graphical representation of error control codes, Tanner [17] introduced the concept of Generalized Low-Density Parity-Check (GLDPC) codes, which may be viewed as an evolution of classic LDPC codes [12]. Then Hamming-code based GLDPC codes were further explored by Boutros *et al.* [31, 32] as well as by Lentmaier and Zigangirov [33]. GLDPC codes are constructed by replacing each single parity check of regular LDPC codes with the parity check matrix of a small linear block code referred to as the constituent code. It has been shown that Hamming-code based GLDPC codes are asymptotically good in the sense of minimum distance and exhibit an excellent performance over both AWGN and Rayleigh channels [31, 32, 33]. Pothier [34] also demonstrated that GLDPC codes can be considered as a generalization of product codes and as a benefit of their higher flexibility in terms of the selection of code length, GLPDC codes constitute a promising design alternative to replace product codes in many applications, such as digital audio and TV broadcasting, high speed packet data transmission and deep space applications. Furthermore, the GLDPC decoder of a Hamming-code based scheme has a regular parallel structure, which renders them amenable to systolic array based practical integrated circuit (IC) implementations.

It has been shown in [31] that turbo codes can be described as a particular case of GLDPC codes, where the interleaver acts only on information symbols. Therefore, the iterative SISO decoding algorithm originally applied to turbo codes can be also applied to GLDPC codes. The trellis-based MAP algorithm [35] was used in [31, 32] for decoding GLDPC codes, whereas the Johansson-Zigangirov *A Posteriori* Probability (APP) algorithm [36], which operates with the aid of a single forward recursion walking its way through a syndrome trellis was used in [33]. A range of further sub-optimal decoding algorithms designed for GLDPC codes are exemplified by the Chase algorithm of [37] which was invoked in [38], Kaneko's

algrithm of [39] which was used in [40] and the MAX-Log-MAP algorithm which was employed in [41]. In addition to the above-mentioned soft decoding algorithms, a less complex iterative hard-decision based decoding algorithm, Weighted Bit Flip Voting (WBFV) algorithm, was proposed by Hirst and Honary [42] for decoding binary Hamming constituent code based GLDPC codes.

This thesis is devoted to the characterization of iterative symbol based hard decoding algorithms designed for GLDPC codes constructed over $GF(q)$, since there is a paucity of results on GLDPC codes employing nonbinary constituent codes. By contrast, binary constituent codes have more often been used for constructing GLDPC codes [38, 40, 41, 42, 43, 44]. Moreover, perhaps the best known classic codes are the maximum-minimum-distance nonbinary RS codes, which are used in numerous standards, such as the Digital Audio Broadcast (DAB) and Digital Video Broadcast (DVB) schemes or in Compact Disc (CD) players. It is therefore worth investigating, how RS codes behave, when they are embedded in GLDPC coding schemes. A particular further advantage of GLDPC codes is that their iterative decoding is based on the decoding of modest-complexity constituent codes, hence the total decoding complexity may be expected to be modest.

The outline of the thesis is as follows. In Chapter 2, we introduce the GLDPC codes' construction algorithm and the iterative Soft-In-Soft-Out decoding algorithm with the aid of an example and briefly summarize a range of classic results. A symbol-flipping based decoding algorithm, designed for GLDPC codes defined over non-binary Galois fields was proposed in Chapter 3. The concept of the WBFV decoding algorithm designed for binary Hamming code based GLDPC codes is introduced first, since the symbol-flipping based decoding algorithm was inspired by the WBFV algorithm. It is demonstrated that GLDPC codes defined over $GF(q)$ have the potential of outperforming similar-rate binary constituent codes. Finally, Chapter 4 offers our concluding remarks.

1.1 Publications Supporting the Thesis

- R. Y. S. Tee, F. C. Kuo and L. Hanzo: Multilevel Generalized Low-Density Parity-Check Codes, IEE Electr. Letters, 2nd of February, 2006, Vol. 42, No. 3, pp 167-168

- F-C. Kuo and L. Hanzo: Symbol-Flipping Based Decoding of Generalized Low-Density Parity-Check Codes Constructed over $GF(q)$, Proceedings and CD ROM of IEEE WCNC 2006, 3-6 April, 2006, Las Vegas

- R. Y. S. Tee, F. C. Kuo and L. Hanzo: Generalized Low-Density Parity-Check Coding Aided Multilevel Codes, CD ROM, IEEE VTC 2006 Spring

Chapter 2

Generalized Low-Density Parity-Check Codes

2.1 The Structure of GLDPC Codes

In this section, the structure of GLDPC codes is introduced [34]. There are two appealing ways of describing the structure of GLDPC codes. The first one is based on the construction of the Parity Check Matrix (PCM), while the other is based on the concept of Tanner graphs [17]. Since GLDPC codes may be regarded as an evolution of classic LDPC codes [12], we will give an example of classic LDPC [12] codes first and then extend the concept to GLDPC codes.

2.1.1 Description of the Parity-Check Matrix

2.1.1.1 Classic LDPC codes

Consider the example shown in Fig. 2.1, portraying the PCM H of size 9×12 of a classic $R = K/N$-rate LDPC (N, K) code [12] using the parameters of $N = 12, K = 3$ and $J = 3$, where H is constructed by concatenating $J = 3$ submatrices, namely H^1, H^2 and H^3. The three submatrices H^1, H^2 and H^3 seen in Fig. 2.1 are of dimension 3×12, which are the PCMs of the super-codes[1] C^1, C^2 and C^3, respectively, that are constructed from the Single Parity Check (SPC)

[1]Super-code is proposed in [32], which is constituted by the direct concatenation of $L = N/n$ number of constituent codes.

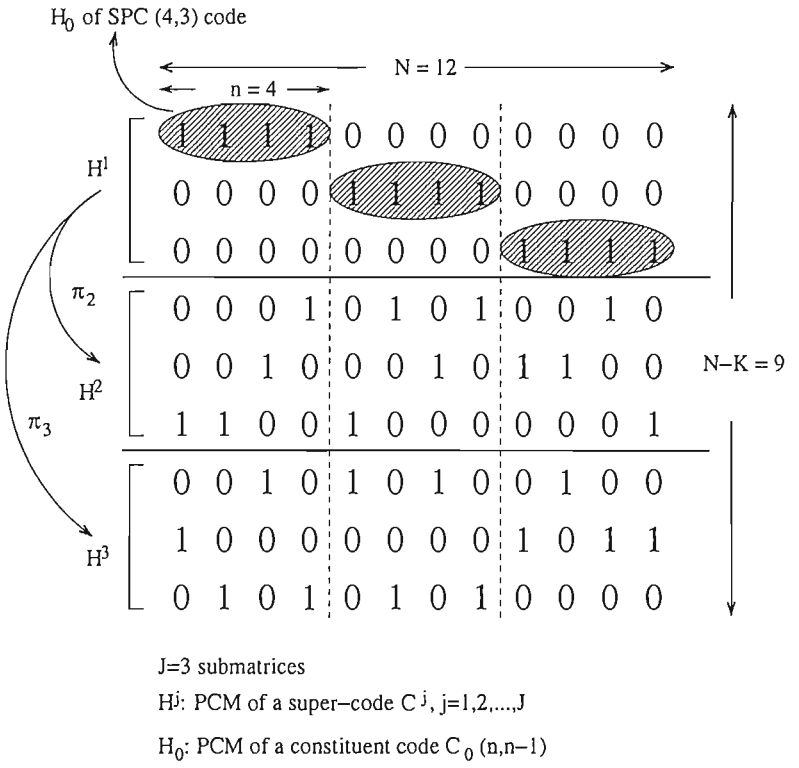


FIGURE 2.1: Parity-check matrix H of an $R = K/N = 1/4$-rate LDPC (12,3) code having $J = 3$ levels, which uses the single parity check code SPC(4,3) as its constituent code.

codes $(n, n-1)$. More explicitly, the first Parity-Check (PC) submatrix H^1 seen in Fig. 2.1 is a *block diagonal matrix* having the matrix elements H_0 along the main diagonal of H^1, which constitute the PCMs H_0 of the SPC$(n, n-1)$ codes associated with $n = 4$. Accordingly, each group of $n = 4$ bits constituting a faction of a codeword of the super-code C^1 is only related to single SPC$(4, 3)$ code rather than to several. Therefore, the super-code C^1 is constituted by the direct concatenation of $L = N/n = 3$ number of SPC$(4, 3)$ codes, which are hence referred to as the constituent codes C_0 seen in the top third of Figure 2.1. Note that in order to distinguish the super-codes and constituent codes, we use superscript and subscript for super-codes and constituent codes, respectively, e.g. H^1 is the PCM of the super-code C^1 and H_0 is the PCM of the constituent code C_0. All the other Parity-Check submatrices, namely H^2 and H^3 are formed by the pseudo-random permutation of all the columns of the submatrix H^1 without interleaving the elements of the columns. This operation is formulated as $H^j = \pi_j(H^1)$ for $j = 2, 3$, explicitly indicating that the super-codes C^2, C^3 are constructed by random interleaving the super-code C^1 as shown in Figure 2.2. Let us explain Figures 2.1 and 2.2 further. As depicted in Figure 2.1, the 1^{st} and 2^{nd} columns

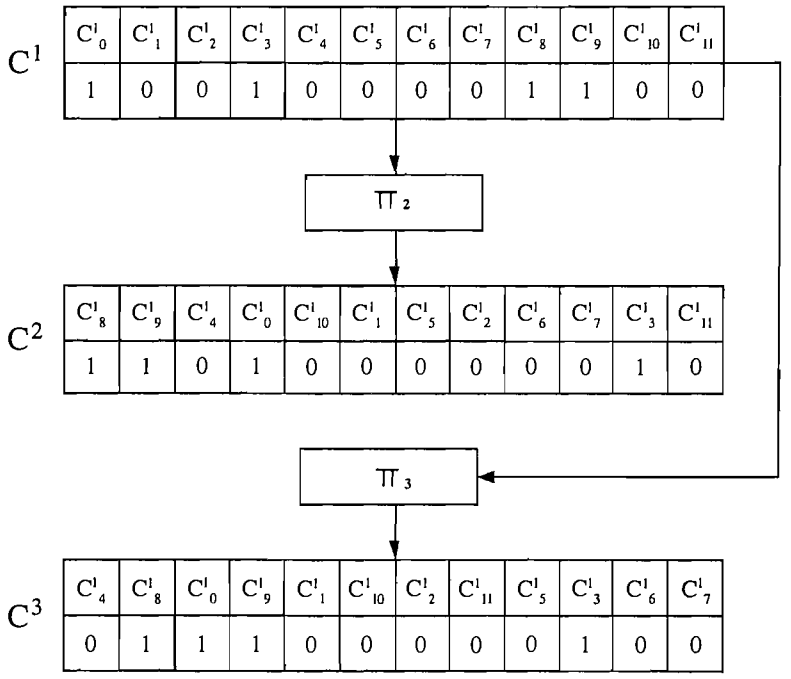


FIGURE 2.2: The interleavers π_2 and π_3 of Figure 2.1

of the PCM H^1 are randomly permuted to the 4^{th} and 6^{th} columns of the PCM H^2, respectively, therefore in Figure 2.2 the codeword bits C_0^1 and C_1^1 of codeword vector C^1 are interleaved to the 4^{th} and 6^{th} positions of codeword vector C^2, respectively. Furthermore, the codeword vectors C^2, $(1,1,0,1,0,0,0,0,0,0,1,0)$, and C^3, $(0,1,1,1,0,0,0,0,0,1,0,0)$ which are formed by interleaving the codeword vector C^1, $(1,0,0,1,0,0,0,0,1,1,0,0)$ satisfy that $C^2 \cdot H^2 = 0$ and $C^3 \cdot H^3 = 0$. As seen in Figure 2.3, the codeword of LDPC C is the intersection, i.e. the common symbols of the super-codes C^1, C^2 and C^3. More explicitly, the codeword C of the LDPC (N, K) code should be checked by the PCM H, which is the concatenation of the $J = 3$ PCMs of the super-codes C^1, C^2 and C^3, therefore we have

$$C \cdot H^1 = C \cdot H^2 = C \cdot H^3 = 0. \tag{2.1}$$

2.1.1.2 GLDPC codes

This example of a classic LDPC [12] code can be generalized for the sake of constructing GLDPC codes. The SPC $(n, n-1)$ code is used as the constituent code, when constructing classic LDPC codes [12], while a more general class of (n, k) block codes may be used as constituent codes, when constructing GLDPC codes.

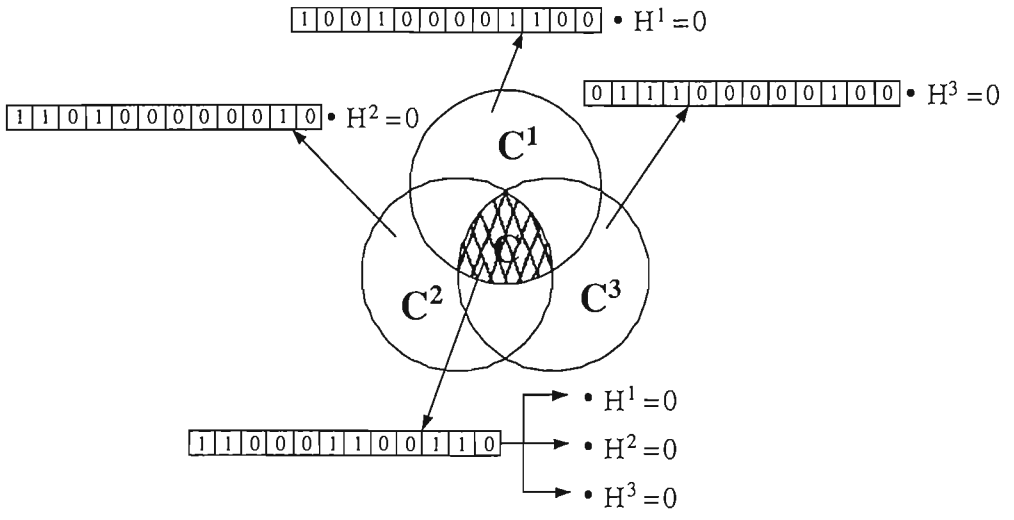


FIGURE 2.3: LDPC codeword C is the intersection of the super-codes C^1, C^2 and C^3

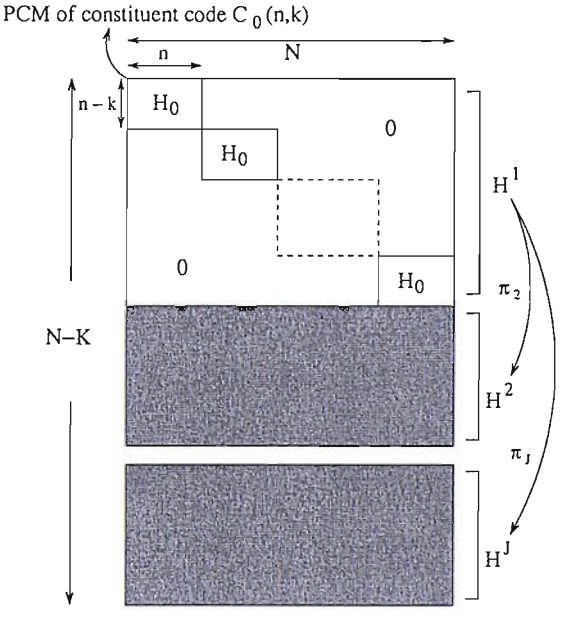


FIGURE 2.4: Parity-check matrix H of a GLDPC (N, K) code

The (n, k) constituent codesmay be binary, such as binary BCH codes [4, 5, 6] or LDPC codes [12, 13] as well as nonbinary BCH codes [4, 5, 6], Reed Solomon codes [7] or nonbinary LDPC codes [45]. We observe by comparing Figures 2.1 and 2.4 that the matrix H_0 now becomes of dimension $(n-k) \times n$ which is the PCM of a constituent code $C_0(n, k)$ instead of dimension $1 \times n$. The first PC sub-matrix H^1 of the super-code C^1 portrayed in the top third of Figure 2.4 produces the direct concatenation of N/n number of constituent codes $C_0(n, k)$ according

to [34]. Hence we have [34]

$$C^1 = \bigoplus_{l=1}^{N/n} C_0, \tag{2.2}$$

where N is the codeword length of the GLDPC (N, K) code, n is the codeword length of the constituent code $C_0(n, k)$ and $\bigoplus$ represents the concatenation operation. Finally, in analogy to Figure 2.1 characterizing the same process for classic LDPC codes, the PCM H of the GLDPC (N, K) code is constructed from the concatenation of the J number of PC submatrices $(H^1 \cdots H^J)$, which are the PCMs of the super-codes $(C^1 \cdots C^J)$, respectively. Thus, the codewords of a GLDPC (N, K) code may be viewed as the intersection of the codewords of the J super-codes [34]:

$$C = \bigcap_{j=1}^{J} C^j. \tag{2.3}$$

Accordingly, the codeword C of the GLDPC (N, K) code should be checked by all the J PCMs of the super-codes $(C^1 \cdots C^J)$, therefore we have

$$C \cdot H^j = 0, \tag{2.4}$$

where $j \in \{1 \cdots J\}$. Furthermore, since the PC submatrices $H^2, \cdots, H^J$ are derived by permuting the columns of the first PC submatrix H^1 without interleaving the elements of the columns, the codewords of the super-codes $C^j, j \in \{2 \cdots J\}$ are constituted by random permutations of the codewords of the super-code C^1, which is expressed as $C^j = \pi_j(C^1)$, where π_j represents the corresponding symbol-interleaver.

2.1.2 Graphical Concept

2.1.2.1 Classic LDPC codes

Fig. 2.5 portrays the bipartite graph of the classic LDPC (12,3) code [12] defined in Fig. 2.1. As shown in Fig. 2.5, the upper part contains $N = 12$ LDPC encoded symbol nodes, while the lower part represents $J \times N/n = 3 \cdot 12/4 = 9$ SPC (4,3) constituent code nodes. More explicitly, the 9 constituent codes are simple binary SPC $(4, 3)$ codes. An edge between an LDPC encoded symbol node and a constituent code node indicates that the corresponding symbol belongs to particular constituent code. Therefore, a 12-symbol LDPC-encoded word seen at the top of

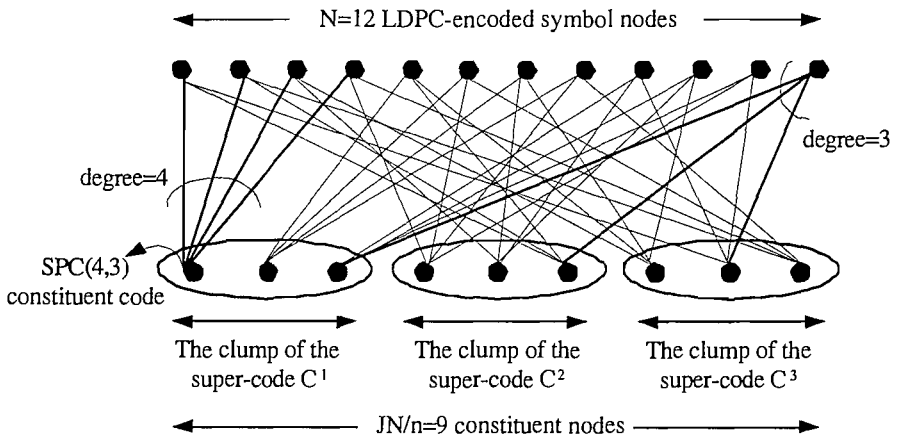


FIGURE 2.5: The bipartite graph of the LDPC (12,3) code using the PCM of Fig. 2.1.

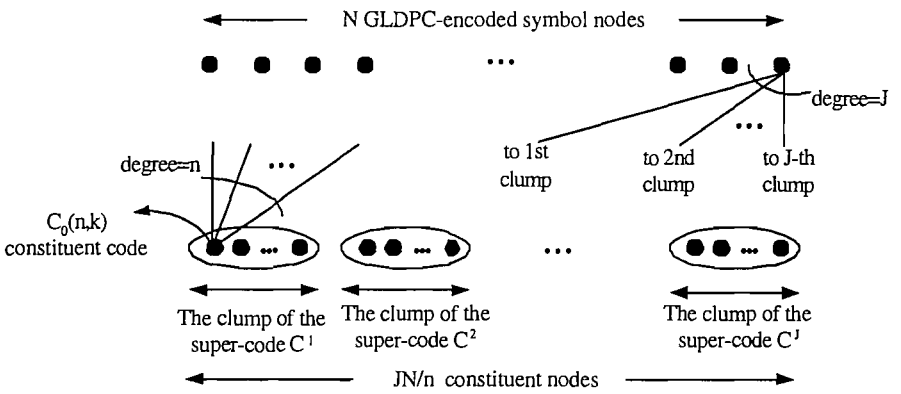


FIGURE 2.6: The bipartite graph of the GLDPC (N, K) code. The corresponding classic LDPC bipartite graph is shown in Figure 2.5

Figure 2.5 is a valid codeword of the resultant LDPC code, if and only if the 9 constituent code nodes of 4 incoming symbols seen at the bottom of Figure 2.5 belong to the valid SPC $(4, 3)$ codewords. Furthermore, the $J = 3$ edges stemming from every single LDPC-encoded symbol node are connected to specific SPC (4,3) constituent code nodes belonging to different so-called clumps [34] of super-codes $C^1, \cdots, C^3$. Hence, we can also see in Fig. 2.5 that the so-called degrees of the LDPC-encoded symbol nodes and those of the SPC (4,3) constituent code nodes are always $J = 3$ and $n = 4$, respectively, which are defined as the number of super-codes and the codeword length of the constituent code SPC $(n, n-1)$.

2.1.2.2 GLDPC codes

Similar to the classic LDPC code [12] of Fig. 2.5, the family of GLDPC code can also be described using a random regular bipartite graph. In this sense, GLDPC codes may also be viewed as the generalization of Tanner codes [17], as depicted in Fig. 2.6. The SPC $(n, n-1)$ code of classic LDPC codes is replaced by a more general (n, k) block code used as the constituent code $C_0(n, k)$. The GLDPC code's length is N symbols, which is the number of GLDPC-encoded symbol nodes seen at the top of Fig. 2.6. The bottom of Fig. 2.6 holds $J \times N/n$ $C_0(n, k)$ constituent code nodes. The degree of the constituent code node is equal to n, while again the constituent code itself is defined as $C_0(n, k)$ in Fig. 2.6. The degree of the GLDPC-encoded symbol nodes is J, which implies that every symbol node is connected to J number of $C_0(n, k)$ constituent codes represented by J different clumps of super-codes $C^j, j \in \{2 \cdots J\}$, respectively. In other words, every GLDPC-encoded symbol is jointly determined by the J $C_0(n, k)$ constituent codes and each $C_0(n, k)$ constituent code belongs to different super-codes.

Similar to the concept of *cycle* [17] in LDPC codes, the *cycle* in GLDPC can also be observed in its bipartite graph, which is defined as the closed loop in the bipartite graph. Thus the length of *cycle* is the number of connections between the GLDPC-encoded symbol nodes and the $C_0(n, k)$ constituent code nodes within the closed loop. Figure 2.7 is a example for the bipartite graph representations of the *length-6 cycle* and *length-4 cycle* in the GLDPC code, which are closed loops with 6 and 4 connections, respectively. Notice that each GLDPC-encoded symbol node or $C_0(n, k)$ constituent code node is capable of benefitting from more information provided by other nodes within the *cycle* if the length of the *cycle* is long. The shortest possible *cycle* of Figure 2.6 is a *length-4 cycle* as shown in Figure 2.7. Recall that in Section 2.1.1, we introduced the permutation function π_j, which allowed us to generate the PC matricies $H^j, j = 2, \cdots, J$ by permuting the columns of of the PC matrix H^1 without interleaving the elements of the columns. Therefore, in practice π_j is chosen at random, but by avoiding that two (or more) GLDPC-encoded symbol nodes are connected to the same J number of $C_0(n, k)$ constituent code nodes, since this would create short *cycles* of length 4 in the Tanner graph [34].

Furthermore, although the PCM of the GLDPC code is constructed by the PCMs of the constituent codes, so far there may be no efficient encoding procedure based

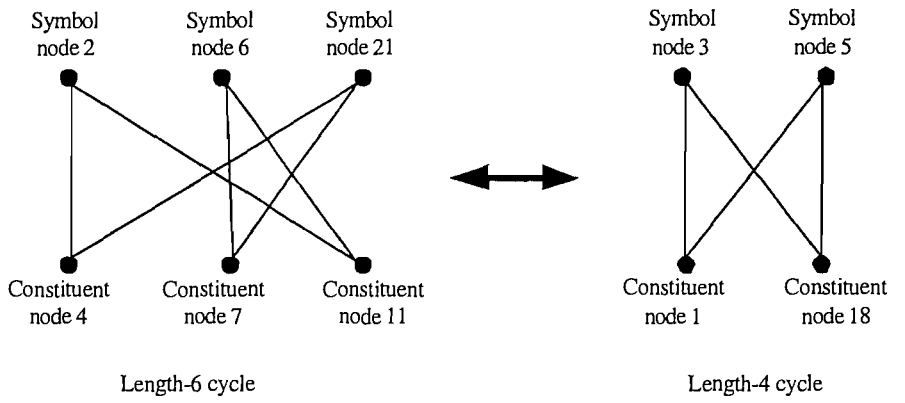


FIGURE 2.7: The example for the bipartite graph representation of the length-6 cycle and the length-4 cycle in GLDPC codes.

only on the constituent codes that does not need the construction of the generator matrix for GLDPC codes, which was also shown in Section 4.5, p.118 of [34]. Hence, the encoding procedure of the GLDPC code is the same as that of the LDPC code [46], which needs the construction of the generator matrix of the GLDPC code.

2.2 Coding Rate [34]

The code rate of a GLDPC code can be lower-bounded by observing its parity-check matrix as suggested in [34]. The number of rows in each PC submatric H^j, $j \in \{1 \cdots J\}$, corresponds to $(n-k)N/n$. Thus the total number of rows $(N-K')$ in the PCM H satisfies

$$N - K' = \frac{J(n-k)N}{n} = J(1-r_0)N \tag{2.5}$$

where $r_0 = k/n$ denotes the rate of the constituent code $C_0(n,k)$. If we assume that the PCM H is has full rank, i.e. when all of its rows are independent, then we have [34]:

$$R = \frac{K'}{N} = 1 - J(1-r_0) \tag{2.6}$$

However, in practice it cannot be readily guaranteed that the rows of H are independent. Hence the actual dimension K of the GLDPC may be higher than K'.

Constituent code				GLDPC Code
Type	n	k	r_0	Minimum Rate
Hamming	7	4	0.57	0.143
Hamming	15	11	0.73	0.467
Shortened Hamming	12	8	0.67	0.333
Shortened Hamming	10	6	0.6	0.2
Hamming	31	26	0.84	0.677
shortened Hamming	25	20	0.8	0.6
shortened Hamming	20	15	0.75	0.5
shortened Hamming	19	14	0.737	0.474
Hamming	63	57	0.90	0.81
shortened Hamming	50	44	0.88	0.76
shortened Hamming	37	31	0.838	0.676

TABLE 2.1: Coding rate of $J = 2$-level GLDPC codes using different constituent codes C_0. [34]

Consequently, the lower bound on the rate of a GLDPC code is [34]:

$$R = \frac{K}{N} \geq 1 - J(1 - r_0). \tag{2.7}$$

2.3 A Special Case: GLDPC Codes Having J = 2 Levels

As shown in Section 2.2, the coding rate R decreases, when the number of super-codes J increases. Furthermore, it has been shown in [31, 32, 33, 34] that GLDPC codes based on binary Hamming or BCH constituent codes are asymptotically good, even if we have as low a number of levels as $J = 2$ and that iterative decoding is very simple to implement in this case. Accordingly, binary GLDPC codes having $J = 2$ levels exhibit the highest possible code rate as well as a low-complexity decoder structure, which are desirable properties in practical applications. Thus, in our study, we consider only GLDPC codes having $J = 2$ levels. Table 2.1 shows the coding rate bound for $J = 2$-level GLDPC codes based on different constituent codes.

A $J = 2$-level GLDPC code is the intersection of the super-codes C^1 and C^2, each of them being composed of N/n independent constituent codes. The constituent codes belonging to the first super-code C^1 are referred to here as the *upper codes*,

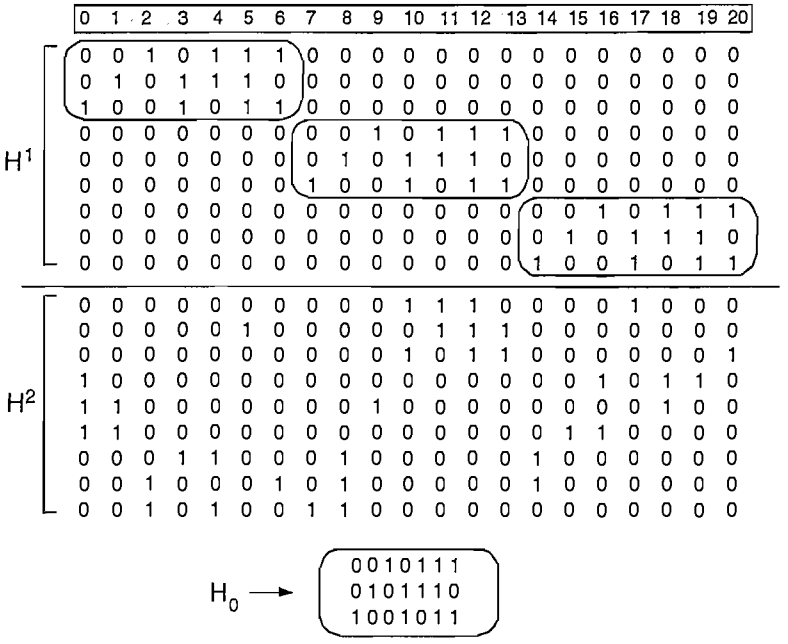


FIGURE 2.8: The $(N-K)\times N = 18\times 21$-dimensional Parity Check Matrix $\boldsymbol{H}$ of the GLDPC (21,3) code using the binary (7,4) Hamming code as its constituent code, where we have $N = 21$, $K = 3$ $J = 2$ and $n = 7$.

while the ones belonging to the second super-code C^2 are the *lower codes*. Furthermore, for practical $J = 2$-level (N, n) GLDPC codes, it is only possible to construct a meritorious PCM H in which no undesirable short cycles of length 4 appear in the graph, if we have $N/n \geq n$ [47]. Therefore, the $(N, 2, n)$ GLDPC codes should always satisfy $N/n \geq n$.

2.4 GLDPC Encoding

In this section, we discuss the encoding procedure of the binary $J = 2$-level GLDPC code as well as give an example. Again, the encoding procedure of the GLDPC code is the same as that of the LDPC code [46], which needs the construction of the generator matrix of the GLDPC code, since there may be no efficient encoding procedure based only on the constituent codes that does not need the construction of the generator matrix for GLDPC codes, which was also mentioned in Section 4.5, p.118 of [34]. Therefore, here we use the same encoding procedure of LDPC code which is described in the chapter 2.4 of [46].

The J-level (N, K) GLDPC code is used in this example, where we have $N = 21$, $K = 3$ and $J = 2$ and adopt the $(7, 4)$ binary Hamming code as our constituent

code. The PCM $\boldsymbol{H}$ is constructed as shown in Figure 2.8, where the upper part $\boldsymbol{H}^1$ is a (9×21)-dimensional block diagonal matrix having the matrix elements $\boldsymbol{H}_0$. In this example, the $(3 \times 7)-$ dimensional matrix H_0 is the PCM of the (7,4) binary Hamming Code. The lower part (9×21)-dimensional $\boldsymbol{H}^2$ of H is generated by permuting the columns of the upper matrix without modifying the elements of the columns. This operation is formulated as $\boldsymbol{H}^2 = \pi_2(\boldsymbol{H}^1)$ as we mentioned in Section 2.1.1. For example, the 20th and 19th columns of H^2 is permuted from the 0th and 9th columns of H^1, respectively.

In order to generate systematic GLDPC code according to the encoding procedure described in chapter 2.4 of [46] for LDPC codes, the $M \times N$-dimensional PCM $\boldsymbol{H}$ is divided into an $(N-K) \times (N-K) = M \times M$-dimensional matrix $\boldsymbol{A}$ and an $M \times (N-M)$-dimensional matrix $\boldsymbol{B}$ as shown below [46]:

$$\boldsymbol{H}_{(M \times N)} = \left[\boldsymbol{A}_{(M \times M)} \middle| \boldsymbol{B}_{(M \times (N-M))}\right] \tag{2.8}$$

Then the $(K \times N)$-dimensional generator matrix G may be expressed as [46]:

$$\boldsymbol{G}_{(K \times N)} = \left[\boldsymbol{I}_{(K \times K)} \middle| \boldsymbol{B}^T \cdot (\boldsymbol{A}^T)^{-1}_{(K \times M)}\right] \tag{2.9}$$

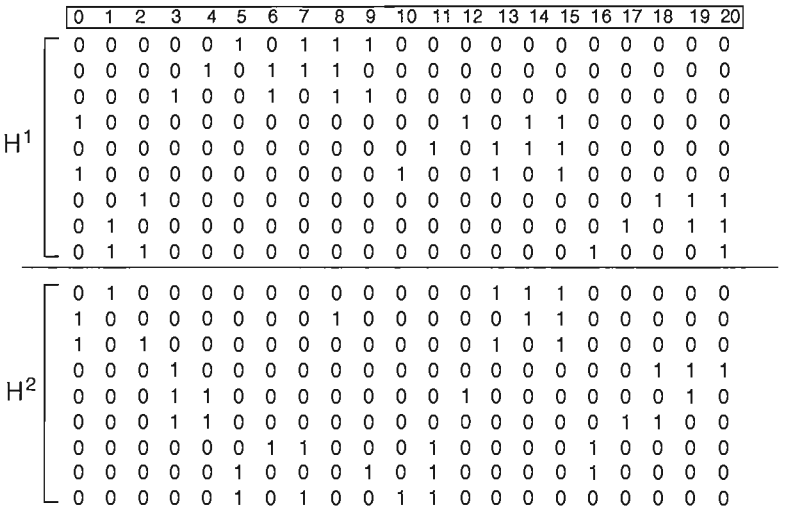
In our specific example, we have

$$\boldsymbol{H}_{(18 \times 21)} = \left[\boldsymbol{A}_{(18 \times 18)} \middle| \boldsymbol{B}_{(18 \times 3)}\right] \tag{2.10}$$

and

$$\boldsymbol{G}_{(3 \times 21)} = \left[\boldsymbol{I}_{(3 \times 3)} \middle| \boldsymbol{B}^T \cdot (\boldsymbol{A}^T)^{-1}_{(3 \times 18)}\right]. \tag{2.11}$$

However, it cannot be guaranteed that the submatrix $\boldsymbol{A}^T$ is invertible. Hence as it was suggested in [46], the columns of the PCM $\boldsymbol{H}$ may have to be reordered for the sake of being able to calculate the inverse of the matrix $\boldsymbol{A}^T$ as it was suggested in [46]. If the original matrix $\boldsymbol{A}^T$ is singular, we will randomly select a column from $\boldsymbol{B}$ and swap it with a randomly chosen column of $\boldsymbol{A}$. This process continues until the matrix $\boldsymbol{A}$ becomes non-singular. Thus upon re-ordering the columns of the matrix $\boldsymbol{H}$, we arrive at the reordered $M \times N = 18 \times 21$-dimensional PCM $\boldsymbol{H}_r$ as shown in Figure 2.9. For example, 0th and 20th columns of $\boldsymbol{H}$ in Figure 2.8 became 2nd and 3rd columns of $\boldsymbol{H}_r$ in Figure 2.9, respectively. Then the $K \times N = 3 \times 21$-dimensional generator matrix $\boldsymbol{G}$ as shown in Figure 2.10



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
H1	0	0	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0
	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0	0	0	0	0
	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0
	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1
	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
H2	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0
	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	0	0	0	0	0
	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0
	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0
	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	0	0	0	0
	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	1	0	0	0	0
	0	0	0	0	0	1	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0

FIGURE 2.9: The $(N-K) \times N = 18 \times 21$-dimensional reordered PCM $\boldsymbol{H}_r$ of the GLDPC (21,3) code using the constituent Hamming (7,4) code, where we have $N = 21$, $K = 3$, $J = 2$ and $n = 7$.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
0	1	0	0	1	0	1	1	1	0	0	1	1	0	1	0	1	1	0	0	0
0	0	1	1	0	1	0	0	0	1	1	0	0	0	1	1	0	0	1	1	1

FIGURE 2.10: The $K \times N = 3 \times 21$-dimensional generator matrix $\boldsymbol{G}$ corresponding to the PCM $\boldsymbol{H}_r$ of the GLDPC (21,3) code using the constituent Hamming (7,4) code, where we have $N = 21$, $K = 3$, $J = 2$ and $n = 7$.

is constructed based on the reordered new PCM $\boldsymbol{H}_r$, and the source $K = 3$-bit-sequence $\boldsymbol{s} = [100]$ is then multiplied by $\boldsymbol{G}$ to get the encoded $N = 21$-bit-sequence $\boldsymbol{u} = [100000000000011000000]$, as shown in Figure 2.11.

Furthermore, the reordered PCM $\boldsymbol{H}_r$ is also the column-permutation of $\boldsymbol{H}$ thus it can be represented as $\boldsymbol{H}_r = \pi'(\boldsymbol{H})$. Therefore, for the sake of recovering L constituent codewords which belong to the super-code C^1 from the codeword $\boldsymbol{u}$ of the GLDPC code, the codeword $\boldsymbol{u}$ needs to be de-interleaved by the de-interleaver π'_1, as shown in the left part of Figure 2.11 where $\pi'_1 = \pi'$. Similarly, in order to recover L constituent codewords which belong to the super-code C^2, the codeword $\boldsymbol{u}$ needs to be de-interleaved by the de-interleaver π'_2, as shown in the right part of Figure 2.11 where $\pi'_2 = \pi'(\pi_2)$.

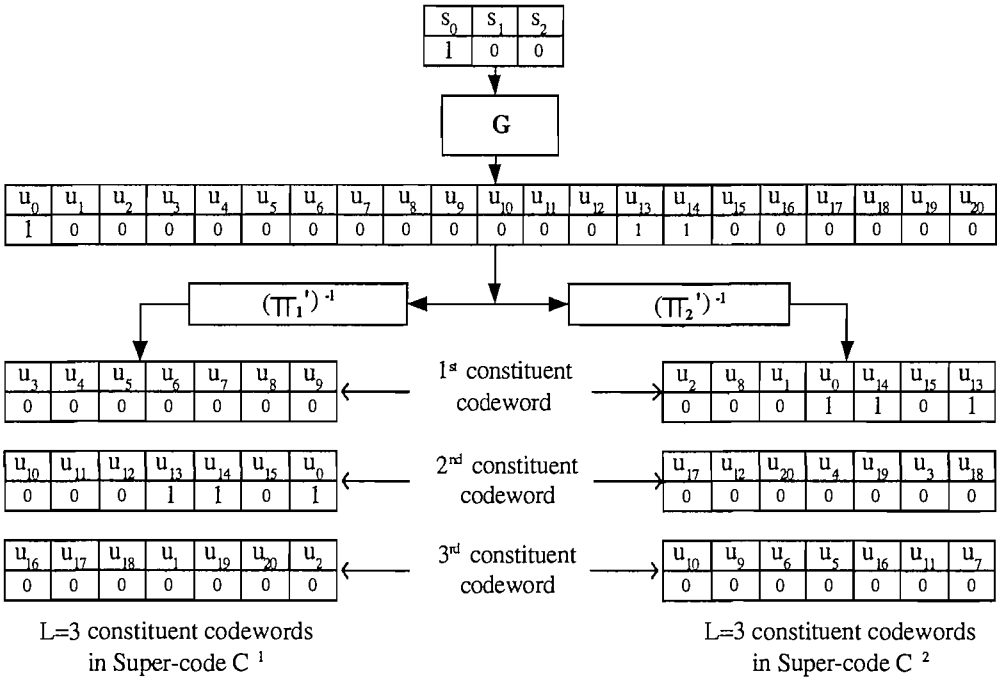


FIGURE 2.11: The encoding process of the GLDPC (21,3) code using the constituent Hamming (7,4) code, where we have $N = 21$, $K = 3$, $J = 2$ and $n = 7$ and the relationship between codeword $\boldsymbol{u}$ and the constituent codes in super-codes C^1 and C^2.

2.5 GLDPC Soft-In/Soft-Out Decoding

2.5.1 Introduction

Gallager presented an iterative decoding scheme designed for LDPC codes in [12]. This algorithm computes iteratively the probability of each coded symbol given a set of received channel observation, where the probability of correct detection increases upon increasing the iteration index. The goal is to estimate the *a posteriori* probability, namely the probability of the coded symbols, given all demodulator output samples. This algorithm is similar to the one proposed in [48] for the well known turbo codes and may be considered as the ancestor of all turbo decoding techniques. Furthermore, it has been known by Boutros *et al.* in [31] that turbo codes can be described as a particular case of GLDPC codes, where the interleaver acts only on the information symbols but not on the parity symbols. Thus, GLDPC codes are decoded using the same approach. Explicitly, for each coded symbol, we compute the probability of each decoded symbol given the corresponding demodulated sample (*a posteriori* information) and the so-called

extrinsic information, assuming that the symbol belongs to the super-code C^1. The latter is fed through the interleaver to the decoder of super-code C^2 as the *a priori* information. We then compute the probability of each symbol given the corresponding demodulated sample (*a posteriori* informaiton) and the *extrinsic* information and assume now that it belongs to, the super-code C^2. The resultant probability is returned of the first super-code's decoder as the *a priori* information and this process is continued while a legitimate GLDPC codeword is found or the affordable number of iterations is exhausted.

A decoding algorithm that processes the soft-decision inputs and produces soft-decision outputs is referred to as a Soft-In/Soft-Out (SIOS) decoding algorithm. A GLDPC code can be efficiently decoded based on iterative SISO decoding of the individual constituent codes, where the code's performance and decoding complexity are heavily dependent on the SISO decoding algorithm employed. Many iterative SISO decoding algorithms [35, 36, 37, 39] have been proposed for decoding linear block codes, both optimal [35] or suboptimal [36, 37, 39]. The most well-known SISO decoding algorithm is the MAP (maximum *a posteriori* probability) decoding algorithm that was devised by Bahl, Cocke, Jelinek, and Raviv (BCJR) in 1974 [35]. The BCJR algorithm was devised to minimize the bit-error probability and to provide reliability values of the decoded symbols. The MAP algorithm (or its suboptimum versions) constitute the heart of turbo or iterative decoding [48, 49]. The trellis-based MAP algorithm was used in [31, 32] for decoding GLDPC codes, whereas the Johansson-Zigangirov *A Posteriori* Probability (APP) algorithm [36], which operates with the aid of a single forward recursion through a syndrome trellis was used in [33]. A range of other sub-optimal decoding algorithms, such as the Chase's algorithm, Kaneko's algrithm and MAX-Log-MAP algorithm were employed in [38], [40] and [41] for decoding GLDPC codes, respectively.

2.5.2 The Constituent Decoder Using Log-MAP Algorithm

In our implementation, we used the Log-MAP algorithm [50] for the sake of striking an attractive compromise between complexity and coding performance. Figure 2.12 shows a single constituent decoder of the GLDPC SISO decoder scheme, which employs Log-MAP algorithm. Consider a binary (n, k) linear block code C_0 as a constituent code of the GLDPC codes. Let $\boldsymbol{u} = (u_1, u_2, \cdots, u_n)$ be a

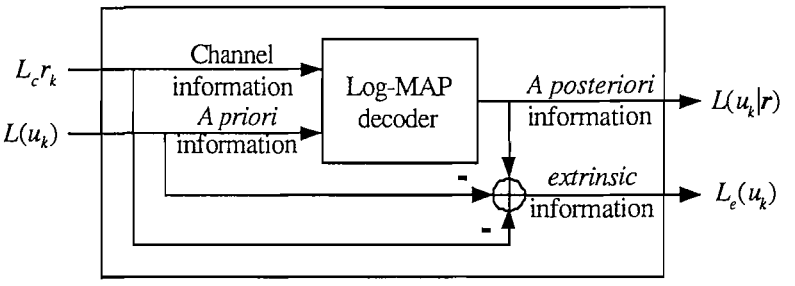


FIGURE 2.12: A constituent decoder using Log-MAP algorithm, which is employed in a GLDPC decoder, showing the input information received and output information

codeword of C_0 and $\boldsymbol{r} = (r_1, r_2, \cdots, r_n)$ be the received noisy sequence from channel. In Figure 2.12, we can see that the Log-MAP decoder accepts two inputs, the *a priori* information $L(u_k)$ and the soft demodulator output $L_c r_k$, where L_c is the channel reliability value. At its output, it produces the *a posteriori* information $L(u_k|\boldsymbol{r})$ as shown in Figure 2.12. The decoder has to calculate the extrinsic information $L_e(u_k)$ imposed by the code constraints from the demodulator's soft-output sequence $\boldsymbol{r}$, but excluding the demodulator's soft-output sample r_k directly engendered by the transmitted data bit, u_k. Hence this information is referred to as the *extrinsic* information for the bit u_k. During the iterative decoding process, only the *extrinsic* information $L_e(u_k)$ will be forwarded to the other constituent decoder as the *a priori* value $L(u_k)$.

2.5.3 Iterative SISO Decoding Algorithm for $J = 2$-Level GLDPC Codes

We now describe how the iterative decoding of $J = 2$-level GLDPC codes is carried out. Figure 2.13 shows the structure of the SISO decoder invoked for $J = 2$-level GLDPC codes. Each super-code's decoding is performed by means of a SISO decoder. The upper part of Figure 2.13 represents the SISO decoder of super-code C^1, while the lower part is the SISO decoder of super-coder C^2. The complexity of each SISO super-code decoder is modest since each super-code is constructed of $L = N/n$ number of *independent* constituent codes of small code length n. To elaborate a little further, the MAP SISO decoder's complexity is typically lower if it carries out the backwards and forwards recursion for shorter constituent codes, as it transpires from the operations detailed in Section 5.3.3, p114-122 of [51]. As

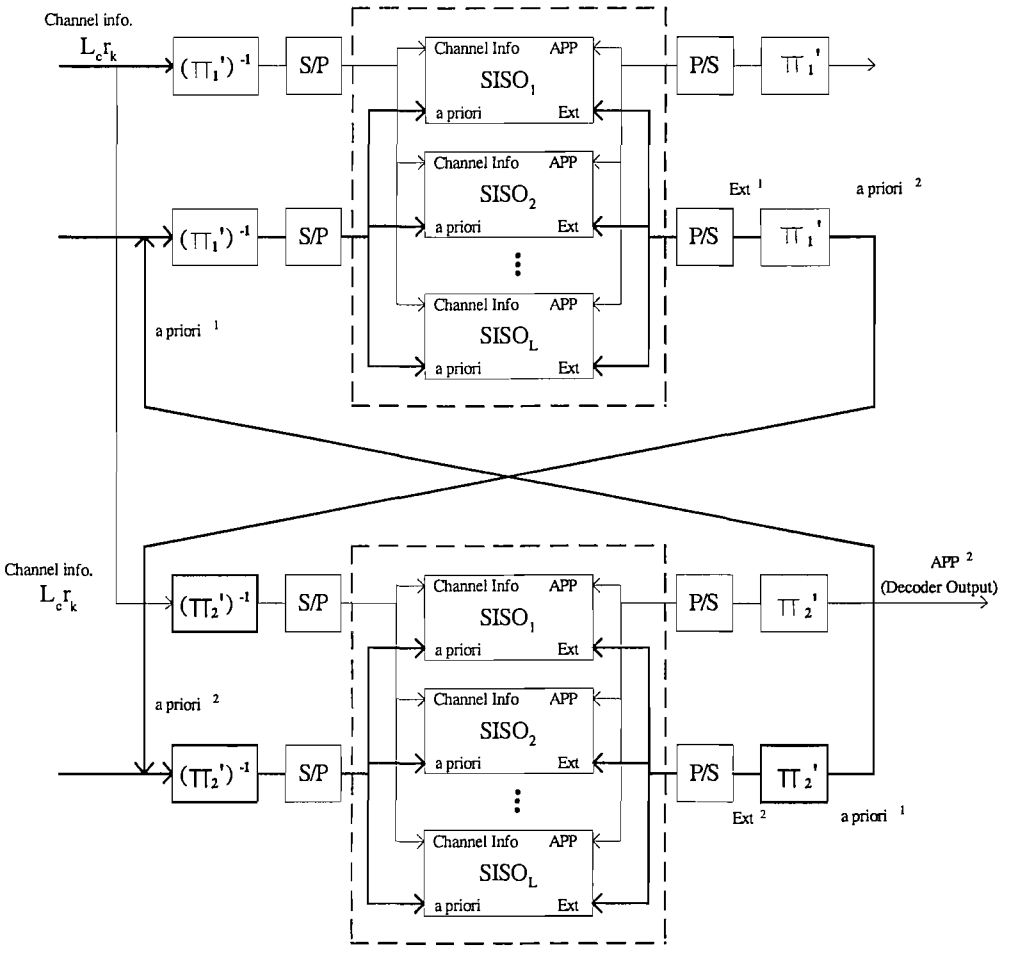


FIGURE 2.13: The structure of SISO decoding for $J = 2$-level GLDPC codes using $L = N/n$ number of (n,k) constituent decoders. In our example we have $L = 21/7 = 3$ (7,4) binary Hamming decoders.

depicted in Fig. 2.13, L low-complexity SISO decoders of the constituent codes operate in parallel in each super-code's decoder.

A step-by-step description of the iterative SISO decoding algorithm for the $J = 2$-level GLDPC codes is described as follows [31, 32, 34]:

1. For transmission over a Gaussian or fading channel using BPSK modulation, the conditional LLRs $L(r_k|u_k)$ can be calculated as [51]:

$$\begin{aligned} L(r_k|u_k) &= \ln\left(\frac{P(r_k|u_k=1)}{P(r_k|u_k=0)}\right) \\ &= \ln\left(\frac{\exp\left(-\frac{E_c}{2\sigma^2}(r_k-a)^2\right)}{\exp\left(-\frac{E_c}{2\sigma^2}(r_k+a)^2\right)}\right) \\ &= L_c r_k, \end{aligned} \tag{2.12}$$

where

$$L_c = 4a\frac{E_c}{2\sigma^2}, \tag{2.13}$$

is defined as the channel reliability value, and a is the fading amplitude. We have $a = 1$ for non-fading AWGN channels. Hence, for BPSK transmission over a possibly fading Gaussian channel, the conditional LLR $L(r_k|u_k)$, which is referred to as the soft output of the channel, is simply the matched filter output r_k multiplied by the channel reliability value L_c. Then the channel's soft output sequence $L_c\boldsymbol{r}$ is fed through the de-interleavers $(\pi_1')^{-1}$ and $(\pi_2')^{-1}$ of Figure 2.13 to the upper and lower decoders, respectively.

2. The de-interleaved soft channel output $L_c r_k$ and *a priori* information $L(u_k)_1, k = 1, \cdots, N$ are received by the upper decoder seen in Figure 2.13 and delivered in parallel to $L = N/n$ number of constituent SISO decoders. Note that during the first iteration the upper decoder has no *a priori* information about the symbols and hence $L(u_k)_1$ is set to 0 for all symbols, corresponding to an *a priori* probability of 0.5.

3. Following the decoding process of each constituent SISO decoder in the upper decoder, the *a posteriori* information $L(u_k|\boldsymbol{r})_1$ and the *extrinsic* information $L_e(u_k)_1$ are generated for each GLDPC coded symbol $u_k, k = 1, \cdots, N$. The latter is fed through the appropriate interleaver and de-interleaver π_1' and $(\pi_2')^{-1}$ of Figure 2.13 to the lower decoder of super-code C^2 as the *a priori* information.

4. Next the lower decoder of super-code C^2 comes into operation. It receives the interleaved soft channel output $L_c r_k$ and the *a priori* information $L(u_k)_2, k = 1, \cdots, N$. The L constituent SISO decoders of the lower decoder seen in Figure 2.13 generate the *a posteriori* information $L(u_k|\boldsymbol{r})_2$ and the *extrinsic* information $L_e(u_k)_2$ for each coded symbol $u_k, k = 1, \cdots, N$. The latter is then fed through the interleaver and de-interleaver π_2' and $(\pi_1')^{-1}$ of Figure 2.13 to the upper decoder of super-code C^1 as *a priori* information.

5. A complete decoding iteration consists of the successive decoding of the upper and the lower super-codes C^1 and C^2, i.e. of two decoding steps. A tentative hard decision concerning the binary value of each coded symbol $\hat{u_k}, k = 1, \cdots, N$ is made based on the *a posteriori* information of the lower decoder $L(u_k|\boldsymbol{r})_2$. Then this tentatively decoded codeword is multiplied with the PCM H^T to generate the syndrome vector.

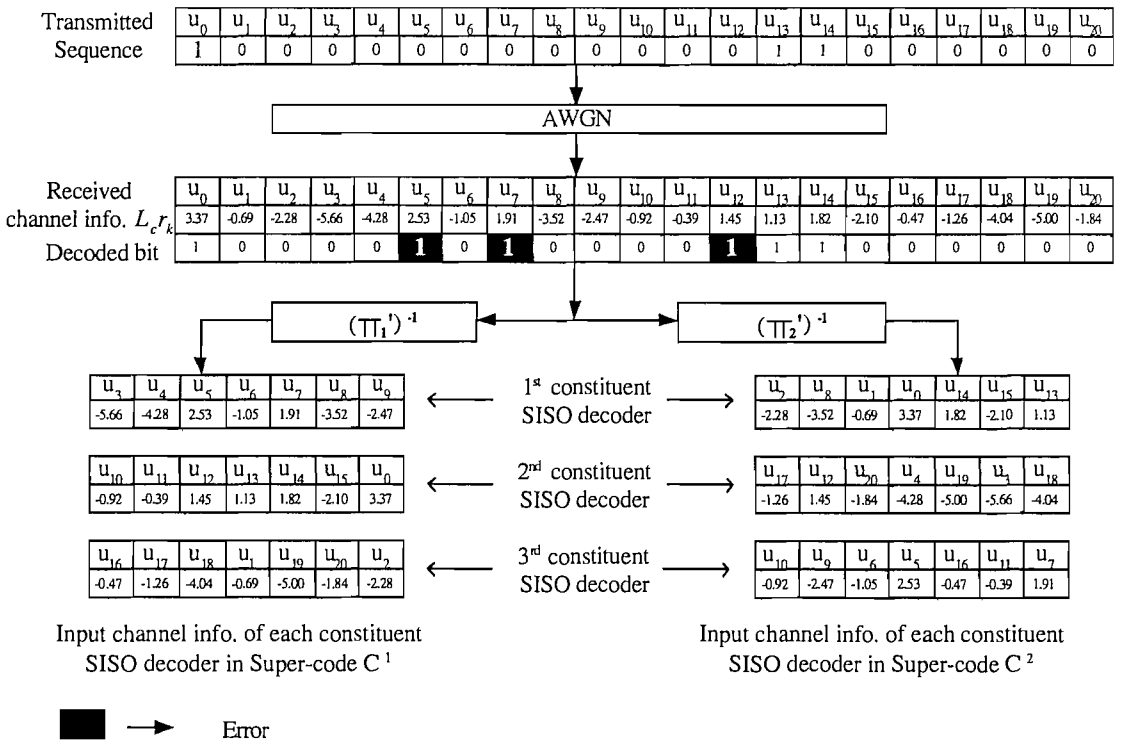


FIGURE 2.14: De-interleaving process of the received channel output sequence. The entire decoder was depicted in Figure 2.13

6. The GLDPC decoder repeats steps(2) to (5), until the resultant syndrome vector becomes an all-zero vector or the maximum number of iterations is reached. If the syndrome vector is an all-zero vector, we declare that a legitimate codeword has been found. By the contrast, if the syndrome vector is not an all-zero vector and the maximum number of iteration is exhausted, we will declare a decoding failure and output the tentatively decoded codeword.

2.6 Decoding Example of the binary $J = 2$-level GLDPC code

In this section, we discuss an example of the iterative SISO decoding of $J = 2$-level GLDPC codes using the Log-MAP algorithm[51]. This example highlights how iterative decoding assists in correcting multiple errors. Our elaborations are based on Section 2.5.

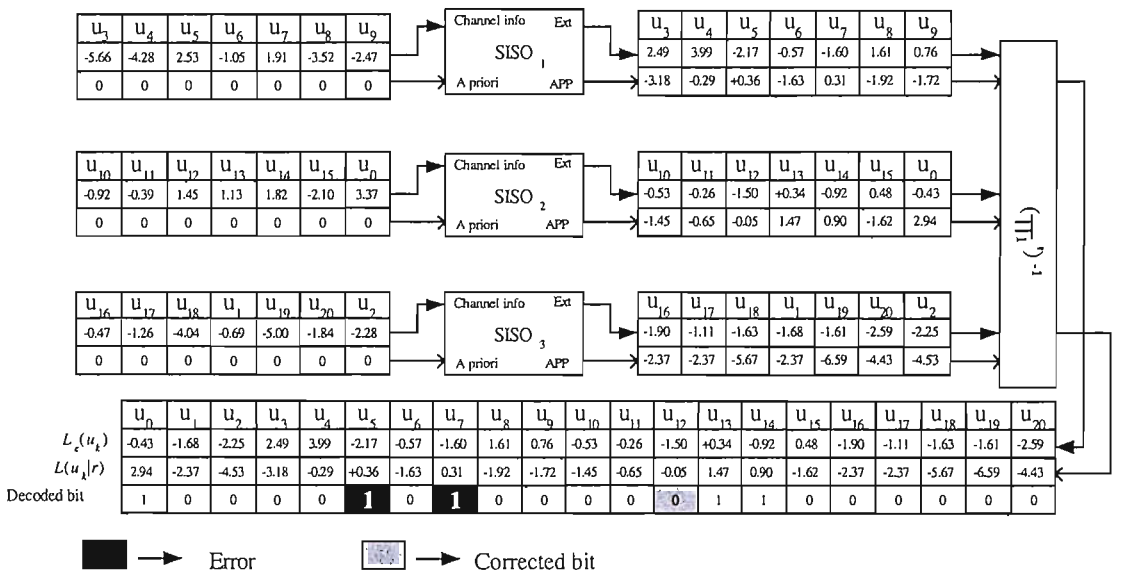


FIGURE 2.15: GLDPC SISO decoding example during the first iteration of the upper decoder seen in Figure 2.13

First of all, let us assume that the PCM $\boldsymbol{H}_r$ of Figure 2.9 is used and relevant generator matrix $\boldsymbol{G}$ is also seen in 2.10. The detailed encoding process is also described in Section 2.4 that the source $K = 3$-bit-sequence $\boldsymbol{s} = [100]$ is multiplied by $\boldsymbol{G}$ to get the encoded $N = 21$-bit-sequence $\boldsymbol{u} = [100000000000011000000]$, as shown in Figure 2.11. After encoding the source bit-sequence $\boldsymbol{s} = [100]$, the encoded sequence $\boldsymbol{u} = (u_0, \cdots, u_{20})$ is transmitted through an AWGN channel having a noise standard deviation of $\sigma = 0.936$ using BPSK modulation, as exemplified in Figure 2.14 based on our simulations. A logical 0 is transmitted as -1 and a logical 1 is transmitted as +1. Once the decoded bit stream was transmitted through the channel, the noise-contaminated received sequence shown in Figure 2.14 may be received. This corresponds to the soft output of the demodulator. As shown in Fig. 2.14, the received sequence has three erroneous bits, u_5,u_7 and u_{12}, according to the hard decision philosophy used. Then the resultant demodulator output sequence is appropriately deinterleaved for the upper decoder of super-code C^1 and for the lower decoder of super-code C^2, respectively as seen in Figure 2.13. In each super-code decoder, the interleaved channel information sequence is divided into $L = N/n = 21/7 = 3$ patterns for the $L = 3$ SISO decoders of the (7,4) binary constituent Hamming code.

At the beginning of each iteration, the upper decoder is activated first. Moreover, during the first iteration of the upper decoder, the *a priori* information fed to each of the $L = 3$ constituent decoders is set to zero. As depicted in Figure 2.15,

the $L = 3$ patterns of the demodulator's output information and the zero-value *a priori* information are fed into the $L = 3$ constituent decoders of the upper decoder. After the Log-MAP decoding of each (7,4) constituent Hamming code, the *a posteriori* information and the *extrinsic* information have been calculated and fed into the interleaver π_1 of Figure 2.15 for the sake of reinstating the original order of bits. The SISO MAP decoding result of the upper decoder is shown in Fig. 2.15, which is based on the *a posteriori* information $L(u_k|\boldsymbol{r})_1$. The results show that the upper decoder was capable of correcting the bit u_{12}. Furthermore, the magnitude of $L(u_k|\boldsymbol{r})_1$ for the two incorrectly decoding bits, namely for u_5 and u_7, is only 0.36 and 0.31, respectively. This is significantly lower than the magnitudes of the a posteriori information for most the other bits and indicates that the decoder is less confident concerning about these two bits being +1.

In the lower decoder, the extrinsic information $L_e(u_k)_1$ of upper decoder is fed through the deinterleaver π_2^{-1} of Figure 2.13 to the lower decoder as the *a priori* information. Again, the *a priori* information is partitioned into $L = 3$ sequents which belong to $L = 3$ SISO decoders of the (7,4) constituent Hamming codes in the lower decoder. Figure 2.16 characterizes the decoding process of the lower decoder of the super-code C^2 seen in Figure 2.13. Each SISO decoder of the (7,4) constituent Hamming code receives the demodulator's output information as well as the *a priori* information and outputs the *a posteriori* information as well as *extrinsic* information. The interleaver π_2 of Figure 2.13 collects all the sequences from the SISO decoder of the (7,4) constituent Hamming code to recover the original order of the bits. The *extrinsic* information can be used as the *a priori* information in the upper decoder during the next iteration after deinterleaving by deinterleaver π_1^{-1} of Figure 2.13. The lower decoder corrects the remaining two erroneous bits, namely u_5 and u_7.

2.7 Simulation Results

In this section, our simulation results will be discussed in order to characterize the achievable performance of the $J = 2$-level binary GLDPC codes introduced in the previous sections, when communicating over both AWGN and uncorrelated Rayleigh fading channels.

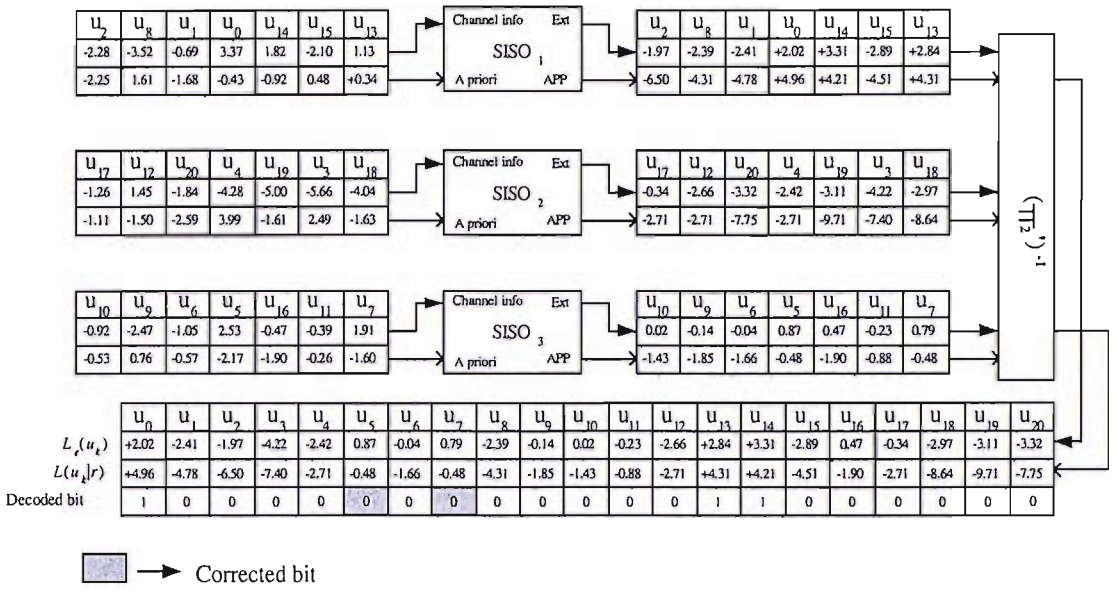


FIGURE 2.16: GLDPC SISO decoding example during the first iteration of the lower decoder seen in Figure 2.13

Using constituent Hamming (15,11) code with coding rate R=0.467			
GLDPC (N, K)	Channel	Decoding algorithm	Number of iterations
(300,140)	AWGN	Log-MAP	2, 4, 8, 12, 20, 35, 60
(1200,560)	AWGN	Log-MAP	2, 4, 8, 12, 20, 35, 60
(6000,2800)	AWGN	Log-MAP	2, 4, 8, 12, 20, 35, 60
(300,140)	URF	Log-MAP	2, 4, 8, 12, 20, 35, 60
(1200,560)	URF	Log-MAP	2, 4, 8, 12, 20, 35, 60
(6000,2800)	URF	Log-MAP	2, 4, 8, 12, 20, 35, 60

TABLE 2.2: Simulation parameters for three J = 2-level GLDPC codes investigated, when communicating over an AWGN channel and an uncorrelated Rayleigh fading (URF) channel using different maximum number of iterations.

2.7.1 Effect of the Number of Iterations

In this subsection, we will use three different GLDPC codes which have the same constituent Hamming (15,11) code, the same coding rate of $R = \frac{7}{15} = 0.467$ but different GLDPC code-lengths. The GLDPC code (N, K) listed in Table 2.2 represents a code having a coded block-length of N bits and conveying K information bits. The decoding algorithm used is the Log-MAP algorithm p.139 of [51]. In our experiments both an AWGN and an Uncorrelated Rayleigh Fading (URF) channel were applied. All system parameters are listed in Table 2.2.

The corresponding simulation results are shown in Figures 2.17- 2.22. All the figures confirm that as the number of iterations used by the decoder increases, the decoder performs significantly better. In order to analyze the effect of the number of iterations, we summarize both the E_b/N_0 required for achieving a BER of $10^{-4}$and the achievable coding gains, when communicating over two different channels in Table 2.3 and Table 2.4. These results were extracted from Figures 2.17 - 2.22. Furthermore, we introduce another quantity termed as the *iteration efficiency*, which is defined as follows. We record the coding gain achieved when using a maximum of 60 iterations as a reference and define the iteration efficiency as the percentage of the maximum achievable coding gain at a given number of iterations. This quantity is also recorded in Table 2.3 and Table 2.4.

Observe in Table 2.3 that upon using a maximum of I=12 iterations, all codes have already achieved over 95% of the maximum attainable coding gain in an AWGN channel. It may be hence inferred that the soft information based iterative decoder achieves most of its attainable coding gain after few iterations. Further iterations in excess of I=12 achieve only a modest further performance improvement at the cost of a high additional decoder complexity. Moreover, the GLDPC code having a block-length of 300 bits achieves a lower iteration efficiency than the GLDPC code having a block-length of 6000 bits, when the maximum number of iterations is I=12. In other words, as expected, the GLDPC code having a higher block-length is more efficient at the same number of iterations. It has been demonstrated in [34] that GLDPC codes can be considered as a generalization of product codes,hence having a longer block-length implies having a longer interleaver, when using the same constituent codes. Since the iterative decoding procedure involves passing the soft-information between the GLDPC encoded symbol nodes as well as the constituent code nodes and the GLDPC code having a higher block-length suffers from a lower correlation between the codeword bits, the performance of longer GLDPC codes may approach the maximum achievable coding gain at a lower number of iterations, when the GLDPC codes use the same constituent codes. The same phenomenon is also observed for transmission over uncorrelated Rayleigh fading channels in Table 2.4.

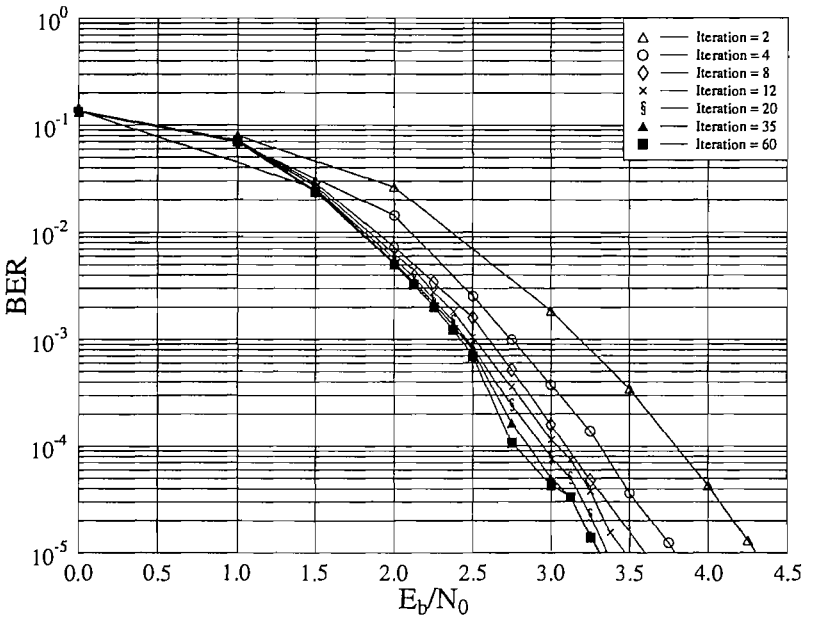


FIGURE 2.17: BER performance of the rate $R = 0.467$ GLDPC code $(300, 140)$ parameterised in Table 2.2, using different number of iterations, when communicating over AWGN channels.

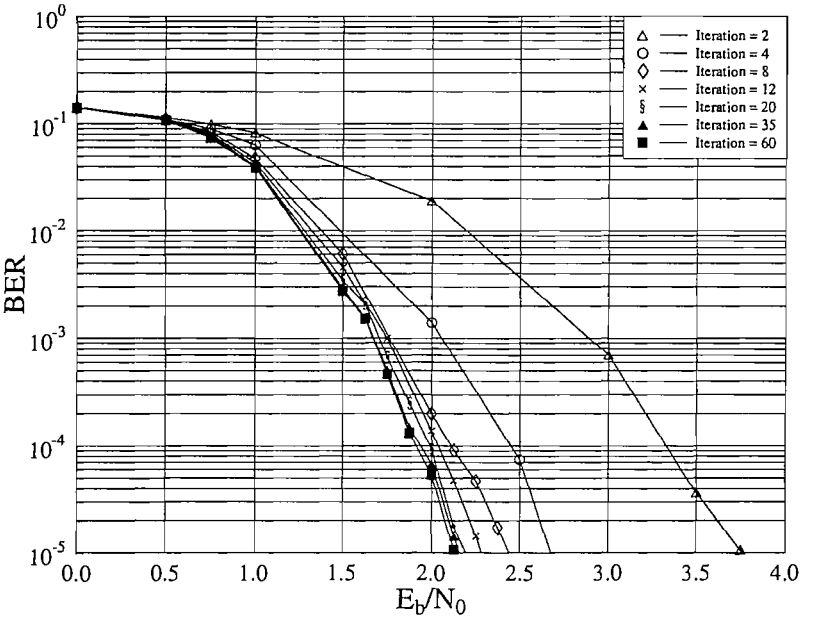


FIGURE 2.18: BER performance of the rate $R = 0.467$ GLDPC code $(1200, 560)$ parameterised in Table 2.2, using different maximum number of iterations, when communicating over AWGN channels.

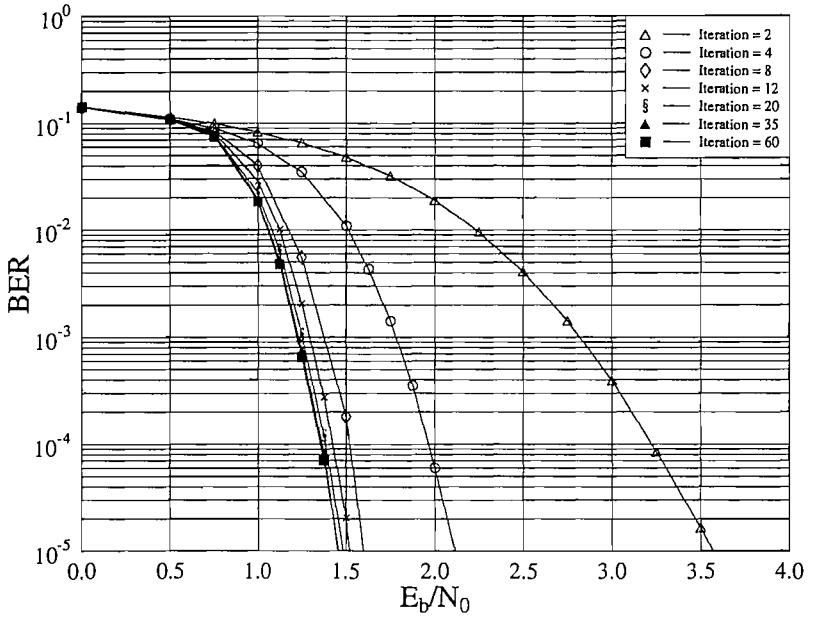


FIGURE 2.19: BER performance of the rate $R = 0.467$ GLDPC code $(6000, 2800)$ parameterised in Table 2.2, using different maximum number of iterations, when communicating over AWGN channels.

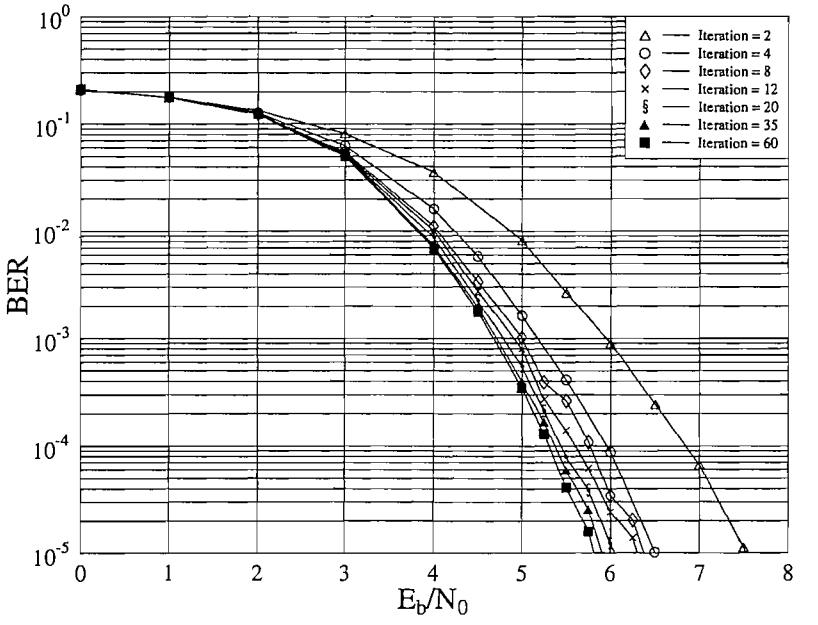


FIGURE 2.20: BER performance of the rate $R = 0.467$ GLDPC code $(300, 140)$ parameterised in Table 2.2, using different maximum number of iterations, when communicating over uncorrelated Rayleigh fading channels.

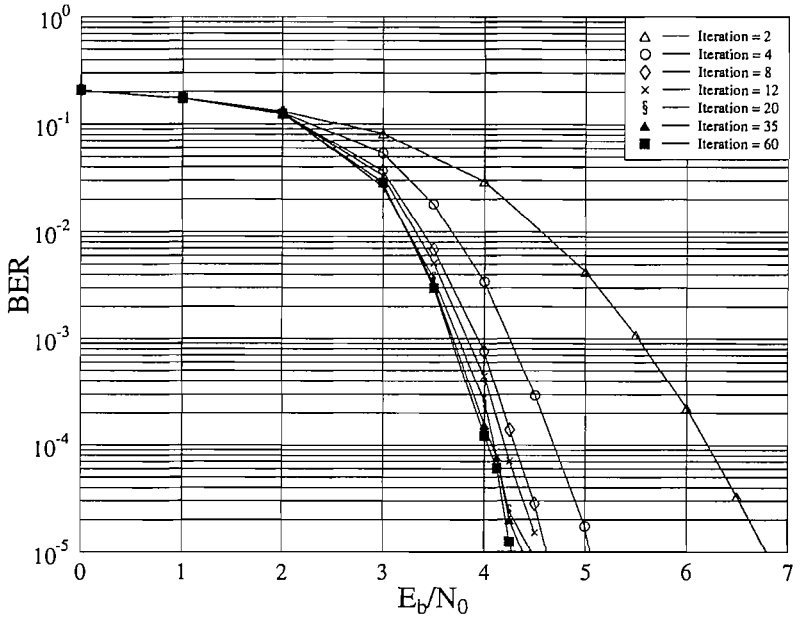


FIGURE 2.21: BER performance of the coding rate $R = 0.467$ GLDPC code $(1200, 560)$ parameterised in Table 2.2, using different maximum number of iterations, when communicating over uncorrelated Rayleigh fading channel.

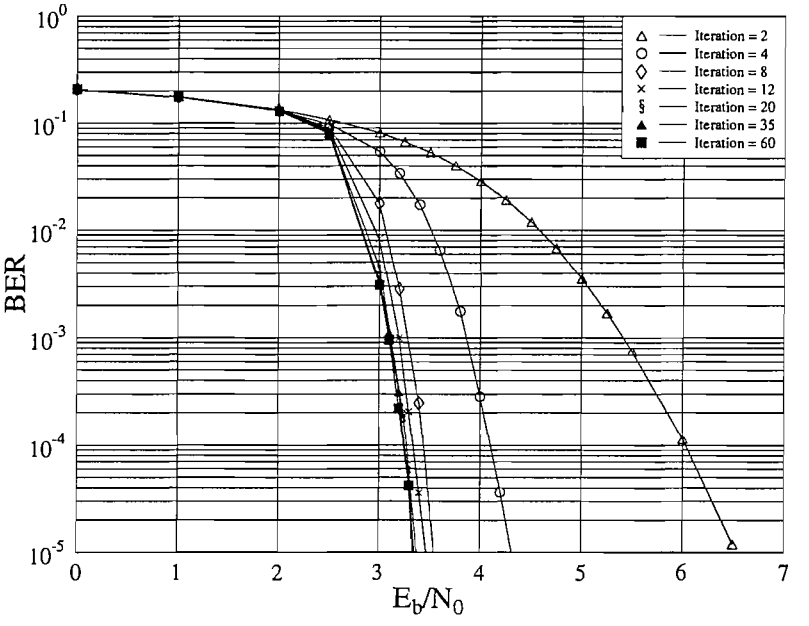


FIGURE 2.22: BER performance of the rate $R = 0.467$ GLDPC code $(6000, 2800)$ parameterised in Table 2.2, using different maximum number of iterations, when communicating over uncorrelated Rayleigh fading channels.

GLDPC code	Maximum number of iterations	Required E_b/N_0 (dB)	Coding gain (dB)	Iteration Efficiency(%)
uncoded	N/A	8.3983	0	0
(300,140)	2	3.7957	4.6026	81.7557
	4	3.3103	5.088	90.3778
	8	3.096	5.3023	94.1844
	12	3.0402	5.3581	95.1756
	20	2.9472	5.4511	96.8275
	35	2.8533	5.5450	98.4955
	60	2.7685	5.6297	100.000
(1200,560)	2	3.3293	5.069	78.1541
	4	2.2621	6.1362	94.6083
	8	2.1102	6.2881	96.9503
	12	2.0367	6.3616	98.0835
	20	1.9937	6.4046	98.7465
	35	1.9377	6.4606	99.6099
	60	1.9124	6.4859	100.000
(6000,2800)	2	3.2227	5.1756	73.5735
	4	1.9640	6.4343	91.4665
	8	1.5204	6.8779	97.7724
	12	1.4234	6.8779	97.7724
	20	1.3829	6.9749	99.1513
	35	1.3637	7.0154	99.7271
	60	1.3551	7.0346	100.000

TABLE 2.3: E_b/N_0 required by three different-length, rate $R = 0.467$ GLDPC codes parameterised in Table 2.2 for achieving a BER of 10^{-4}, when communicating over AWGN channels.

GLDPC code	Maximum number of iterations	Required E_b/N_0 (dB)	Coding gain (dB)	Iteration Efficiency(%)
uncoded	N/A	33.9781	0	0
(300,140)	2	6.8418	27.1363	94.6423
	4	5.9575	28.0206	97.7264
	8	5.7674	28.2107	98.3894
	12	5.6007	28.3774	98.9708
	20	5.4405	28.5376	99.5295
	35	5.3739	28.6042	99.7618
	60	5.3056	28.6725	100.000
(1200,560)	2	6.2082	27.7699	92.7460
	4	4.6897	29.2884	97.8174
	8	4.3515	29.6266	98.9470
	12	4.2018	29.7763	99.4469
	20	4.0992	29.8789	99.7896
	35	4.0773	29.9008	99.8627
	60	4.0362	29.9419	100.000
(6000,2800)	2	6.0182	27.9599	90.9833
	4	4.1016	29.8765	97.2201
	8	3.4934	30.4847	99.1992
	12	3.3409	30.6372	99.6954
	20	3.2774	30.7007	99.9021
	35	3.2567	30.7214	99.9694
	60	3.2473	30.7308	100.000

TABLE 2.4: E_b/N_0 required by three different-length, rate $R = 0.467$ GLDPC codes parameterised in Table 2.2 for achieving a BER of 10^{-4}, when communicating over Uncorrelated Rayleigh Fading (URF) channels.

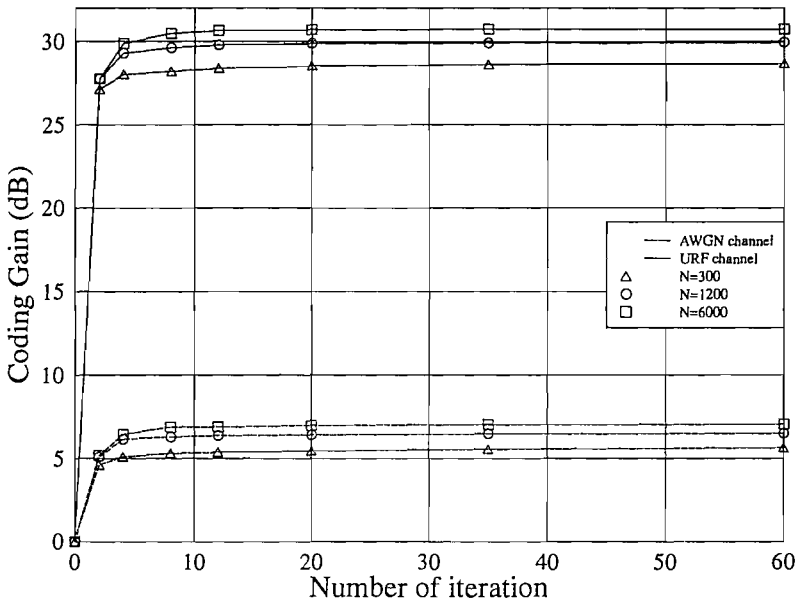


FIGURE 2.23: Coding gain achieved by three different-length, rate $R = 0.467$ GLDPC codes parameterized in Table 2.2, at a BER of 10^{-4}, when communicating over both AWGN and uncorrelated Rayleigh fading channels

2.7.2 Effect of the SISO Decoding Algorithm

Figures 2.24 - 2.29 show our comparisons between GLDPC codes using the MAP, Log-Map and Max-Log-MAP decoding algorithms described in Chapter 5 of [51]. The GLDPC codes characterised in Table 2.5 were simulated. In Figures 2.24 - 2.29, the "Log-MAP" curve refers to a decoder, which calculates the correction term of $f_c(\delta)$ in [51] by using a look-up table in conjunction with eight values of $f_c(\delta)$ stored, as described in [50], and hence introduces an approximation in the calculation of the LLRs. As Robertson *et al.* found [50], the look-up procedure for the values of the correction terms $f_c(\delta)$ imposes no significant degradation on the performance of the decoder. It can bee seen that, as expected, the MAP and the Log-MAP decoding algorithms attain identical performances.

It can also be seen from Figures 2.24 - 2.29 that the Max-Log-MAP algorithm imposes a slight performance degradation compared to the MAP and Log-MAP algorithms. As shown in Figures 2.24 - 2.29, under the same channel conditions and using the same constituent codes but different block-lengths, the higher the

Constituent code	Codeword Length(N)	Number of iterations	Decoding algorithm	Channel
Hamming (7,4)	301	20	Exact-MAP	AWGN/URF
	301	20	Log-MAP	AWGN/URF
	301	20	Max-Log-MAP	AWGN/URF
	3003	20	Exact-MAP	AWGN/URF
	3003	20	Log-MAP	AWGN/URF
	3003	20	Max-Log-MAP	AWGN/URF
Hamming (15,11)	300	20	Exact-MAP	AWGN/URF
	300	20	Log-MAP	AWGN/URF
	300	20	Max-Log-MAP	AWGN/URF
	3000	20	Exact-MAP	AWGN/URF
	3000	20	Log-MAP	AWGN/URF
	3000	20	Max-Log-MAP	AWGN/URF
Hamming (31,26)	1209	20	Exact-MAP	AWGN/URF
	1209	20	Log-MAP	AWGN/URF
	1209	20	Max-Log-MAP	AWGN/URF
	6014	20	Exact-MAP	AWGN/URF
	6014	20	Log-MAP	AWGN/URF
	6014	20	Max-Log-MAP	AWGN/URF

TABLE 2.5: Simulation parameters for six different $J = 2$-level GLDPC codes investigated, when communicating over both an AWGN channel and an URF channel using different decoding algorithms.

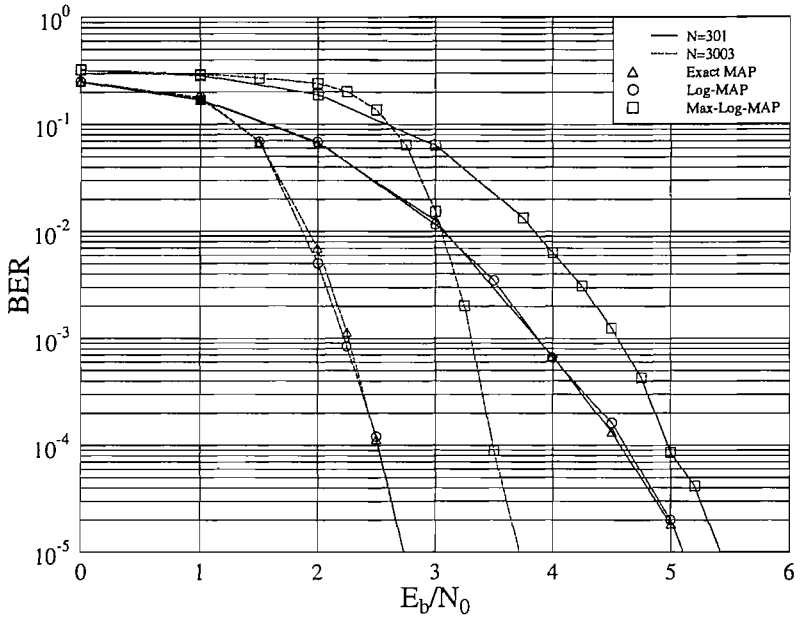


FIGURE 2.24: BER performance of two different-length, rate $R = 0.143$ GLDPC codes, (301,43) and (3003,429), parameterized in Table 2.5, while using different decoding algorithms for communicating over AWGN channels.

block-length, cause the larger the performance degradation imposed by the Max-Log-MAP algorithm compared to the MAP and Log-MAP algorithms. For example, observe in Figure 2.24, that at a BER of 10^{-4} the associated degradation is about 0.4 dB and 1 dB for the Max-Log-MAP algorithm in the context of GLDPC codes having block-lengths of 301 and 3003 bits, respectively. Furthermore, when comparing Figure 2.24 to 2.25, under AWGN channel conditions and at a similar block-length, the Max-Log-MAP algorithm inflicts larger degradations, when GLDPC codes having lower coding rates are used. For instance, at a BER of 10^{-4} and a block-length of 300 bits, the associated degradation is about 1 dB and 0.5 dB for the Max-Log-MAP algorithm applied by GLDPC codes having a rate of 0.143 and 0.467, respective. The same phenomenon can also be observed by comparing Figure 2.27 to Figure 2.28.

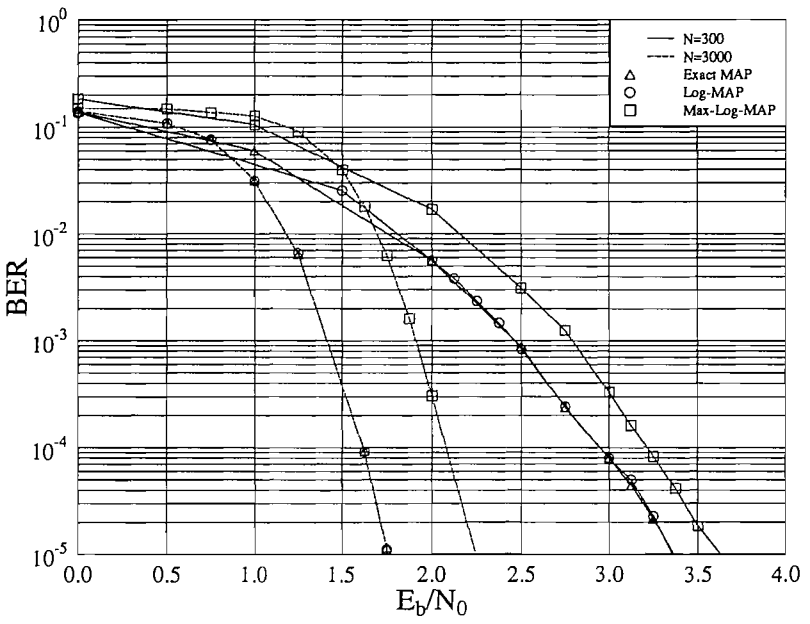


FIGURE 2.25: BER performance of two different-length, rate $R = 0.467$ GLDPC codes, (300,140) and (3000,1400), parameterized in Table 2.5, while using different decoding algorithms for communicating over AWGN channels.

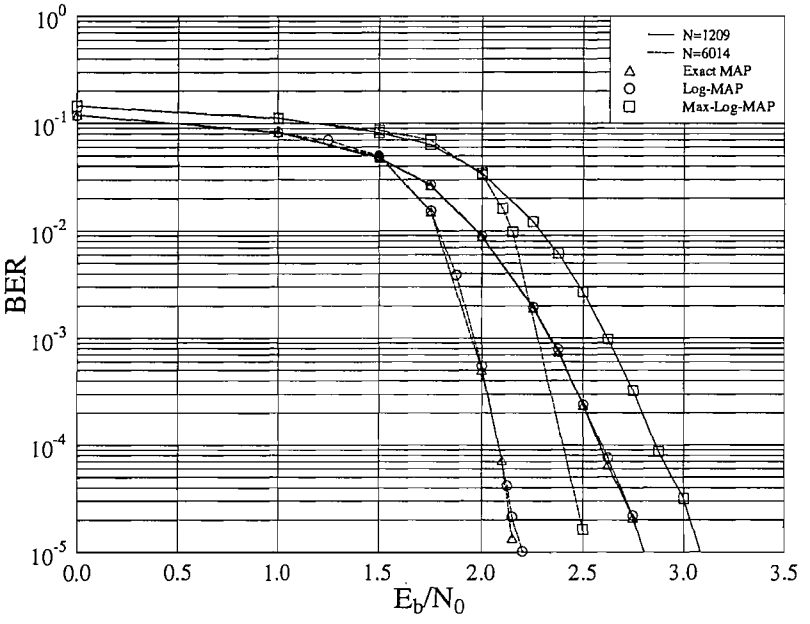


FIGURE 2.26: BER performance of two different-length, rate $R = 0.677$ GLDPC codes, (1209,819) and (6014,4074), parameterized in Table 2.5, while using different decoding algorithms for communicating over AWGN channels.

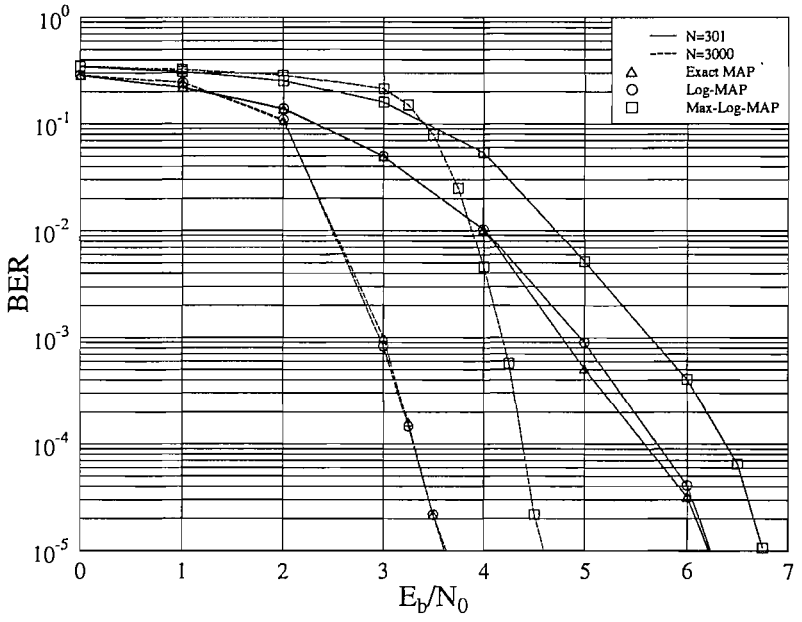


FIGURE 2.27: BER performance of two different-length, rate $R = 0.143$ GLDPC codes, (301,43) and (3003,429), parameterized in Table 2.5, while using different decoding algorithms for communicating over uncorrelated Rayleigh fading channels.

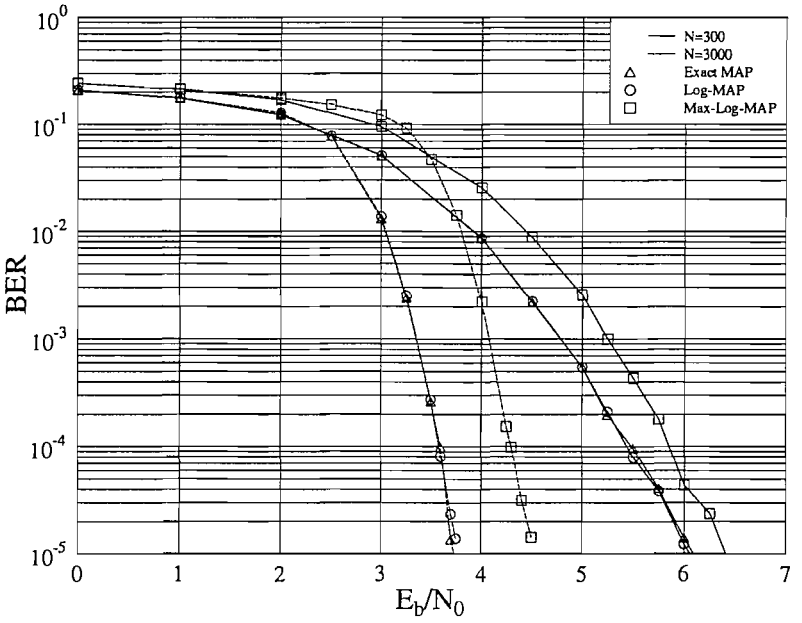


FIGURE 2.28: BER performance of two different-length, rate $R = 0.467$ GLDPC codes, (300,140) and (3000,1400), parameterized in Table 2.5, while using different decoding algorithms for communicating over uncorrelated Rayleigh fading channels.

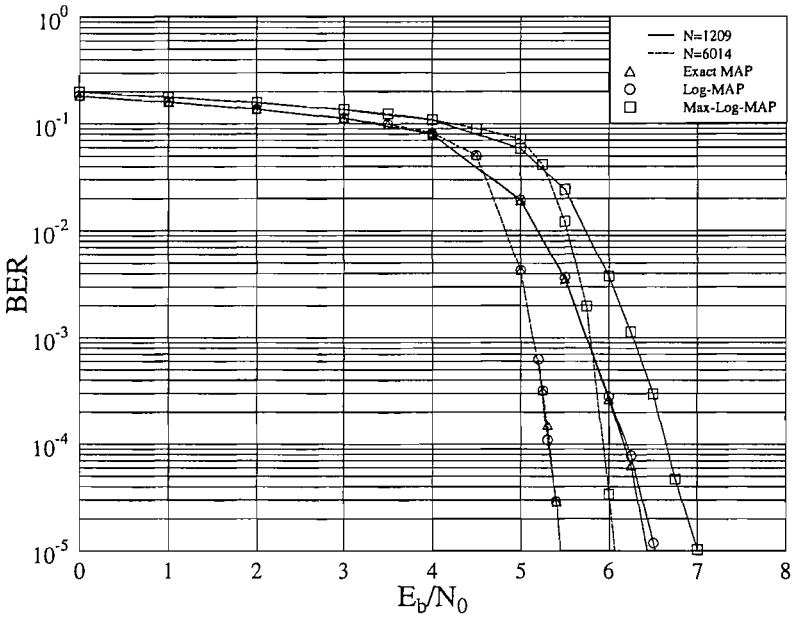


FIGURE 2.29: BER performance of two different-length, rate $R = 0.677$ GLDPC codes, (1209,819) and (6014,4074), parameterized in Table 2.5, while using different decoding algorithms for communicating over uncorrelated Rayleigh fading channels.

2.7.3 Effect of Different Coding Rates

In this section, the performance of various GLDPC codes will be evaluated at different coding rates. The simulation parameters of the GLDPC codes used are summarized in Table 2.6. The coding rate of the GLDPC codes is varied by changing the coding rates of constituent code. The relationship of the coding rate between the GLDPC code and its constituent codes was outlined in Section 2.2. As shown in Figures 2.30 and 2.31, the curves recorded for various coding rates

Constituent code	Codeword Length(N)	Coding rate R	Number of iterations	Decoding algorithm	Channel
Hamming (15,11)	6000	0.467	20	Log-MAP	AWGN /URF
Hamming (31,26)	6014	0.677	20	Log-MAP	AWGN /URF
Hamming (63,57)	5985	0.810	20	Log-MAP	AWGN /URF

TABLE 2.6: Simulation parameters for three different $J = 2$-level GLDPC codes investigated, when communicating over both an AWGN channel and an URF channel using different coding rates.

Constituent code	Codeword Length (N)	Coding rate R	Shannon limit (dB)	Required E_b/N_0 (dB)	Distance from the Shannon limit (dB)
Hamming (15,11)	6000	0.467	0.042	1.45	1.405
Hamming (31,26)	6014	0.677	1.125	2.2	1.075
Hamming (63,57)	5985	0.810	2.137	3.05	0.913

TABLE 2.7: Distance for the Shannon capacity limit for various GLDPC codes using different coding rates R, when communicating over AWGN channels.

ranging from 0.467 to 0.81 are aligned as expected and the performance of the GLDPC code having a rate of $R = 0.467$ is better than that of the other two GLDPC codes.

We also evaluated the AWGN channel's Shannon capacity limit [1] for each GLDPC codes using different constituent codes. In Table 2.7 and Table 2.8, we recorded the E_b/N_0 distance with respect to the Shannon capacity limit for the various rate GLDPC codes studied, when communicating over both AWGN and uncorrelated Rayleigh fading channels, respectively. For the GLDPC code using the (15,11) constituent Hamming code for communicating over AWGN channels, which has a coding rate of $R = 0.467$, the associated performance is about 1.405 dB away in terms of E_b/N_0 from the Shannon capacity limit at a BER of 10^{-5}. As we increase the coding rate to 0.81 by using the (63,57) constituent Hamming codes for communicating over AWGN channels, the performance curve is within about 0.913 dB of the Shannon limit, again, when viewed at a BER of 10^{-5}. By observing Table 2.7, as the coding rate R increases, the discrepancy between the associated performance curve and the Shannon limit is reduced in the context of the AWGN channel. By contrast, as shown in Table 2.8, as the coding rate increases, the distance for the Shannon limit increases when communicating over uncorrelated Rayleigh fading channels.

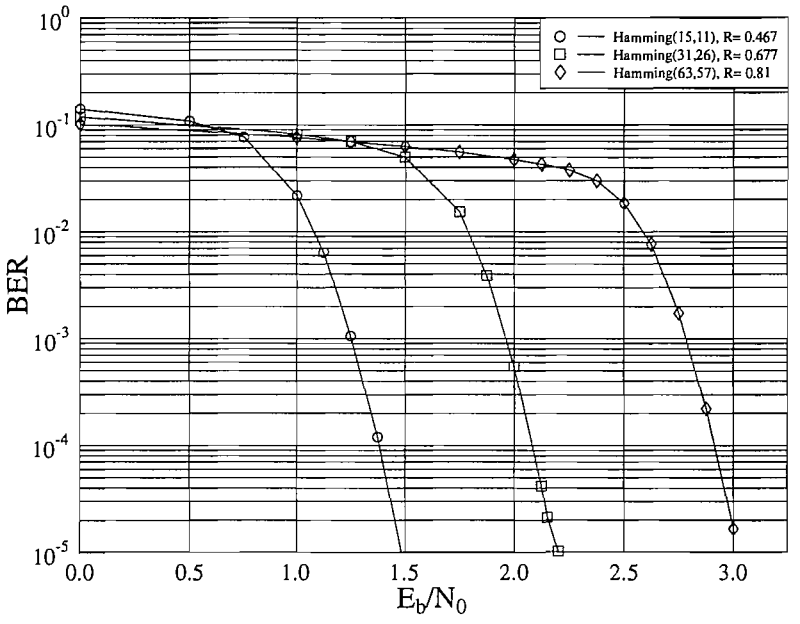


FIGURE 2.30: BER performance of the GLDPC codes parameterized in Table 2.6, using different constituent codes which result in different coding rates, when communicating over AWGN channels.

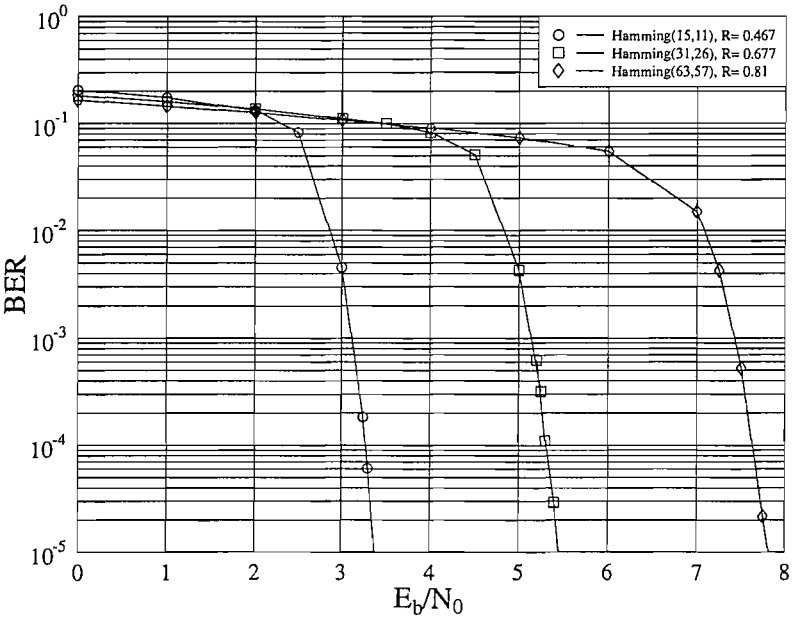


FIGURE 2.31: BER performance of the GLDPC codes parameterized in Table 2.6, using different constituent codes which result in different coding rates, when communicating over uncorrelated Rayleigh fading channels.

Constituent code	Codeword Length (N)	Coding rate R	Shannon limit (dB)	Required E_b/N_0 (dB)	Distance from the Shannon limit (dB)
Hamming (15,11)	6000	0.467	0.042	3.3	3.258
Hamming (31,26)	6014	0.677	1.125	5.4	4.275
Hamming (63,57)	5985	0.810	2.137	7.8	5.663

TABLE 2.8: Distance for the AWGN channel's Shannon capacity limit for GLDPC codes using different coding rates R, when communicating over uncorrelated Rayleigh fading channels.

2.7.4 Effect of Different Codeword Lengths

The GLDPC codes characterized in Table 2.9 were simulated when using different block-lengths. The corresponding simulation results are depicted in Figures 2.32 - 2.37. As we expected, the results show that the GLDPC codes having a higher block-length perform better, regardless of the coding rate or the channel condition. It has been shown in [31] that the average minimum distance of the binary GLDPC codes is a linear function of their length. As the block-length N decreases, the number of constituent codes in the GLDPC code becomes smaller. Thus the correlation between the super-code C^1 and C^2 increases. Since iterative decoding procedure involves passing soft-information between the GLDPC encoded symbol nodes as well as the constituent code nodes, but the bits between super-code C^1

Constituent code	Coding rate R	Codeword Length(N)	Number of iterations	Decoding algorithm	Channel
Hamming (7,4)	0.143	301 1204 3003 5999	20	Log-MAP	AWGN /URF
Hamming (15,11)	0.467	300 1200 3000 6000	20	Log-MAP	AWGN /URF
Hamming (31,26)	0.677	1209 3007 6014	20	Log-MAP	AWGN /URF

TABLE 2.9: Simulation parameters for the eleven different $J = 2$-level GLDPC codes investigated, when communicating over both an AWGN and URF channels using different block lengths.

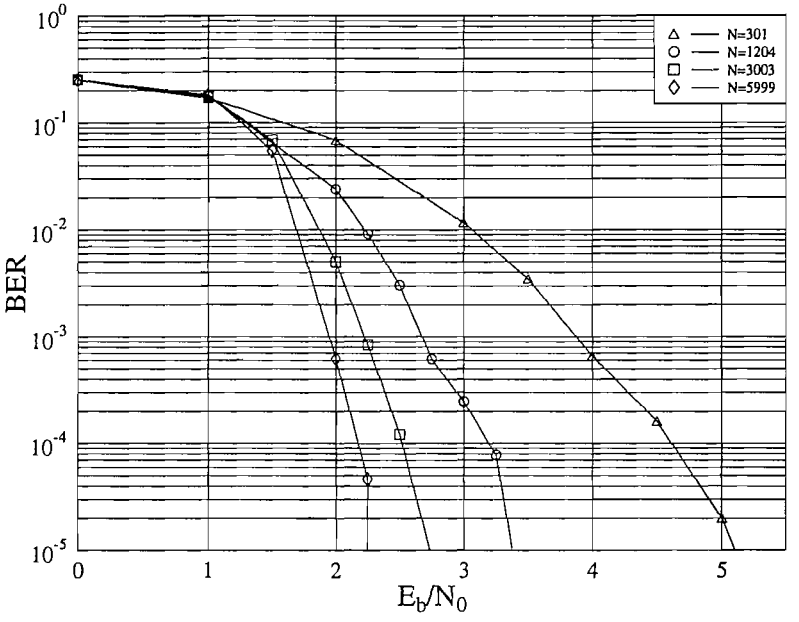


FIGURE 2.32: BER performance of the GLDPC codes employing (7,4) constituent Hamming codes, parameterized in Table 2.9, and using different block-lengths, for communicating over AWGN channels.

and C^2 become more dependent on each other, the iterative SISO decoder of the GLDPC code becomes less capable of correcting the errors.

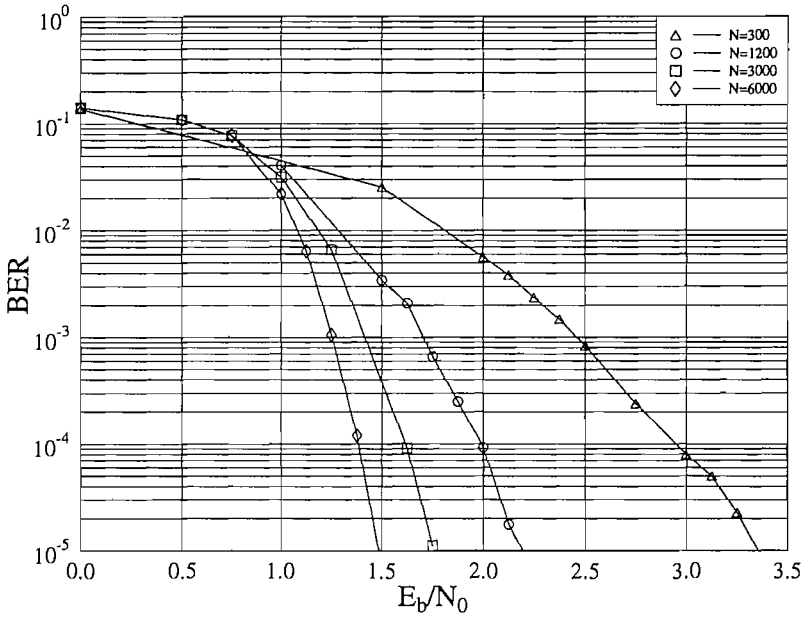


FIGURE 2.33: BER performance of the GLDPC codes employing (15,11) constituent Hamming codes, parameterized in Table 2.9, and using different block-lengths, for communicating over AWGN channels.

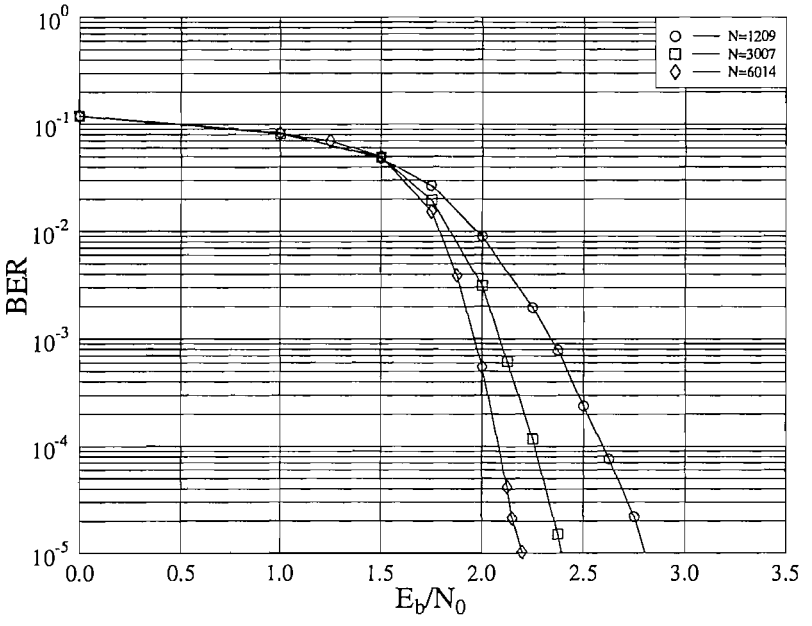


FIGURE 2.34: BER performance of the GLDPC codes employing (31,26) constituent Hamming codes, parameterized in Table 2.9, and using different block-lengths, for communicating over AWGN channels.

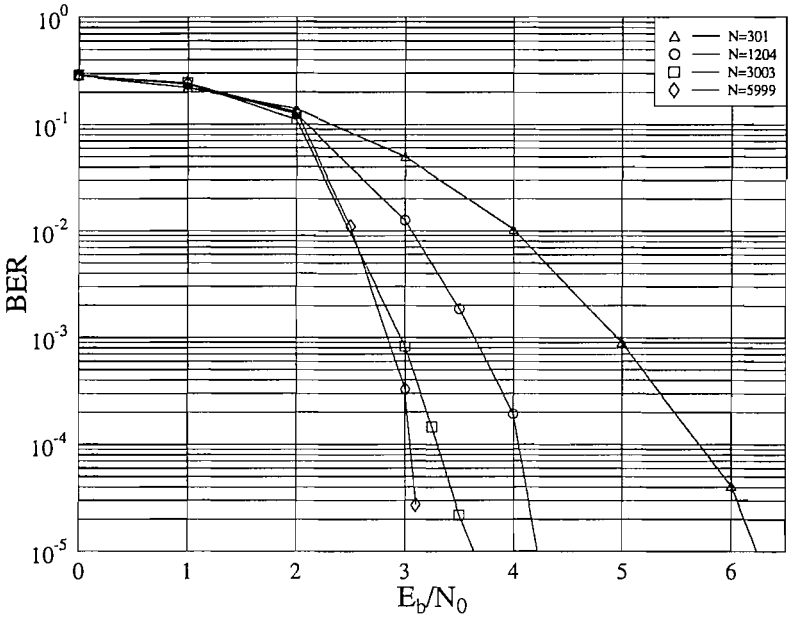


FIGURE 2.35: BER performance of the GLDPC codes employing (7,4) constituent Hamming codes, parameterized in Table 2.9, and using different block-lengths, for communicating over URF channels.

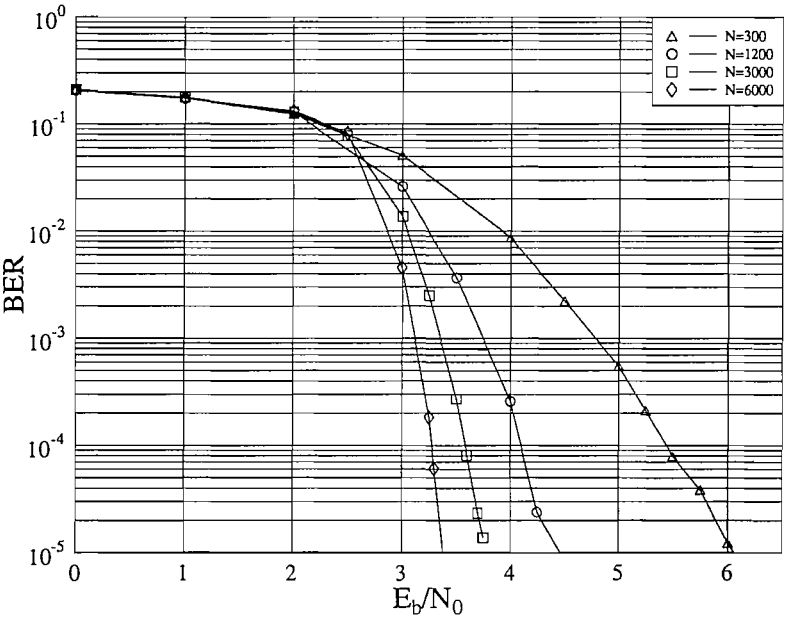


FIGURE 2.36: BER performance of the GLDPC codes employing (15,11) constituent Hamming codes, parameterized in Table 2.9, and using different block-lengths, for communicating over URF channels.

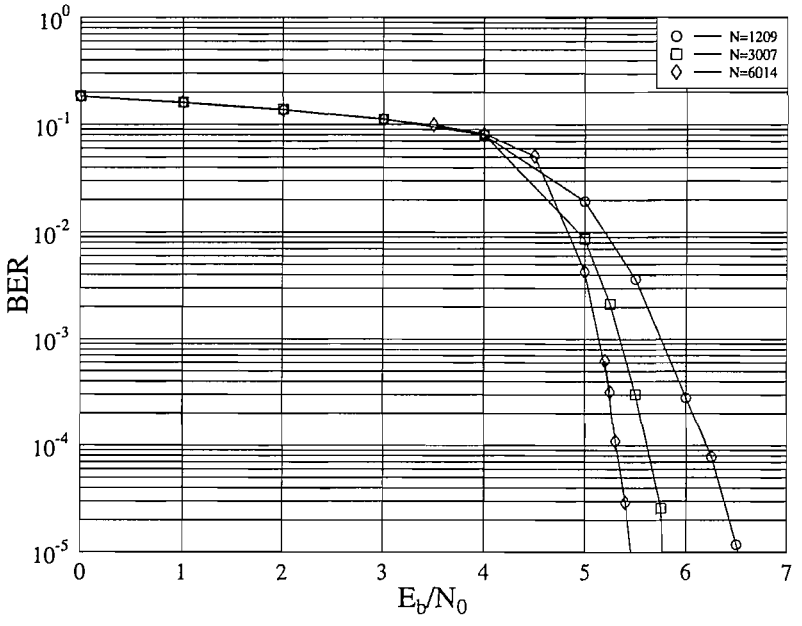


FIGURE 2.37: BER performance of the GLDPC codes employing (31,26) constituent Hamming codes, parameterized in Table 2.9, and using different block-lengths, for communicating over URF channels.

2.7.5 Shortened Constituent Hamming Codes

In this section, we will discuss various GLDPC codes using shortened Hamming codes as constituent codes. In the context of system design, if a GLDPC code of suitable coding rate cannot be found, it may be desirable to use a shortened constituent block code to meet the system's requirements. For example, for designing an attractive multi-level coding [52, 53], component codes having specific coding rates are required. Therefore, we are interested in the characteristics of GLDPC

Constituent code			Coding	Codeword	Decoding	Channel
Type	n	k	rate R	Length(N)	algorithm	
Hamming	15	11	0.467	3000	Log-MAP	AWGN/URF
Hamming	31	26	0.677	3007	Log-MAP	AWGN/URF
Shortened Hamming	25	20	0.6	3000	Log-MAP	AWGN/URF
Shortened Hamming	20	15	0.5	3000	Log-MAP	AWGN/URF
Shortened Hamming	19	14	0.474	3002	Log-MAP	AWGN/URF
Shortened Hamming	37	31	0.676	2997	Log-MAP	AWGN/URF

TABLE 2.10: Simulation parameters for the six various $J = 2$-level GLDPC codes investigated, when communicating over both an AWGN channel and URF channels using primitive and shortened Hamming codes as constituent codes, when the maximum number of iterations is 20.

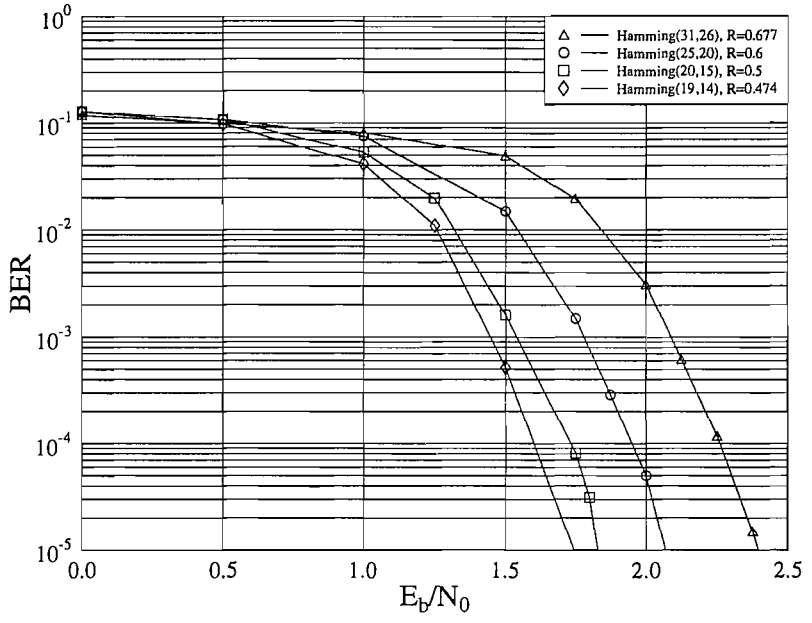


FIGURE 2.38: BER performance of the GLDPC codes parameterized in Table 2.10, using different shortened constituent Hamming codes generated from the (31,26) Hamming code, when communicating over AWGN channels.

codes using shortened block code as their constituent codes. The GLDPC codes characterized in Table 2.10 were studied, when using different primitive and shortened Hamming codes as constituent codes. Figure 2.38 and Figure 2.39 shows the achievable BER performance of the GLDPC code using shortened Hamming codes generated from the (31,26) Hamming code, when communicating over both AWGN and URF channels. Since the constituent codes were shortened, the coding rate of the GLDPC code was reduced.

Figure 2.40 and Figure 2.41 characterize the attainable performance of a range of similar coding rate GLDPC codes using primitive and shortened Hamming codes as their constituent codes. As shown in the figures, the GLDPC codes constructed from shortened Hamming codes has the same performance as their counterparts using primitive Hamming codes. Generally, it is more beneficial to choose primitive Hamming or BCH codes as constituent codes, if both primitive and shortened Hamming or BCH codes are capable of constructing the same coding rate GLDPC code, since the constituent code having shorter block-length exhibits a lower complexity.

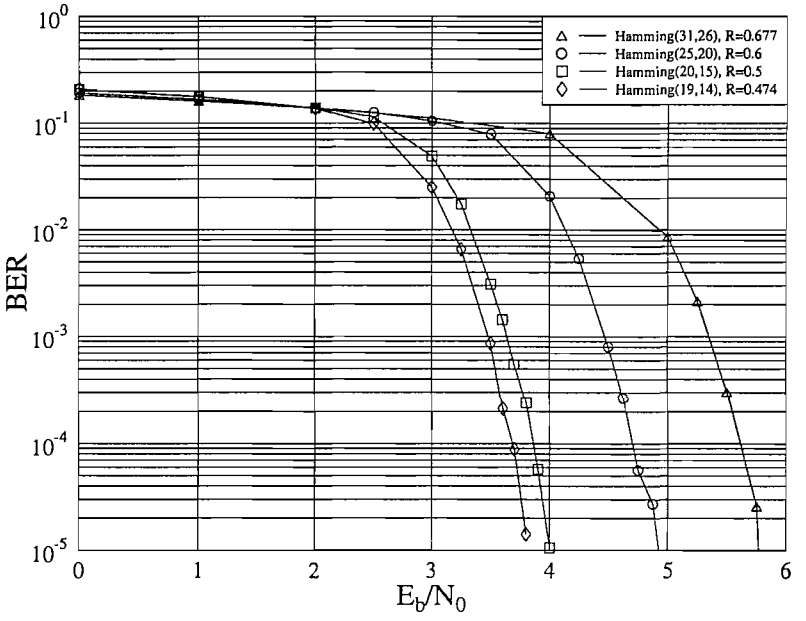


FIGURE 2.39: BER performance of the GLDPC codes parameterized in Table 2.10, using different shortened constituent Hamming codes generated from the (31,26) Hamming code, when communicating over URF channels.

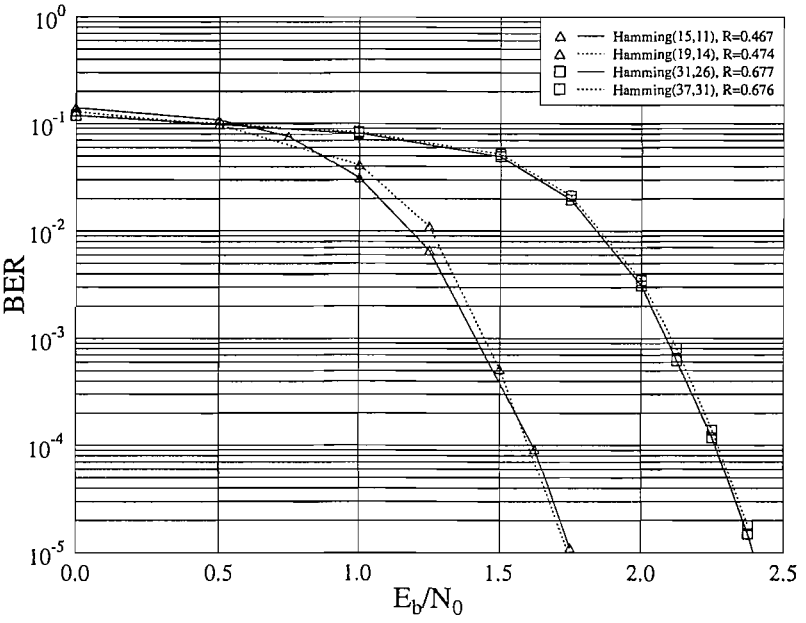


FIGURE 2.40: BER performance of the similar rate GLDPC codes parameterized in Table 2.10, using primitive (15,11) constituent Hamming code and shortened constituent Hamming codes generated from the (31,26) Hamming code, when communicating over AWGN channels.

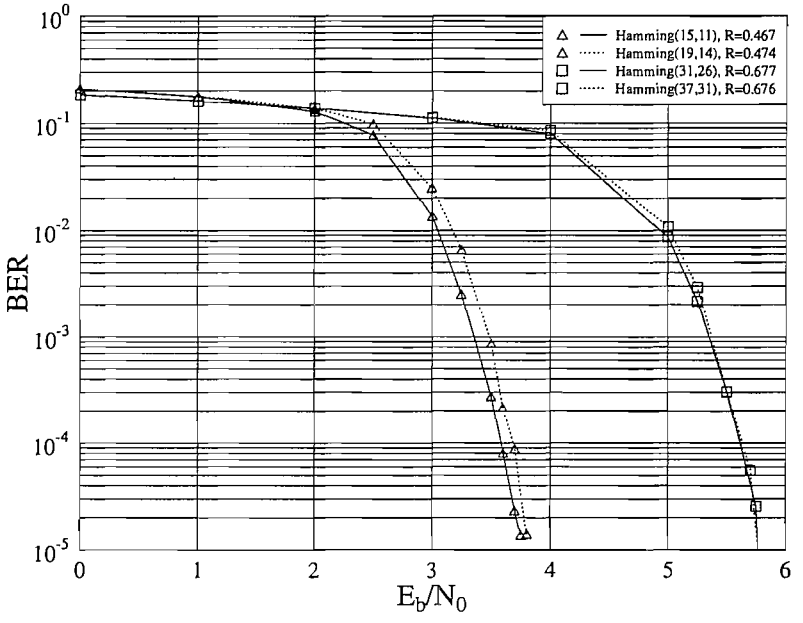


FIGURE 2.41: BER performance of the similar rate GLDPC codes parameterized in Table 2.10, using primitive (15,11) constituent Hamming code and shortened constituent Hamming codes generated from the (31,26) Hamming code, when communicating over URF channels.

2.8 Conclusions

In this chapter, the family of GLDPC codes was introduced. In Section 2.1 the structure of GLDPC codes was described both by the PCM and by the concept of Tanner graphs [17]. We commenced by providing an example of classic LDPC [12] codes, which was then extended to GLDPC codes, since GLDPC codes may be regarded as an evolution of classic LDPC codes [12]. Section 2.2 described the coding rate calculation of GLDPC codes, which is naturally related to the choice of the constituent code. It has been shown in [31, 32, 33, 34] that GLDPC codes based on binary Hamming or BCH constituent codes are 'asymptotically good', even if we have $J = 2$ and that iterative decoding is very simple to implement in this case. Furthermore, as the coding rate decrease linearly with J, the case of J = 2-level GLDPC codes is of particular interest, as we argued in Section 2.3. Section 2.5 gave an introduction to iterative SISO GLDPC decoding algorithms. Moreover, a detailed example of the binary J = 2-level GLDPC code using the Log-MAP decoding algorithm was provided in Section 2.6. In Section 2.7, the performance of binary GLDPC codes was characterized in various scenarios. The effect of increasing the GLDPC decoder's complexity, i.e. the number of iterations

was studied in Section 2.7.1, and as seen in Figures 2.17 - 2.22, using eight to twelve iterations strikes an attractive a compromise between the attainable coding gain and the associated decoding complexity. The performance of the MAP, the Log-MAP and the Max-Log-MAP decoding algorithms used by the constituent decoder was characterized in Figures 2.24 - 2.29 of Section 2.7.2 and it was found that the MAP and the Log-MAP decoding algorithms attain identical performances, while the Max-Log-MAP algorithm exhibits a slight performance degradation compared to the MAP and Log-MAP algorithms. The amount of the degradation is affected by both the block-length and the coding rate. The performance of GLDPC codes having different coding rates was evaluated in Figures 2.30 - 2.31 of Section 2.7.3. In the specific scenarios investigated in Figure 2.30, as the coding rate R increased, the distance from Shannon limit was reduced in the context of AWGN channel for coding rate of 0.467, 0.677 and 0.81. The opposite was found in Figure 2.31 for the uncorrelated Rayleigh fading channel. The effects of increasing the GLDPC codes' coded block-length was recorded in Figures 2.32 - 2.37 of Section 2.7.4 and demonstrated that GLDPC codes having a higher block-length perform better, regardless of the coding rate or the channel. Finally, in Section 2.7.5 the characteristics of GLDPC codes using shortened Hamming code as constituent code were depicted that the GLDPC codes with shortened Hamming codes have the same performance, as GLDPC codes invoking primitive Hamming codes. Generally, it is more beneficial to choose primitive Hamming or BCH codes as constituent codes if both primitive and shortened Hamming or BCH code are capable of constructing the same coding rate GLDPC codes, since the constituent codes having shorter block-length exhibit lower complexity.

Chapter 3

Symbol-Flipping Based Decoding of Nonbinary GLDPC Codes

3.1 Introduction

In this chapter, we investigate the attainable performance of symbol-based hard decision decoding algorithm designed for GLDPC codes employing nonbinary constituent codes. The benefit of using purely symbol-based channel codes combined with symbol-based QAM modulations [54] transmitting a high number of bits per symbol is the associated high throughput, which is achieved without any bandwidth expansion. Furthermore, these non-binary systems are considered robust against short error bursts confined to a single symbol.

By contrast, binary constituent codes have more often been used for constructing GLDPC codes [38, 40, 41, 42, 43, 44]. Moreover, perhaps the best known classic codes are the maximum-minimum-distance nonbinary RS codes [7], which are used in numerous standards, such as the Digital Audio Broadcast (DAB) and Digital Video Broadcast (DVB) schemes or in Compact Disc (CD) players. It is therefore worth investigating, how RS codes behave, when they are embedded in GLDPC coding schemes. A particular further advantage of GLDPC codes is that their iterative decoding is based on the decoding of modest-complexity constituent codes, hence the total decoding complexity may be expected to be low.

In this treatise a symbol-flipping algorithm is designed for symbol-based hard decision decoding, which may be considered to be an extension of the bit-flipping

algorithm of [12, 42, 55]. Again, the first bit flipping scheme was originally proposed by Gallager for LDPC codes [12]. Based on the appealing conceptual and implementational simplicity of the bit-flipping algorithm, the Weighted Bit-Flipping (WBF) algorithm was developed in [55] for the sake of achieving an improved performance by exploiting some bit-reliability information, which results in an attractive tradeoff between the achievable performance and the decoding complexity imposed [55]. The concept of bit flipping algorithms using votes [42] was generalised for employment in GLDPC codes using binary Hamming constituent codes and hence it was termed as Weighted Bit Flip Voting (WBFV) [42]. Based on the philosophy of the WBFV algorithm developed for binary Hamming-code based GLDPC codes, here we propose a symbol flipping algorithm for employment in nonbinary GLDPC codes. Similar to the WBFV algorithm of [42], the error correcting capability of the constituent codes is exploited for more accurately determining the position of the least reliable symbols. However, in the context of the symbol-flipping algorithm, not only the error positions, but also the $(q-1)$ legitimate error magnitudes have to be evaluated. This can be achieved, if the classic algebraic decoders of the nonbinary constituent codes of the GLDPC codes are applied. In each decoding iteration, the least reliable symbols are corrected according to the error magnitude provided by the algebraic decoder of the nonbinary constituent code. We will provide simulation results to demonstrate that symbol-flipping algorithms can be successfully employed for the decoding of nonbinary GLDPC codes. We will also demonstrate that the proposed coding scheme results in an improved error rate performance in comparison to binary GLDPC codes using the WBFV decoding algorithm of [42], when communicating over both AWGN and uncorrelated Rayleigh fading channels.

The remainder of this chapter is organized as follows. In Section 3.2, a brief review of the WBFV algorithm [42] designed for binary Hamming based GLDPC codes is provided. Although here a different vote regime is required, since different constituent algebraic decoders are used. Hence the generation of the votes will be discussed in Section 3.3. Furthermore, in Section 3.4 we will discuss how to design the so-called vote weights for the symbol-flipping algorithm. A step-by-step description of the symbol-flipping algorithm designed for nonbinary GLDPC codes is summarized in Section 3.5. Our simulation results are presented in Section 3.6 and, finally, our conclusions are offered in Section 3.7

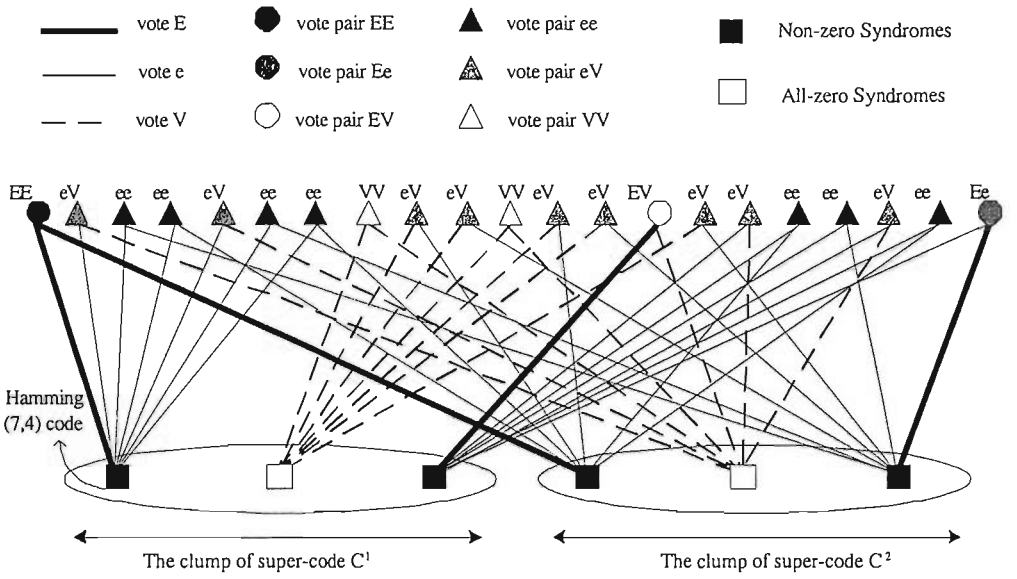


FIGURE 3.1: Example of the voting process for the $J = 2$-level binary GLDPC $(21, 3)$ code using the constituent Hamming$(7, 4)$ codes in one iteration. The 21 GLDPC encoded bit nodes are seen in the upper part, while the $J \cdot N/n = 2 \cdot 21/7 = 6$ constituent code nodes are portrayed in the lower part of the figure. Similarly to Figure 2.13, the corresponding $J = 2$-level GLDPC decoder is represented by the super-code clumps C^1 and C^2, each having $L = N/n = 21/7 = 3$ constituent Hamming (7,4) decoders

3.2 Weighted Bit Flip Voting algorithm for Binary Hamming Code-Based GLDPC codes [42]

In this section we will briefly describe the WBFV algorithm of [42] designed for binary Hamming-based GLDPC codes. Recall that the PCM of GLDPC codes was outlined in Figure 2.4, the corresponding bipartite graph was seen in Figure 2.6 and the associated SISO decoder was portrayed in Figure 2.13. Given the definition of the GLDPC codes in terms of J number of interleaved super-codes [31, 32, 33], the decoding philosophy of the $J = 2$-level GLDPC (N, K) code of Figure 2.13 is similar to that of a product code, where every symbol of the GLDPC (N, K) codeword is decoded by two constituent decoders, which belong to two independent super-codes [31, 32, 33]. Accordingly, the WBFV [42] algorithm decodes the GLDPC codes using an iterative method which is different from the MAP SISO decoder of Figure 2.13.

More specifically, a hard decision decoder is applied by each constituent code

$C_0(n,k)$ which generates n votes for the n bits it covers. For the specific $J = 2$-level GLDPC constructions employed, the information bits will belong to the (n,k) binary Hamming constituent codes in the super-code C_1 and C_2, respectively, as seen earlier in Figure 2.11 of Chapter 2. Moreover, in contrast to the MAP SISO decoder of Figure 2.13, in the WBFV algorithm, bits' votes are generated by the hard decision decoders (HDDs) of constituent codes in super-code C_1 and C_2 as shown in Fig 3.1. Accordingly, a vote pair will be produced for each bit by the two HDDs. Note that for the sake of plausible and straightforward explanation, the GLDPC code considered in Figure 3.1 was constructed without avoiding the short cycles of length 4. For binary algebraic Hamming decoders, there are only two possible decoding outcomes [42]:

- All-zero syndromes: this implies the presence of a valid (V) codeword, hence all the n constituent Hamming code bits are labelled by the character V.
- Non-zero syndromes: Non-zero syndrome implies an invalid codeword, thus the indicated error position will be labelled with a vote E, while the votes e are assigned to all the other bits of the (n,k) code.

The magnitude of a vote indicates, how reliable a Hamming constituent code node considers the current symbol's value to be. To elaborate a little further, it was proposed by Hirst and Honary [42] that the various votes V, e, E should be given numerical values so that the votes arriving from the $J = 2$ decoders for all the $N = 21$ bits, which are either VV, eV, EV, ee, eE or EE, may be ranked in terms of their reliability according to the sum of the $J = 2$ constituent votes [42]. It was aslo suggested in [42] that using the weight of $V = 0$, $e = 1$ and $E = 2$ is capable of producing the lowest possible BERs, where having higher weights represents a lower reliability. Table 3.1 shows the weights of the various vote pairs adopted from Hirst and Honary [42] for the WBFV algorithm. In summary, the WBFV decoding algorithm of Figure 3.1 may be formulated as follows [42]:

1. Apply the HDDs of the binary Hamming constituent codes in super-code C_1 and C_2, respectively.
2. Compute the $J = 2$ vote weights for each of the N bits based on the type of vote pair.

$V = 0, e = 1, E = 2$					
Vote pair	EE	Ee	EV, ee	eV	VV
Vote weight	4	3	2	1	0

TABLE 3.1: Vote Weights for WBFV algorithm for the binary Hamming-based GLDPC codes [42].

3. Rank the bits according to their reliability based on their vote weights, as seen in Table 3.1.

4. Flip all bits of lowest reliability, i.e. the bits having the maximum vote weight.

5. Repeat steps (1) to (4). This process of bit flipping continues, until the highest-weight vote pair becomes VV or the affordable maximum number of iterations is reached.

3.3 The Vote Pairs for Symbol-Flipping Based Non-Binary GLDPC Decoding

Our symbol-flipping based decoding algorithm is extended from the concept of the WBFV algorithm designed for the binary Hamming-code based GLDPC codes [42] for the GLDPC codes using either nonbinary BCH or RS constituent codes. Moreover, since different constituent algebraic decoders are used, different vote pairs have to be generated which will discuss in this section. Furthermore, in the symbol-flipping based decoding algorithm not only the error positions, but also the error magnitudes defined over $GF(q)$ have to be evaluated, where the algebraic decoders of the nonbinary constituent codes provide both the error positions as well as the error magnitudes.

Fig. 3.2 shows the symbols' vote generation process for the GLDPC $(21, 9)$ code constructed over $GF(8)$ using the constituent code RS $(7, 5)$. Note that for the sake of plausible and straightforward explanation, the GLDPC code considered in Fig. 3.2 was constructed without avoiding the short cycles of length 4. The votes concerning the specific values of the symbols are generated by the Peterson-Gorenstein-Zierler (PGZ) or the Berlekamp-Massey (BM) [51] hard decision decoders (HDDs) of the RS constituent codes in the super-code C^1 and C^2. In

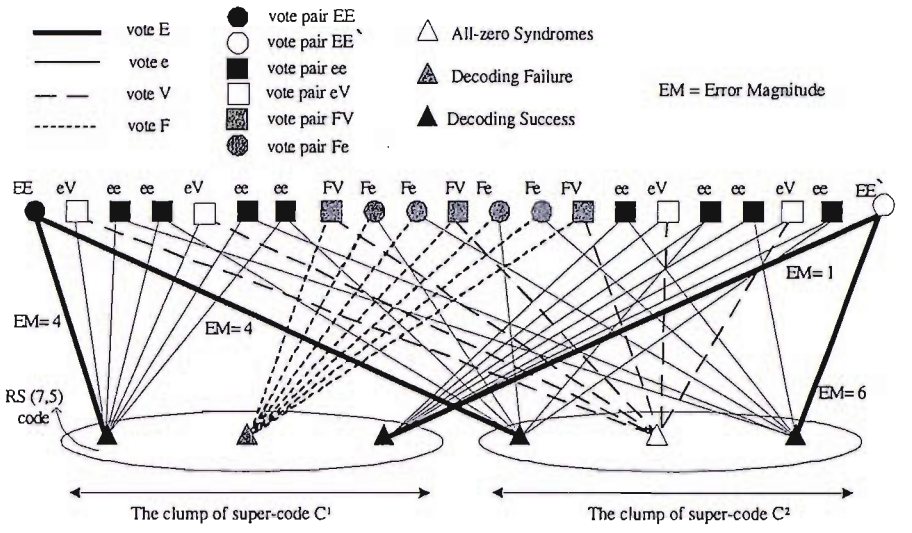


FIGURE 3.2: Example of the voting process for the $J = 2$-level GLDPC $(21, 9)$ code constructed over $GF(8)$ using the constituent RS $(7, 5)$ codes in one iteration. The 21 GLDPC encoded symbol nodes are seen in the upper part, while the $J \cdot N/n = 2 \cdot 21/7 = 6$ constituent code nodes are portrayed in the lower part of the figure. Similarly to Figure 2.13, the corresponding $J = 2$-level GLDPC decoder is represented by the super-code clumps C^1 and C^2, each having $L = N/n = 21/7 = 3$ constituent RS (7,5) decoders

nonbinary HDDs, owing to the limited error-correction capability of the RS code, a *decoding failure* occurs, when the corrupted codeword is not within the so-called decoding sphere of a valid codeword. Hence, for the nonbinary algebraic RS constituent decoders used in the symbol-flipping algorithm, there are three possible decoding scenarios, which are also featured in Fig. 3.2 and discussed below:

- All-zero syndromes: this implies the presence of a valid (V) codeword, hence all the n constituent RS code symbols are labelled by the character V.

- Non-zero syndromes and decoding success: this indicates the presence of an invalid but correctable received word. Since successful decoding took place, the corresponding error positions are labelled with E indicating that the symbols are in error. By contrast, the symbol label e is assigned to all other symbols, which were deemed to be the correct symbols in an erroneously received but correctable codeword.

- Non-zero syndromes and decoding failure: no error positions were identified owing to decoding failure (F) , therefore no corrective action may be carried out and no useful information may be gleaned from this decoder. Hence all the n RS code symbols of the codeword are labelled by F.

As shown in Figure 3.2, after the votes have been assigned, each GLDPC encoded symbol node will be assigned a vote pair: either EE, EF, Ee, EV, FF, Fe, FV, ee, eV, or VV. Moreover, a vote pair EE' in Figure 3.2 indicates that different GF(q) error magnitudes were suggested by the $J = 2$ constituent decoders. In this case, we will randomly opt for the error magnitude suggested by one of the two constituent decoders.

In contrast to the binary Hamming (7,4) constituent code based GLDPC codes of Section 3.2, in the symbol-flipping based decoding of nonbinary GLDPC codes, we have the extra vote label F, which results in more legitimate vote pairs and in the specific vote pair EE', which accrues from different estimated GF(q) error magnitudes. Therefore, the reliability of each vote pair is unknown in symbol-flipping based decoding algorithms nonetheless. Figure 3.3 constitutes a useful indicator of the reliability of vote pairs, which are the probabilities that a symbol is in error conditioned on its received vote pair. The eleven conditional probabilities for the eleven possible vote pairs are plotted in Figure 3.3, which were evaluated by simulation after a single iteration using the GLDPC (300,140) code constructed over $GF(16)$ using the constituent RS(15,11) code. We then corrupted each GF(q) symbol with a certain probability to any of the legitimate GF(q) values. It can be seen that for a given symbol error rate (SER) p the following reliability ordering may be inferred: $VV > eV > FV > EV > ee > Fe > Ee > FF > EF > EE' > EE$. It may be argued further that the ordering of the vote reliability i.e. obeys $V > e > F > E$ or $V > e = F > E$.

3.4 The Design of the Vote Weight

Similarly to the binary Hamming (7,4) coded scenario of Section 3.2, the various votes V, F, e, E are given numerical values called vote weights so that the votes arriving from the $J = 2$ decoders for all the N GLDPC symbols, which are either EE, EF, Ee, EV, FF, Fe, FV, ee, eV, or VV, may be ranked in terms of their reliability according to the sum of the $J = 2$ constituent votes. In this section, we will discuss how to design the vote weights for the symbol-flipping based decoding algorithm. The conditional probability evaluation of Figure 3.3 used in Section 3.3 for vote pair ordering is only valid for the first decoding iteration. Therefore, in practice, the vote weights must be optimized by simulation as a function of the iteration index. We define furthermore a vote rule as a set of vote weights and

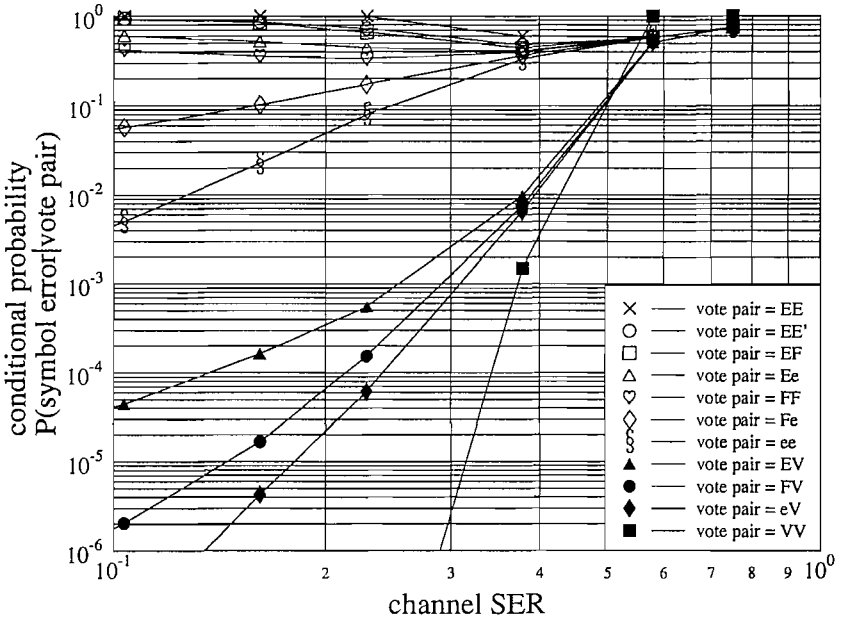


FIGURE 3.3: Conditional probabilities for the vote pairs EE, EE', EF, Ee, EV, FF, Fe, FV, ee, eV, and VV, which are evaluated from simulation after a single iteration using the GLDPC (300,140) code constructed over $GF(16)$ using the RS (15,11) constituent code.

the resultant vote pair ordering. As before in Section 3.2 we use larger values to indicate unreliable symbols and smaller values to indicate more reliable symbols.

For the PGZ or BM [51] HDDs of the RS constituent codes, we assume that the vote weights are ordered according to $V < e < F < E$, $V < F < e < E$ or $V < e = F < E$. Based on our simulations similar to those reported in Figure 3.3 but not detailed here, we propose seven different vote rules for the symbol-flipping based decoding algorithm, which are summarized in Table 3.2. According to each rules, the weight of the vote pair EE' is smaller than that of the vote pair EE, but larger than that of any of the other vote pairs. Rule 1 - Rule 3 of Table 3.2 are designed for satisfying $V < e < F < E$. As seen in Table 3.2, the weights of all vote pairs ordered according to Rule 1 are different, while according to Rule 2 and Rule 3, some vote pairs share the same weight. To elaborate a little further, more vote pairs share the same weight according to Rule 3, than in Rule 2. By contrast, Rule 5 - Rule 7 are designed for satisfying $V < F < e < E$. In Rule 7, the weight of each vote pair is different, and more vote pairs in Rule 5 share the same weights than in Rule 6. Finally, in Rule 4, the weights of vote F and vote e are the same.

Vote Rule				Vote Pair Ordering										
Rule 1				EE	EE'	EF	Ee	FF	Fe	ee	EV	FV	eV	VV
E	F	e	V	6	5.5	5	4.75	4	3.75	3.5	3	2	1.75	0
3	2	1.75	0											
Rule 2				EE	EE'	EF	Ee	FF	Fe	ee, EV		FV	eV	VV
E	F	e	V	6	5.5	5	4.5	4	3.5	3		2	1.5	0
3	2	1.5	0											
Rule 3				EE	EE'	EF	Ee, FF		Fe, EV		ee, FV		eV	VV
E	F	e	V	6	5.5	5	4		3		2		1	0
3	2	1	0											
Rule 4				EE	EE'	EF,Ee		FF, Fe, ee, EV				FV, eV		VV
E	F	e	V	6	5.5	4.5		3				1.5		0
3	1.5	1.5	0											
Rule 5				EE	EE'	Ee	EF, ee		Fe, EV		FF, eV		FV	VV
E	F	e	V	6	5.5	5	4		3		2		1	0
3	1	2	0											
Rule 6				EE	EE'	Ee	EF	ee	Fe	FF, EV		eV	FV	VV
E	F	e	V	6	5.5	5	4.5	4	3.5	3		2	1.5	0
3	1.5	2	0											
Rule 7				EE	EE'	Ee	EF	ee	Fe	FF	EV	eV	FV	VV
E	F	e	V	6	5.5	5	4.75	4	3.75	3.5	3	2	1.75	0
3	1.75	2	0											

TABLE 3.2: Vote pair weights and their ordering for the seven vote rules designed based on simulations similar to those characterized in Figure 3.3.

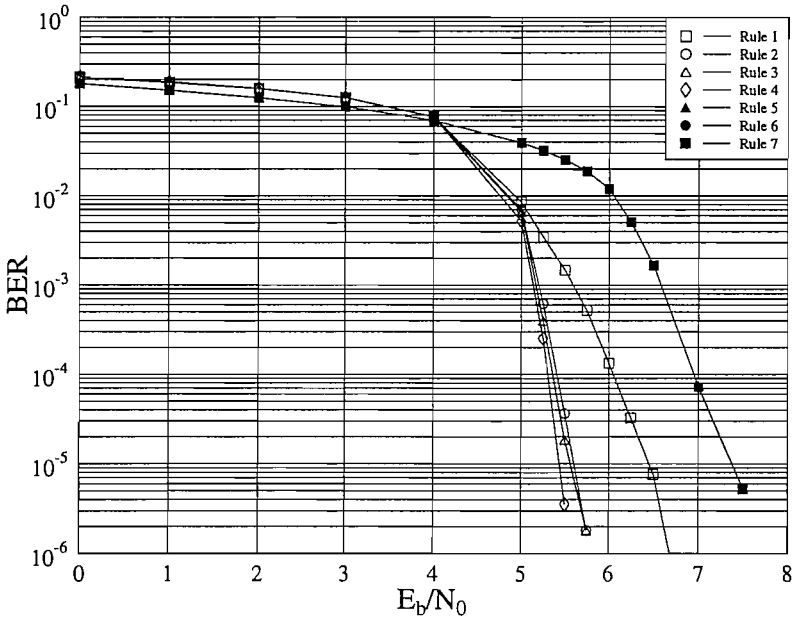


FIGURE 3.4: BER versus E_b/N_0 performance of the **coding rate** $R = 0.467$ GLDPC (4005,1869) code using **RS(15,11), GF(16)** constituent codes for transmission over an AWGN channel according to the weighting rules 1-7 of Table 3.2.

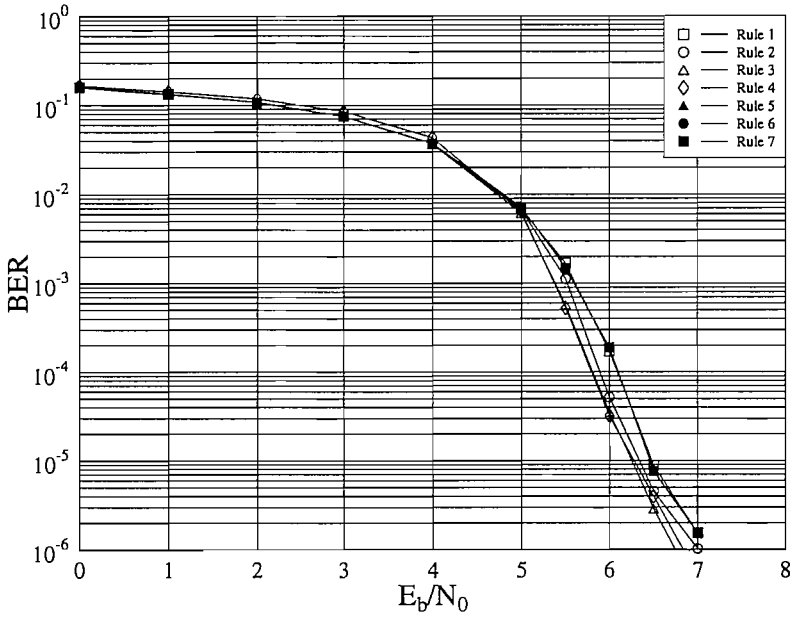


FIGURE 3.5: BER versus E_b/N_0 performance of the **coding rate** $R = 0.73$ GLDPC (4005,2937) code using **RS(15,13), GF(16)** constituent codes for transmission over an AWGN channel according to the weighting rules 1-7 of Table 3.2.

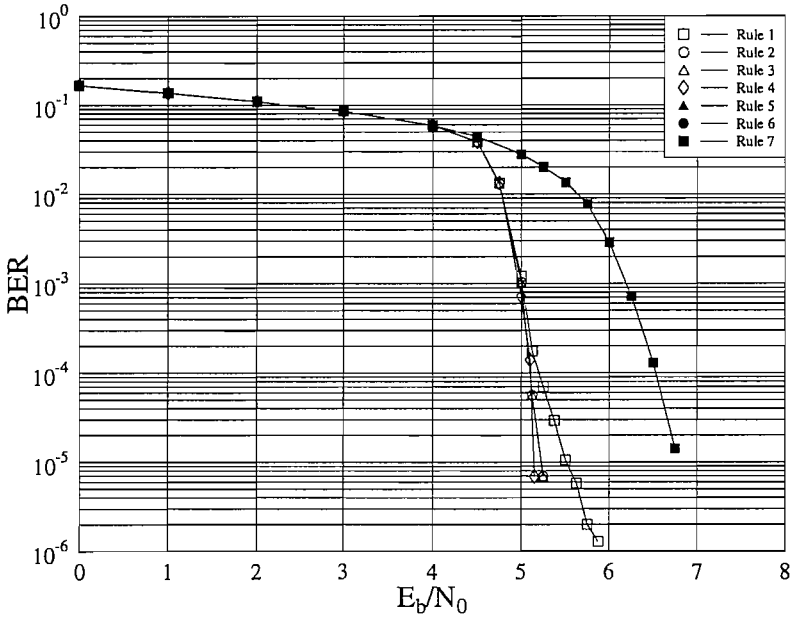


FIGURE 3.6: BER versus E_b/N_0 performance of the **coding rate** $R = 0.484$ GLDPC (1612,728) code using **RS(31,23), GF(32)** constituent codes for transmission over an AWGN channel according to the weighting rules 1-7 of Table 3.2.

Page 59 missing

Page 60 missing

$V = 0$, $e = F = 1.5$, $E = 3$						
Vote pair	EE	EE'	EF,Ee	FF, Fe, ee, EV	FV, eV	VV
Vote weight	6	5.5	4.5	3	1.5	0

TABLE 3.3: Vote weights for the symbol-based decoding of nonbinary GLDPC codes.

Using constituent RS (15,11) code over GF(16) with coding rate R=0.467			
GLDPC (N, K)	Channel	Decoding algorithm	Number of iterations
(300,140)	AWGN/URF	SF	2, 4, 5, 10, 20, 35, 60
(1005,469)	AWGN/URF	SF	2, 4, 5, 10, 20, 35, 60
(4005,1869)	AWGN/URF	SF	2, 4, 5, 10, 20, 35, 60

TABLE 3.4: Simulation parameters for three different-length J = 2-level GLDPC codes constructed over GF (16), when communicating over an AWGN channel and an uncorrelated Rayleigh fading (URF) channel using different **number of iterations.**

3.6 Simulation Results

3.6.1 Effect of the Number of Iterations

We will use three different-length J = 2-level GLDPC codes constructed over GF(16) and having a coding rate of R = 0.467 for demonstrating the achievable performance improvement upon using an increased number of decoding iterations. The detailed simulation parameters are listed in Table 3.4. The decoding algorithm shown in Table 3.4 is the Symbol-Flipping (SF) based decoding algorithm. Both AWGN and Uncorrelated Rayleigh Fading (URF) channels were applied.

It may be observed in Figure 3.8 that for the (300,140) GLDPC code the maximum number of iterations required is less than 10, when transmitting over an AWGN channel. Similarly, for the (1005,469) GLDPC code characterized in Figure 3.9 and for the (4005,1869) GLDPC code evaluated in Figure 3.10, no further improvements may be attained, when the number of iterations becomes higher than I =20. We can also observe Figures 3.8 - 3.10 that when the code's blocklength is increased, more iterations are necessary to eliminate the erroneous symbols in the codeword. Furthermore, the number of useful decoding iterations gracefully increases with N. Similarly, the same phenomenon is observed happen in the uncorrelated Rayleigh fading channel scenarios, as shown in Figures 3.11 -Figure 3.13. We can see that setting the number of iterations to I =20 may be deemed sufficiently high for fully exploiting the error-correction power of the decoder, when

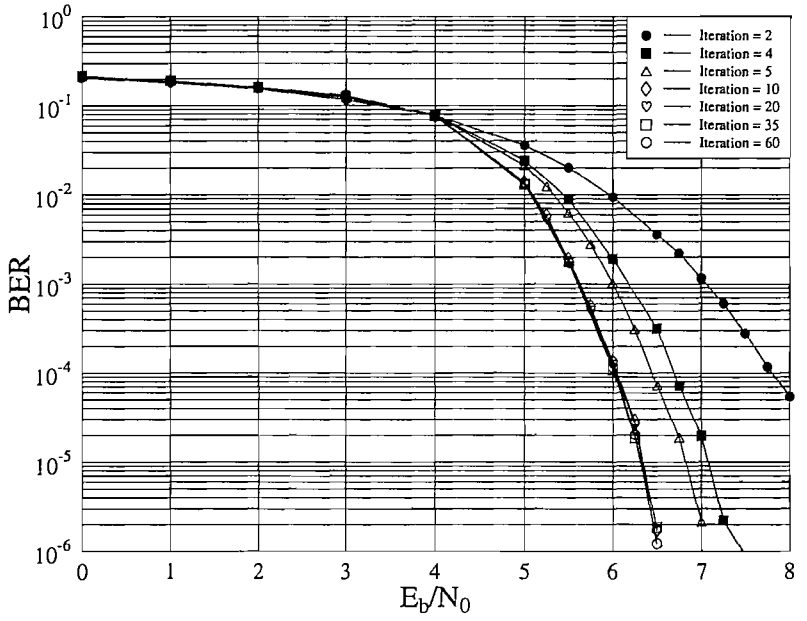


FIGURE 3.8: BER performance of the coding rate $R = 0.467$ **GLDPC code** $(300, 140)$ parameterized in Table 3.4, using different number of iterations, when communicating over an **AWGN** channel.

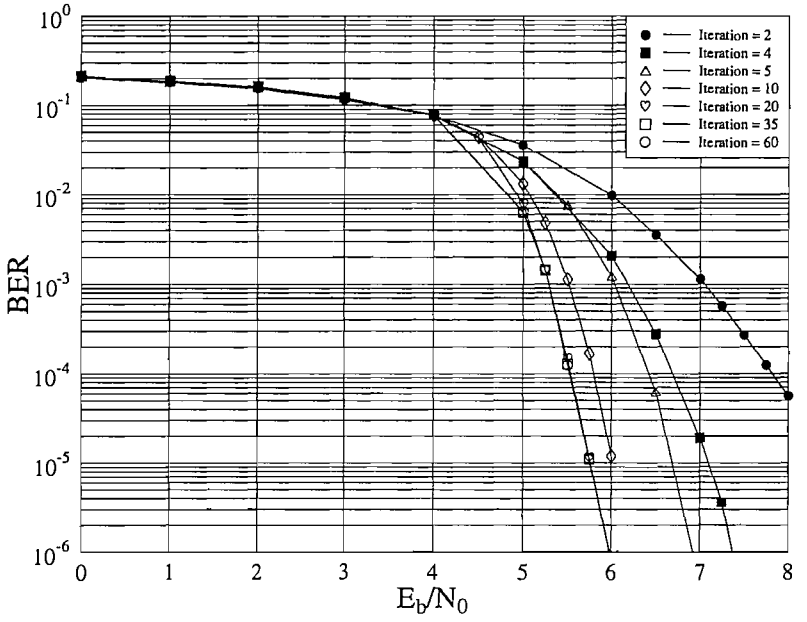


FIGURE 3.9: BER performance of the coding rate $R = 0.467$ **GLDPC code** $(1005, 469)$ parameterized in Table 3.4, using different number of iterations, when communicating over an **AWGN** channel.

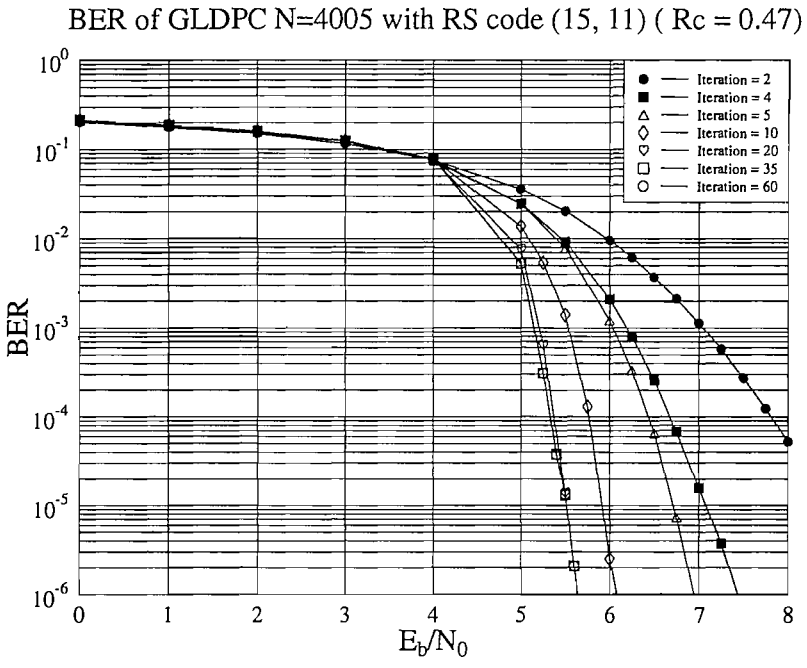


FIGURE 3.10: BER performance of the coding rate $R = 0.467$ **GLDPC code** $(4005, 1869)$ parameterized in Table 3.4, using different number of iterations, when communicating over an **AWGN** channel.

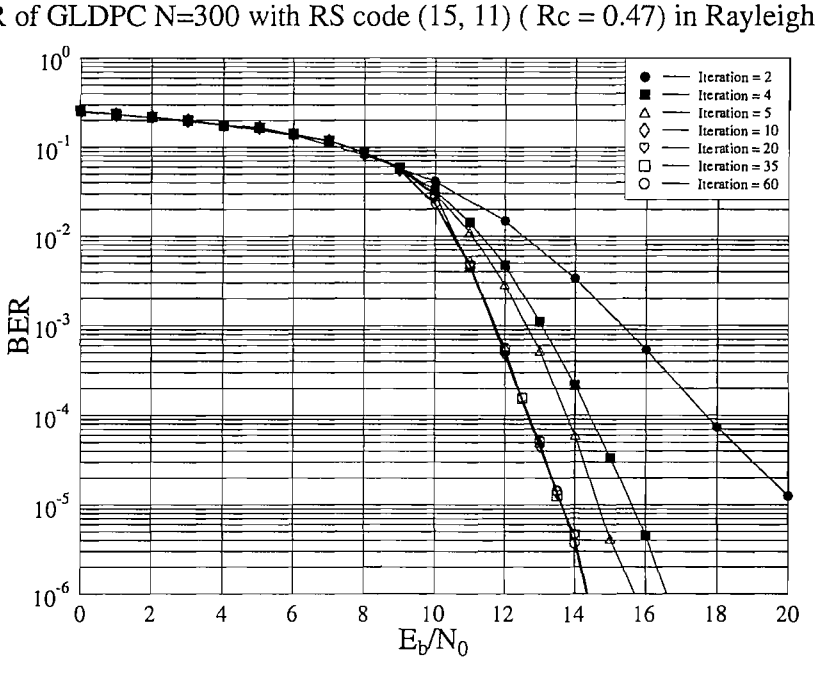


FIGURE 3.11: BER performance of the coding rate $R = 0.467$ **GLDPC code** $(300, 140)$ parameterized in Table 3.4, using different number of iterations, when communicating over an **URF** channel.

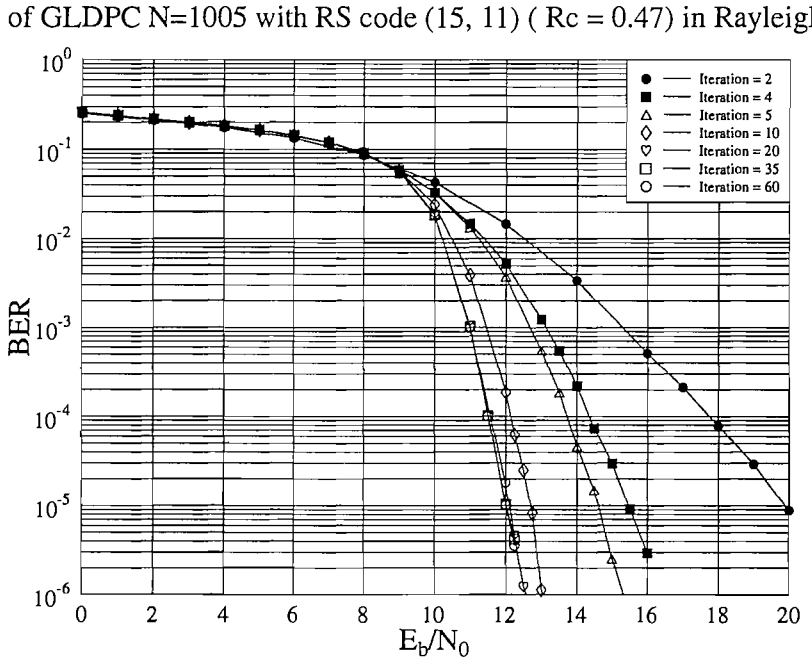


FIGURE 3.12: BER performance of the coding rate $R = 0.467$ **GLDPC code** $(1005, 469)$ parameterized in Table 3.4, using different number of iterations, when communicating over an **URF** channel.

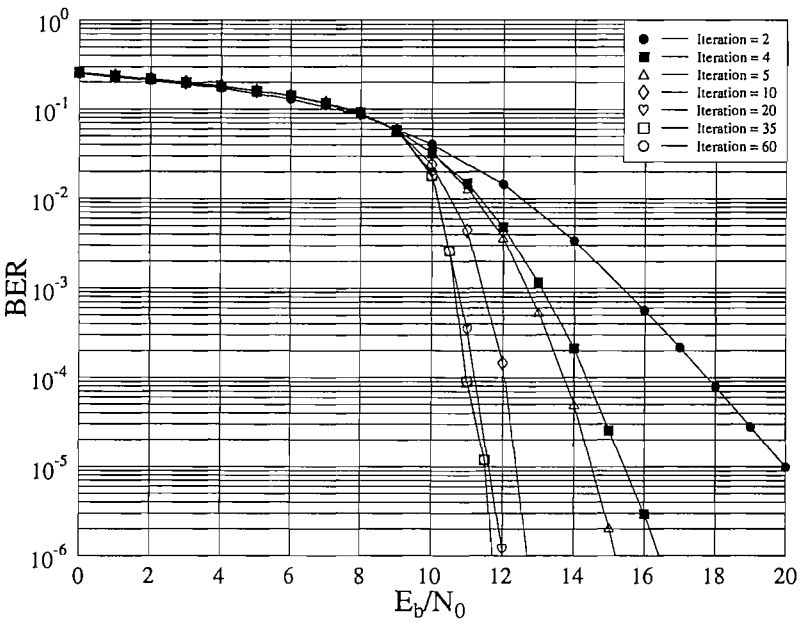


FIGURE 3.13: BER performance of the coding rate $R = 0.467$ **GLDPC code** $(4005, 1869)$ parameterized in Table 3.4, using different number of iterations, when communicating over an **URF** channel.

the block-length is less than 4000 symbols (16000 bits). Therefore, in our future investigations, we opted for setting the number of iterations to 20, when the block-length is less than 16000 bits.

3.6.2 Effects of the GLDPC Block-length

As seen in [42], the block-length of the binary GLDPC code is important, when using WBFV decoding algorithm owing to the associated increased minimum distance of longer codes. In this subsection, we will investigate the associated performance trends of nonbinary GLDPC codes, when the block-length of the GLDPC code is increased. The corresponding simulation parameters are listed in Table 3.5.

As shown in Figures 3.14 - 3.17, the achievable BER performance imposed upon increasing the block-length N, when communicating over both AWGN and URF channels. Figures 3.18 and 3.19 were plotted for further quantifying the relationship between the coding gain and block-length. As depicted in Figures 3.18 and 3.19, the coding gain increases relatively rapidly for the block-lengths between $N = 1000$ and then increases more slowly, as the the block-length exceeds this value, when communicating over both AWGN and URF channels.

GLDPC (N, K)	Constituent code	Coding rate R	Decoding algorithm	Number of iterations	Channel
(300,140) (510,238) (1005,469) (2010,938) (3000,1400) (4005,1869)	RS(15,11)	0.467	SF	20	AWGN /URF
(300,220) (510,374) (1005,737) (2010,1474) (3000,2200) (4005,2937)	RS(15,13)	0.73	SF	20	AWGN /URF

TABLE 3.5: Simulation parameters for $J = 2$-level GLDPC codes constructed over GF (16), when communicating over an AWGN channel and an URF channel using different **codeword lengths**.

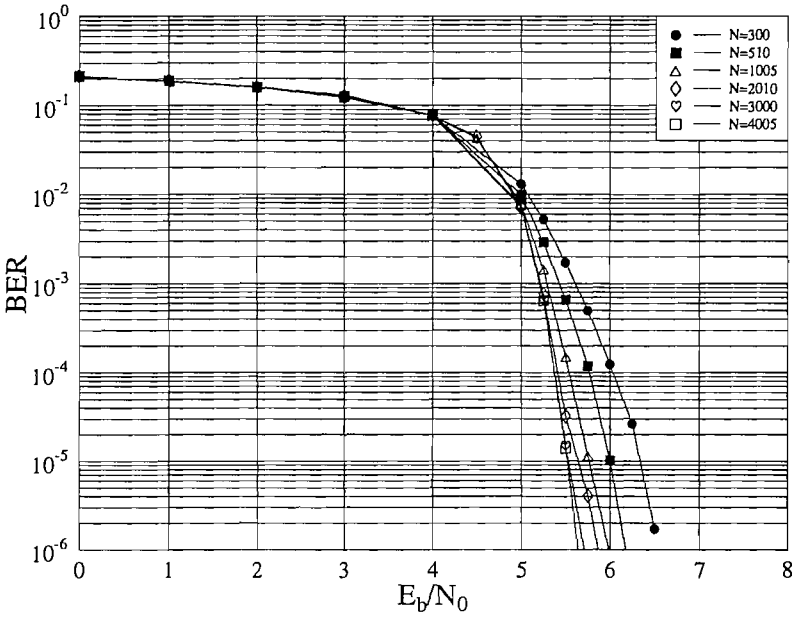


FIGURE 3.14: BER performance of the coding rate $R = 0.467$ GLDPC (300,140), (510,238), (1005,469), (2010,938), (3000,1400), (4005,1869) codes using **RS(15,11)** constituent codes parameterized in Table 3.5, and different block-lengths, when communicating over an **AWGN** channel.

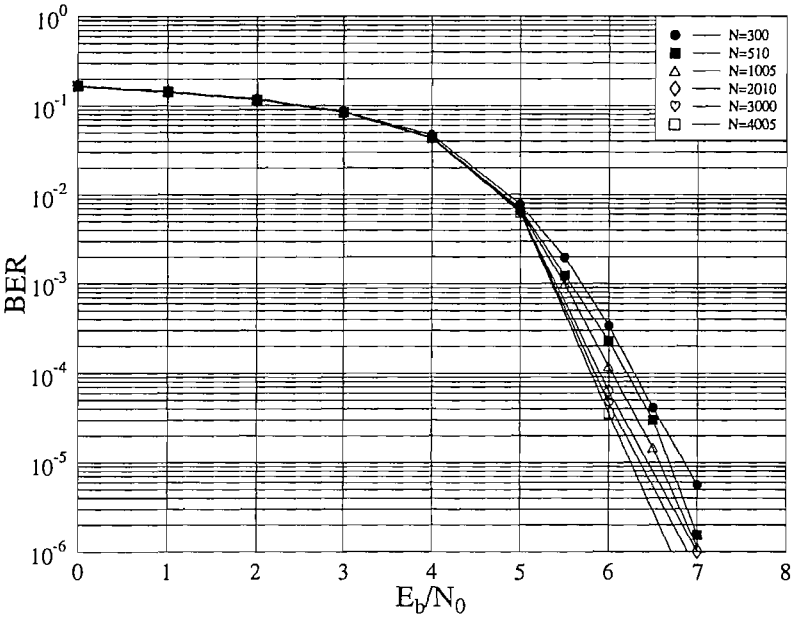


FIGURE 3.15: BER performance of the coding rate $R = 0.73$ GLDPC (300,220), (510,374), (1005,737), (2010,1474), (3000,2200), (4005,2937) codes using **RS(15,13)** constituent codes parameterized in Table 3.5, and different block-lengths, when communicating over an **AWGN** channel.

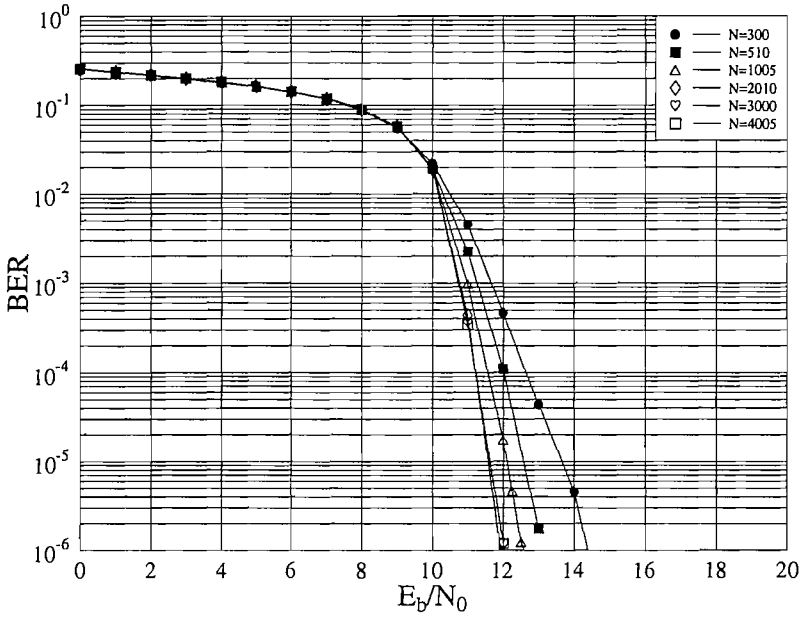


FIGURE 3.16: BER performance of the coding rate $R = 0.467$ GLDPC (300,140), (510,238), (1005,469), (2010,938), (3000,1400), (4005,1869) codes using **RS(15,11)** constituent codes parameterized in Table 3.5, and different block-lengths, when communicating over an **URF** channel.

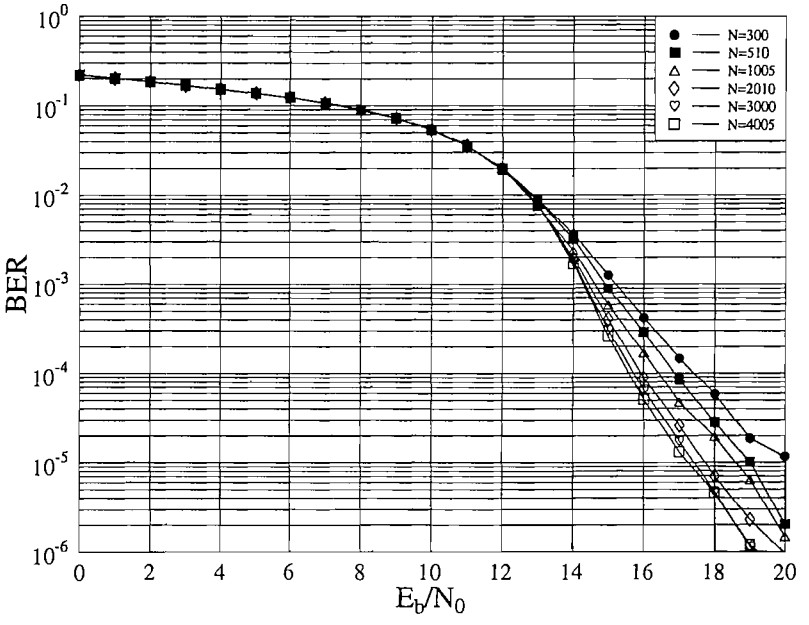


FIGURE 3.17: BER performance of the coding rate $R = 0.73$ GLDPC (300,220), (510,374), (1005,737), (2010,1474), (3000,2200), (4005,2937) codes using **RS(15,13)** constituent codes parameterized in Table 3.5, and different block-lengths, when communicating over an **URF** channel.

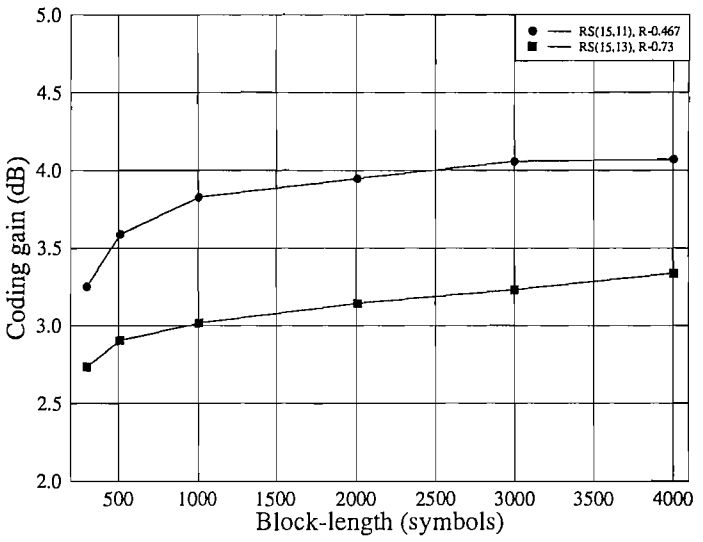


FIGURE 3.18: Coding gain achieved by the rate $R = 0.467$ GLDPC (300,140), (510,238), (1005,469), (2010,938), (3000,1400), (4005,1869) codes and rate $R = 0.73$ GLDPC (300,220), (510,374), (1005,737), (2010,1474), (3000,2200), (4005,2937) codes having different lengths and parameterized in Table 3.5, at a BER of 10^{-5}, when communicating over an **AWGN** channel.

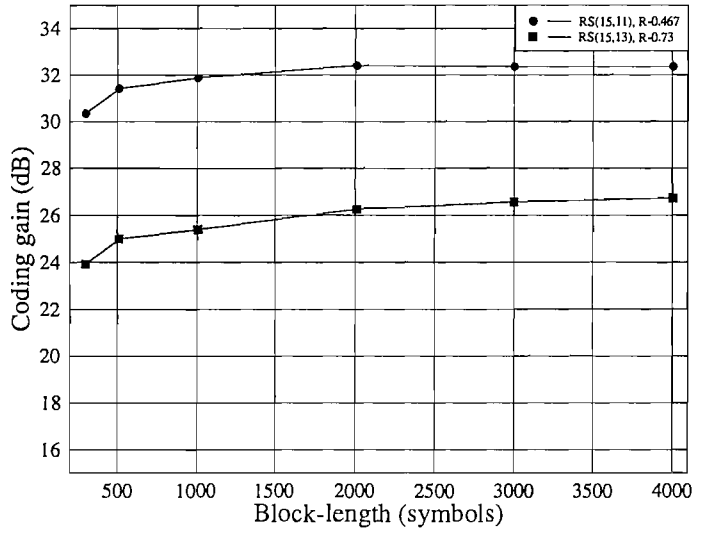


FIGURE 3.19: Coding gain achieved by the rate $R = 0.467$ GLDPC (300,140), (510,238), (1005,469), (2010,938), (3000,1400), (4005,1869) codes and rate $R = 0.73$ GLDPC (300,220), (510,374), (1005,737), (2010,1474), (3000,2200), (4005,2937) codes having different lengths and parameterized in Table 3.5, at a BER of 10^{-5}, when communicating over an **URF** channel.

3.6.3 Effects of the RS constituent Code

For the sake of quantifying the effects of different RS codes, nine nonbinary GLDPC codes defined over GF(16) and GF(32) were employed in our simulations using BPSK modulation for communicating over both AWGN and URF channels. The corresponding simulation parameters are listed in Table 3.6. Six codes were constructed over GF(32) and had a codeword length of $N = 1612$ 5-bit symbols. Specifically, the RS codes $(31, 19)$, $(31, 21)$, $(31, 23)$, $(31, 25)$, $(31, 27)$, $(31, 29)$ were used as our constituent codes, which have error correcting capabilities of $t = 6, 5, 4, 3, 2$ and 1 5-bit symbols, respectively. Similarly, three codes were constructed over GF(16) and had a codeword length of $N = 2010$ 4-bit symbols. The RS codes $(15, 9)$, $(15, 11)$, $(15, 13)$ were used as our constituent codes, which have error correcting capabilities of $t = 3, 2$ and 1 4-bit symbols, respectively. Since the constituent codes' rate affects the GLDPC codes' coding rates as well as coding performance, we are interested in their effect, when these RS codes are used in the GLDPC codes as the constituent codes.

It can be observed in Fig. 3.20 that the rate $R = 0.467$ GLPDC (2010,938) code using the RS (15,11) constituent code achieves the highest coding gain at a BER of 10^{-5}, when communicating over an AWGN channel. By contrast, the rate $R = 0.613$ GLPDC (1612,988) code using the RS (31,25) constituent code achieves the highest coding gain at a BER of 10^{-5}, when communicating over an AWGN channel as seen in in Figure 3.21.

GLDPC (N, k)	Constituent code	Coding rate R	Galois field	Decoding algorithm	Number of iterations	Channel
(2010,1474)	RS(15,13)	0.73	16	SF	20	AWGN
(2010,938)	RS(15,11)	0.467				/URF
(2010,402)	RS(15,9)	0.2				
(1612,1404)	RS(31,29)	0.871	32	SF	20	AWGN
(1612,1196)	RS(31,27)	0.742				/URF
(1612,988)	RS(31,25)	0.613				
(1612,780)	RS(31,23)	0.484				
(1612,572)	RS(31,21)	0.355				
(1612,364)	RS(31,19)	0.226				

TABLE 3.6: Simulation parameters for $J = 2$-level GLDPC codes constructed over GF(16) and GF(32), when communicating over an AWGN channel and an URF channel using different **RS codes** as constituent codes.

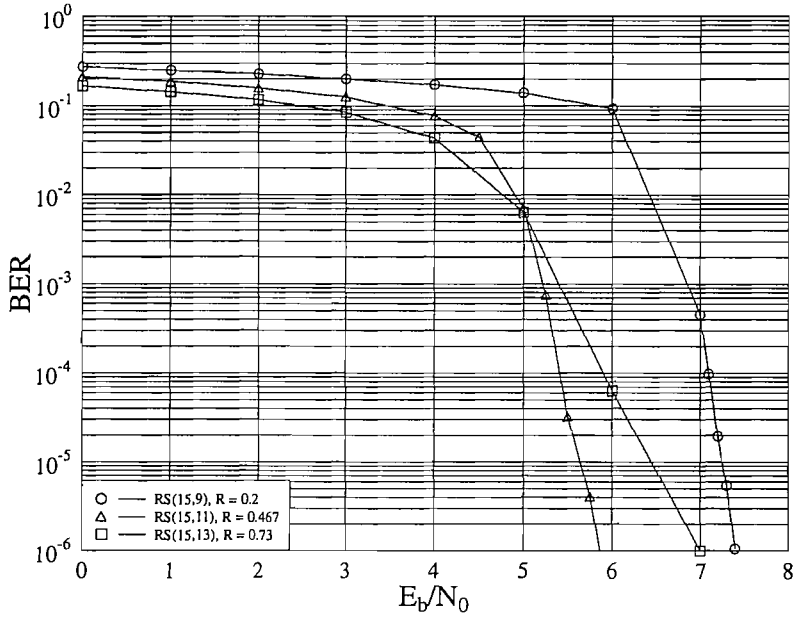


FIGURE 3.20: BER performance of the GLDPC (2010,1474), (2010,938), (2010,402) codes using **RS(15,k) constituent codes** parameterized in Table 3.5, when communicating over an **AWGN** channel.

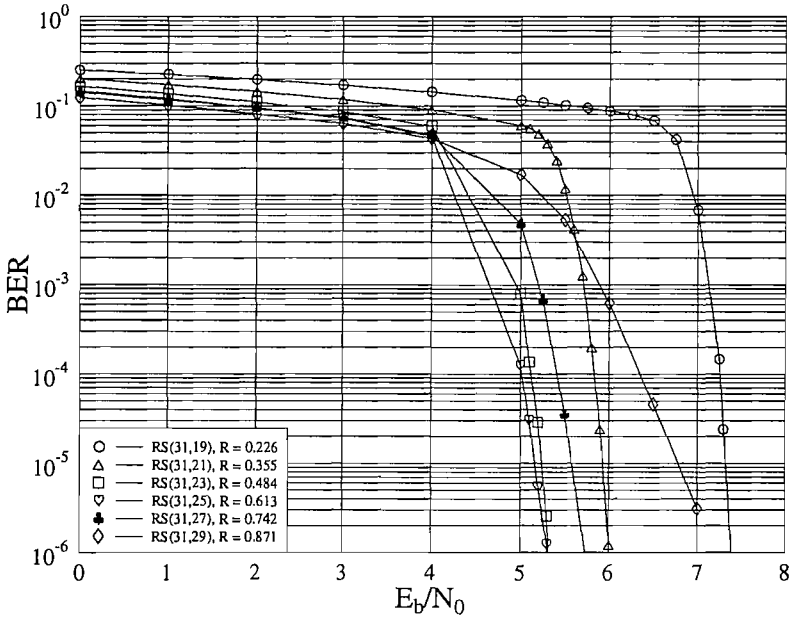


FIGURE 3.21: BER performance of the GLDPC (1612,1196), (1612,988), (1612,780), (1612,572), (1612,364) codes using **RS(31,k) constituent codes** parameterized in Table 3.5, when communicating over an **AWGN** channel.

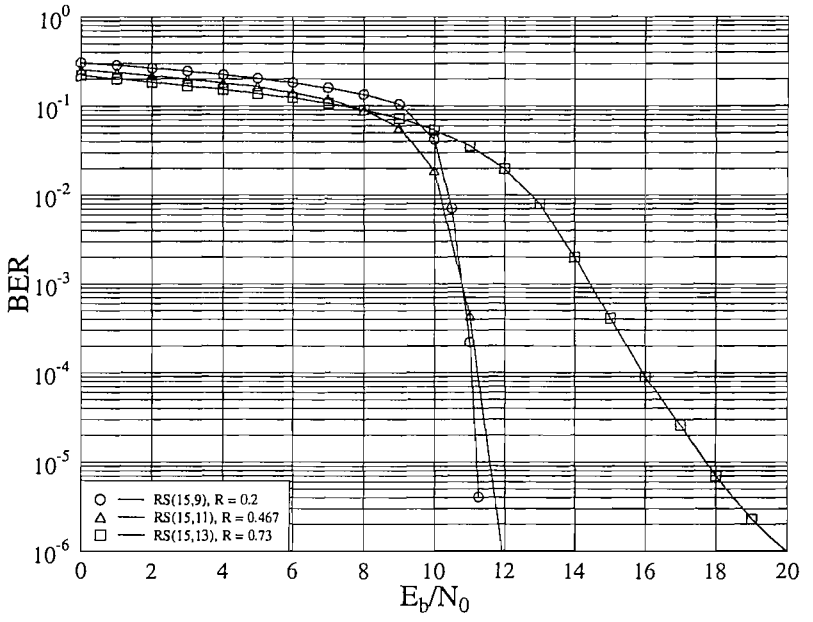


FIGURE 3.22: BER performance of the GLDPC (2010,1474), (2010,938), (2010,402) codes using **RS(15,k) constituent codes** parameterized in Table 3.5, when communicating over an **URF** channel.

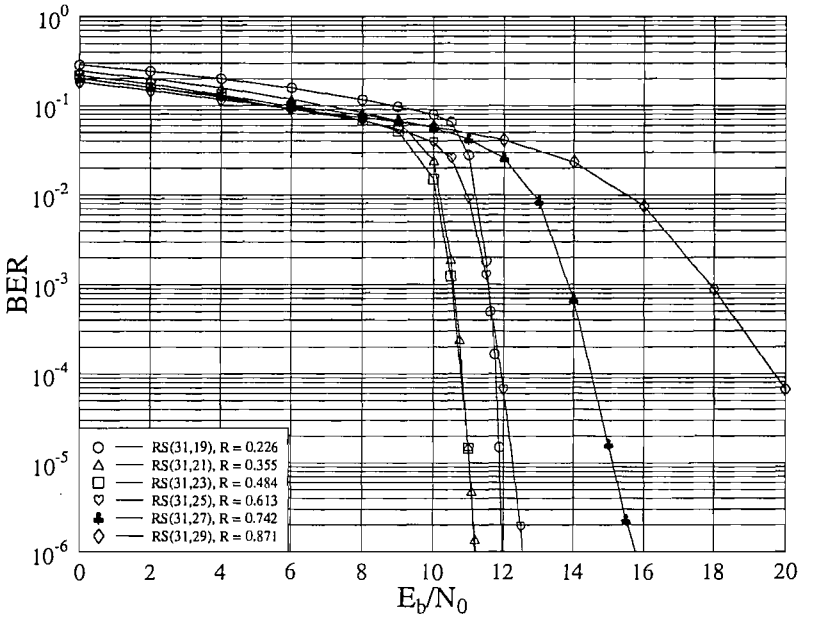


FIGURE 3.23: BER performance of the GLDPC (1612,1196), (1612,988), (1612,780), (1612,572), (1612,364) codes using the **RS(31,k) constituent codes** parameterized in Table 3.5, when communicating over an **URF** channel.

On the other hand, the best constituent code for the GLDPC codes defined over GF(16) and transmitted over an uncorrelated Rayleigh fading channel was seen to be the RS (15, 19) code in Figure 3.22. For GLDPC codes constructed over GF(32) and using RS(31,k) constituent codes, the GLDPC code using the RS(31,23) constituent codes achieves the highest coding gain, as depicted in Figure 3.23. Furthermore, the coding rate of the best GLDPC codes is lower, when communicating over a fading channel than over an AWGN channel, i.e. more powerful constituent codes have to be used as constituent codes, when the channel conditions degrade.

3.6.4 Effect of Different Galois Fields

In this section, we will evaluate the achievable performance of various GLDPC codes having the same block-length expressed in terms of bits and having similar coding rates despite using different Galois fields, when communicating over both AWGN and uncorrelated Rayleigh fading channels. The associated simulation parameters are summarized in Table 3.7.

Figure 3.24 characterized the attainable performance of various near-half-rate GLDPC codes studied having a similar binary block-length of 8000 bits, when communicating over an AWGN channel and an uncorrelated Rayleigh fading channel. At the BER of 10^{-6}, the GLDPC codes using the RS (31, 21) and RS(15,11) constituent codes both having a rate of $R = 0.484$ exhibits an E_b/N_0 improvement of 1 dB and 0.5 dB in comparison to the binary GLDPC code employing the (15, 11) Hamming constituent codes at a similar coding rate of $R = 0.467$, when communicating over an AWGN channel, respectively. In other words, the

Maximum number of iteration = 20					
GLDPC (N, K)	Constituent code	Coding rate R	Galois field	Decoding algorithm	Channel
(8010,3738)	Hamming(15,11)	0.467	2	WBFV	AWGN /URF
(2010,938)	RS(15,11)	0.467	16	SF	
(1612,780)	RS(31,23)	0.484	32	SF	
(2010,1474)	RS(15,13)	0.73	16	SF	
(1612,1196)	RS(31,27)	0.742	32	SF	

TABLE 3.7: Simulation parameters for $J = 2$-level GLDPC codes, when communicating over an AWGN channel and an URF channel using constituent RS codes constructed over different **Galois fields**.

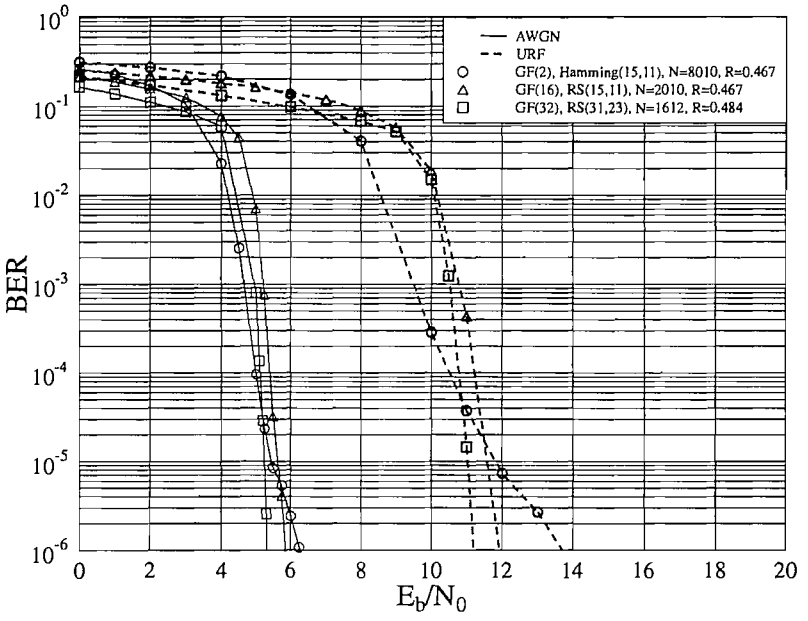


FIGURE 3.24: BER performance of the **coding rate** $R \approx 0.47$ GLDPC (8010,3738), (2010,938), (1612,780) codes using constituent codes constructed over different Galois fields parameterized in Table 3.5, when communicating over both an AWGN channel and an URF channel.

larger the Galois field applied, the better the achievable BER performance. However, in an AWGN channel, the binary GLDPC code outperforms the GLDPC code constructed over GF(16) at a BER lower than 10^{-5}. It may be attributed to the fact that the number of RS(15,11) constituent codes used by the GLDPC codes constructed over GF(16) is lower than that of the Hamming (15,11) constituent codes since their binary block-lengths are similar. Thus the correlation between the decisions of the super-codes C^1 and C^2 increases. As the symbols of the super-codes become more dependent on each other, the iterative decoder of the GLDPC code is less likely to correct the errors, although the error-correction capability of the RS(15,11) code is higher than that of the Hamming(15,11) code. In Figure 3.25 a similar phenomenon is observed for the GLDPC codes having a higher coding rate, when communicating over both an AWGN channel and an uncorrelated Rayleigh fading channels. Furthermore, as shown in Figure 3.25 for a coding rate coding rate R of 0.73, the GLDPC code constructed over GF(32) is capable of combatting the effects of a fading channel significantly better than the GLDPC constructed over GF(16).

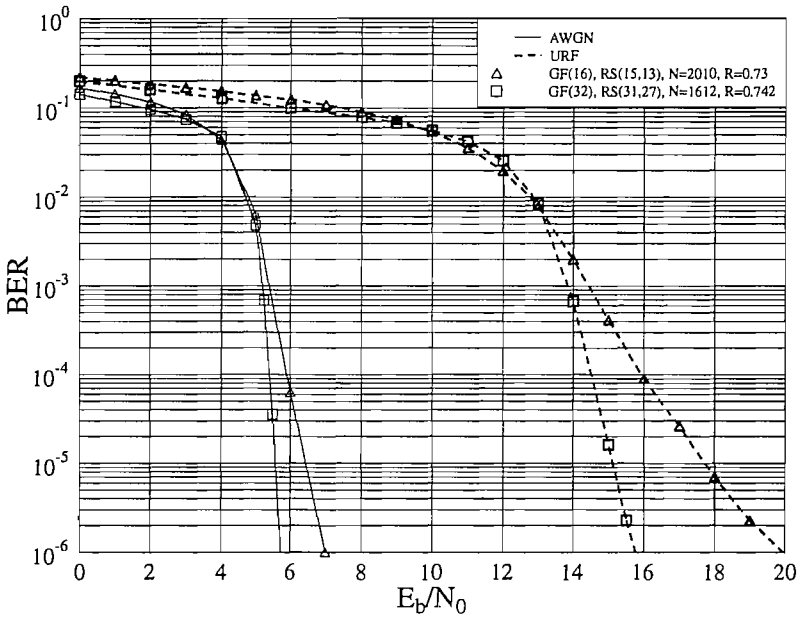


FIGURE 3.25: BER performance of the **coding rate** $R \approx 0.73$ GLDPC (2010,1474), (1612,1196) codes using constituent codes constructed over different Galois fields parameterized in Table 3.5, when communicating over both an AWGN channel and an URF channel.

3.7 Conclusion

In this chapter, we proposed the symbol-flipping based decoding of GLDPC codes constructed over $GF(q)$, which employ nonbinary constituent codes, e.g. nonbinary BCH codes or RS codes. Since our symbol-flipping based decoding algorithm was inspired by the concept of the WBFV algorithm designed for the binary Hamming-code based GLDPC codes [42], we gave a brief description of the WBFV decoding algorithm in Section 3.2. A further discussion of vote pairs designed for symbol-flipping based decoding algorithms was given in Section 3.3. The vote pairs used by the symbol-flipping algorithm are constituted by the vote labels E, e, F or V, which are assigned by the PGZ or BM HDDs of the nonbinary RS constituent codes. Furthermore, the vote pair reliability ordering, $VV > eV > FV > EV > ee > Fe > Ee > FF > EF > EE' > EE$ deduced from Figure 3.3 indicates the likely ordering of the probabilities that a symbol is in error, conditioned on its associated vote pair. In Section 3.4 we proposed seven different vote rules to design the vote weight and use simulations to find the most beneficial vote rule, which was deemed to be $E = 3$, $V = 0$, and $e = F = 1.5$, as shown in Table 3.3. We used larger values to indicate more unreliable symbols

and smaller values to indicate more reliable symbols. In Section 3.5, a step-by-step description of the symbol-flipping algorithm designed for nonbinary GLDPC codes was provided. In Section 3.6, the performance of nonbinary GLDPC codes using the advocated symbol-flipping algorithm was characterized in various scenarios. The effect of increasing the GLDPC decoder's complexity, i.e. the number of iterations was studied in Section 3.6.1. As seen in Figures 3.8 - 3.13, as the code's block-length is increased, more iterations are necessary to eliminate the erroneous symbols in the codeword, when communicating over AWGN channel and uncorrelated Rayleigh fading channels. However, the number of useful decoding iterations increases only slowly with N. The effect of increasing the GLDPC codes' block-length was demonstrated in Section 3.6.2, indicating that the attainable BER performance improves upon increasing the block-length N, when communicating over both AWGN and uncorrelated Rayleigh fading channels. The performance of GLDPC codes using the family of RS constituent codes having different error correction capability was evaluated in Section 3.6.3. For GLDPC codes using the family of RS(15,k) constituent codes, the best constituent codes were the RS(15,11) and RS(15,9) codes for transmission over AWGN and uncorrelated Rayleigh fading channels, respectively. By contrast, when considering the GLDPC codes using the family of RS(31,k) constituent codes, the best constituent codes are the RS(31,25) and RS(31,23) schemes, when communicating over AWGN and uncorrelated Rayleigh fading channels, respectively. In Section 3.6.4, the performance of GLDPC codes having the same block-length expressed in bits and having similar coding rates despite being constructed over different Galois fields was evaluated. At the BER of 10^{-6}, the GLDPC codes constructed over higher order Galois field exhibits a better performance, when communicating both AWGN and uncorrelated Rayleigh fading channels. In conclusion, the symbol-flipping decoding algorithm advocated can be successfully used for decoding GLDPC codes constructed from nonbinary constituent codes. The simulation results demonstrated that GLDPC codes defined over $GF(q)$ have the potential of outperforming similar-rate binary constituent codes.

Chapter 4

Conclusions

In this thesis, an efficient symbol-based hard decision aided decoding algorithm was designed for nonbinary GLDPC codes using RS constituent codes.

First of all, in Chapter 2 we introduced the structure of $J = 2$-level GLDPC codes and their iterative SISO decoding algorithm with the aid of an example. Then we outlined the characteristics of binary GLDPC codes using iterative SISO decoding in various quantifying scenarios, the effects of the number of iterations, those of the decoding algorithm applied by the constituent decoders as well as those of the coding rate, block-length and benefits of shortened binary Hamming codes used as constituent codes.

In Chapter 3, we proposed a novel symbol-based hard decision based symbol-flipping decoding algorithm designed for GLDPC codes constructed over GF(q) using RS constituent codes. This decoding philosophy was motivated by the Weighted Bit Flip Voting (WBFV) algorithm designed for binary Hamming based GLDPC codes. Thus we briefly introduced the WBFV algorithm in Section 3.2. The entire design process of the symbol-flipping based decoding algorithm was demonstrated in Section 3.3 and Section 3.4. The concept of the vote pairs, VV, eV, FV, EV, ee, Fe, Ee, FF, EF, EE', and EE in the symbol-flipping based decoding algorithm. Seven vote rules were proposed in Section 3.4 and the suggested optimal voting rule was deemed to be $E = 3$, $V = 0$, and $e = F = 1.5$, as shown in Table 3.3, where larger values indicates unreliable symbols and smaller values indicates more reliable symbols. A summary of the symbol-flipping based decoding algorithm designed for nonbinary GLDPC codes using RS constituent codes was provided in Section 3.5. Our simulations demonstrated that the symbol-flipping

decoding algorithm inspired by the WBFV algorithm of [42] can be successfully used for decoding nonbinary GLDPC codes constructed from RS constituent codes. It was also demonstrated that GLDPC codes defined over $GF(q)$ have the potential of outperforming similar-rate binary constituent codes. Furthermore, we also characterized the achievable performance of GLDPC codes invoking various numbers of iterations, block-lengths, as well as different RS codes as constituent codes.

Our future research will design a large variety of GLDPC codes, in an effort to identify the best combination of system components.

Bibliography

[1] C. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, pp. 379–427, 1948.

[2] R. Hamming, "Error Detection and Error Correcting Codes," *Bell System Technical Journal*, vol. 29, pp. 147–160, 1950.

[3] P. Elias, "Coding of Noisy Channels," *IRE Convention Record*, vol. 4, pp. 37–47, 1955.

[4] A. Hocquenghem, "Codes correcteurs d'erreurs," *Chiffres*, vol. 2, pp. 147–156, September 1959.

[5] R. C. Bose and D. K. Ray-Chaudhuri, "On a class of error correcting binary group codes," *Information and Control*, vol. 3, pp. 68–79, March 1960.

[6] R. C. Bose and D. K. Ray-Chaudhuri, "Further results on error correcting binary group codes," *Information and Control*, vol. 3, pp. 279–290, September 1960.

[7] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society of Industrial and Applied Mathematics SIAM*, vol. 8, pp. 300–304, June 1960.

[8] G. D. Forney, *Concatenated Codes.* Cambridge, MA: MIT Press, 1966.

[9] V. A. Zinov'ev, "Generalizeed Concatenated Codes," *Problemy Peredachi Informatsii*, vol. 12, no. 1, pp. 5–15, 1976.

[10] F. J. MacWilliams and N. J. A. Sloane, *The theory of Error-correcting Codes.* Amsterdam, The Netherlands: North-Holland, 1993.

[11] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes," in *IEEE International Conference on Communications*, pp. 1064–1070, 1993.

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

[13] R. Gallager, "Low density parity check codes," *IEEE Transaction on Information Theory*, vol. 8, pp. 21–28, January 1962.

[14] M. Sipser and D. A. Spielman, "Expander codes," *IEEE Transactions on Information Theory*, vol. 42, no. 6, pp. 1710 – 1722, 1996.

[15] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Transactions on Information Theory*, vol. 45, pp. 1710–1722, Mar. 1999.

[16] D. J. C. MacKay and R. M. Neal, "Good codes based on very sparse matrices," in *C. Boyd (editor) Cryptography and Coding: 5th IAM Conference*, vol. 2, pp. 100–111, Springer 1995. Lecture Notes in Computer Science.

[17] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on Information Theory*, vol. IT-27, pp. 533 – 547, Sept. 1981.

[18] N. Wiberg, H.-A. Loeliger, and R. Kotter, "Codes and Iterative Decoding on General Graphs," *European Transactions on Telecommunications*, pp. 513–525, September 1995.

[19] N. Wiberg, *Codes and Decoding on General Graphs.* Phd thesis, Linkoping University, Sweden, 1996.

[20] B. J. Frey and F. R. Kschischang, "Probability propagation and iterative decoding," in *Proceedings of the 34 th Allerton Conference*, October 1996.

[21] B. J. Frey, "Pseudo-Random Codes Based on Bayesian Networks," in *Proceedings of the 5th Candian Workshop on Information Theory(CWIT)*, pp. 15–18, June 1997.

[22] F. R. Kschischang and B. Frey, "Iterative Decoding of compound Codes by Probability Propagation in Graphical Models," *IEEE Journal on Selected Area in Communications (JSAC)*, vol. 16, pp. 219–230, February 1998.

[23] B. J. Frey, F. R. Kschischang, H. A. Loeliger, and N. Wiberg, "Factor graphs and algorithms," in *Proceedings of the 35 th Allerton Conference*, October 1997.

[24] F. R. Kschischang, B. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on Information Theory*, vol. 47, pp. 498–519, February 2001.

[25] G. D. Forney, "On iterative decoding and the two-way algorithm," in *Proceedings of the International Symposium Turbo Codes and Related Topics*, Sept. 1997.

[26] G. D. Forney, "Codes on graphs: Normal realizations," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 520–548, 2001.

[27] G. D. Forney, "Codes on graphs: constraint complexity of cycle-free realizations of linear codes," *IEEE Transactions on Information Theory*, vol. 49, no. 7, pp. 1597–1610, 2003.

[28] S. Benedetto and G. Montorsi, "Serial concatenation of block and convolutional codes," *Electronics Letters*, vol. 32, pp. 887–888, May 1996.

[29] S. Benedetto and G. Montorsi, "Iterative decoding of serially concatenated convolutional codes," *Electronics Letters*, vol. 32, pp. 1186–1187, June 1996.

[30] J. Pearl, "Fusion, propagation and structuring in belief networks," *Artificial Intelligence*, vol. 29, no. 3, pp. 241–288, 1986.

[31] J. Boutros, O. Pothier, and G. Zemor, "Generalized low density (Tanner) codes," in *IEEE Internationl Conference on Communications (ICC 1999)*, vol. 1, pp. 441 – 445, June 1999.

[32] O. Pothier, L. Brunel, and J. Boutros, "A low complexity FEC scheme based on the intersection of interleaved block codes," in *Vehicular Technology Conference*, pp. 274–278, 1999.

[33] M. Lentmaier and K. S. Zigangirov, "On generalized low-density parity-check codes based on Hamming component codes," *IEEE Communication Letters*, vol. 3, pp. 248 – 250, Aug. 1999.

[34] O. Pothier, *Compound codes based on graphs and their iterative decoding.* Ph.d. thesis, Ecole Nationale Suprieure des Telecommunications, Jan. 2000. http://www.comelec.enst.fr/ boutros/coding/.

[35] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimum Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Transaction on Information Theory*, vol. 20, pp. 284–287, March 1974.

[36] T. Johansson and K. Zigangirov, "A simple one-sweep algorithm for optimal APP symbol decoding of linear block codes," *IEEE Transactions on Information Theory*, vol. 44, pp. 3124–3129, Nov. 1998.

[37] D. Chase, "A class of algorithms for decoding block codes with channel measurement information," *IEEE Transaction on Information Theory*, vol. 18, pp. 170–182, Jan. 1972.

[38] S. Hirst and B. Honary, "Application of efficient Chase algorithm in decoding of generalized low-density parity-check codes," *IEEE Communications Letters*, vol. 6, pp. 385 – 387, Sept. 2002.

[39] T. Kaneko, T. Nishijima, H. Inazumi, and S. Hirasawa, "An efficient maximum-likelihood-decoding algorithm for linear block codes with algebraic decoder," *IEEE Transaction Information Theory*, vol. 40, pp. 320–327, Mar. 1994.

[40] T. M. N. Ngatched and F. Takawira, "Efficient decoding of generalized low-density parity-check codes based on long component codes," in *2003 IEEE Wireless Communications and Networking*, vol. 1, pp. 705 – 710, March 2003.

[41] T. Zhang and K. Parhi, "High-performance, low-complexity decoding of generalized low-density parity-check codes," in *IEEE Global Telecommunications Conference*, vol. 1, pp. 181 – 185, Nov. 2001.

[42] S. Hirst and B. Honary, "Decoding of generalised low-density parity-check codes using weighted bit-flip voting," *IEE Proceedings Communications*, vol. 149, pp. 1 – 5, Feb. 2002.

[43] T. Zhang and K. Parhi, "A class of efficient-encoding generalized low-density parity-check codes," in *2001 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '01)*, vol. 4, pp. 2477 – 2480, May 2001.

[44] I. Djordjevic, O. Milenkovic, and B. Vasic, "Generalized low-density parity-check codes for optical communication systems," *Journal of Lightwave Technology*, vol. 23, pp. 1939 – 1946, May 2005.

[45] M. C. Davey and D. MacKay, "Low-density parity check codes over GF(q)," *IEEE Communications Letters*, pp. 165–167, June 1998.

[46] F. Guo, *Low Density Parity Check Coding.* Ph.D. thesis, University of Southampton, January 2005.

[47] T. Zhang, *Efficient VLSI Architectures for Error-Correction Codingh.* Ph.D. thesis, University of Minnesota, July 2002. http://www.ecse.rpi.edu/homepages/tzhang/RVSAL/research.html.

[48] C. Berrou and A. Glavieux, "Near optimum error correcing coding and decoding: turbo-codes," *IEEE Transactions on Communications*, vol. 44, pp. 1261–1271, Oct. 1996.

[49] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Transactions on Information Theory*, vol. 42, pp. 429–445, March 1996.

[50] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," in *IEEE International Conference on Communications*, vol. 2, pp. 1009–1013, June 1995.

[51] L. Hanzo, T. Liew, and B. L. Yeap, *Turbo Coding, Turbo Equalisation and Space-Time Coding.* John Wiley & Sons Ltd., 2002.

[52] H. Imai and S. Hirakawa, "A new multilevel coding method using error-correcting codes," *IEEE Transactions on Information Theory*, vol. 23, pp. 371–377, May 1977.

[53] U. Wachsmann, R. Fischer, and J. Huber, "Multilevel Codes: Theoretical Concepts and Practical Design Rules," *IEEE Transactions on Information Theory*, vol. 45, p. 1361V1391, July 1999.

[54] L. Hanzo, S. X. Ng, T. Keller, and W. Webb, *Quadrature Amplitude Modulation: From Basics to Adaptive Trellis-Coded, Turbo-Equalised and Space-Time Coded OFDM, CDMA and MC-CDMA Systemss.* John Wiley and Sons Ltd., 2 ed., November 2004.

[55] Y. Kou, S. Lin, and M. P. C. Fossorier, "Low-density parity-check codes based on finite geometries," *IEEE Transactions on Information Theory*, vol. 47, pp. 2711–2736, Nov. 2001.