TELECOM
PARIS

Thèse

présentée pour obtenir le grade de docteur

de l'Ecole Nationale Supérieure
des Télécommunications

Spécialité : Electronique et Communications

Olivier Pothier

Codes composites construits à partir de graphes et leur décodage itératif

soutenue le 27 janvier 2000 devant le jury composé de

Gilles Zémor	Président
Gérard Battail	Rapporteurs
Ezio Biglieri	
Antoine Chouly	Examinateurs
Ramesh Pyndiah	
Rüdiger Urbanke	
Joseph Boutros	directeur de thèse

Ecole Nationale Supérieure des Télécommunications, Paris

To my parents,
for the education they gave me.

To my teachers, in all fields,
I know so little compared to them.

A mes parents,
pour l'éducation qu'ils ont su me donner.

A mes professeurs dans tous les domaines,
puissé-je un jour en savoir autant qu'eux !

Remerciements

Il est de coutûme pour un thésard, au moment de mettre un point final à son rapport, de remercier les personnes qui l'ont aidé et soutenu tout au long de sa thèse. Cette tâche me paraissait, il y a encore quelques semaines, lointaine, presque inaccessible. Par moment, au cours de la rédaction, je laissais divaguer mon esprit, et me voyais rédiger cette page dans un halo de brume joyeuse. Je construisais de belles formules raffinées pour exprimer tour à tour ma gratitude aux personnes qui apparaissaient telles des icônes, à travers la brume. Ces visions m'ont été réconfortantes, reposantes et stimulantes lors de la rédaction, tâche ingrate.

Au moment de passer à l'acte, la brume se fait brouillard, le halo n'est plus, les icônes ont disparu, les mots ne viennent pas, seule une corne de brume lointaine et nostalgique semble sonner l'oraison de ces trois ans et quelques mois supplémentaires. Dépression *post-partum* ? Icônes, fendez le brouillard, et apparaissez (dans l'ordre protocolaire, s'il vous plaît) ! Que toutes celles que je vais oublier ne s'en offusquent pas, mais s'imaginent avec humour gravées sur le bois doré...

Je tiens tout d'abord à remercier Gilles Zémor, de l'E.N.S.T., pour m'avoir fait l'honneur de présider le jury de cette thèse. Les discussions que j'ai eues avec lui, en particulier concernant la théorie des graphes, m'ont été d'une grande aide, bien que concourant souvent à faire naître en moi un autre brouillard, celui de la prise de conscience de mon ignorance. Ce dernier est heureusement soluble dans le travail.

Mes plus vifs remerciements s'adressent au Professeur Gérard Battail, qui a cordialement accepté d'être rapporteur de ce travail, et pour lequel j'éprouve le plus grand respect. La qualité de ce rapport s'est vue remarquablement améliorée par sa relecture attentive et pointilleuse, et ses nombreuses corrections.

Je tiens également à exprimer ma profonde gratitude envers le Professeur Ezio Biglieri, du Politecnico di Torino, pour avoir lui aussi accepté d'être rapporteur. Ses commentaires élogieux m'ont touché. Qu'il trouve ici l'expression de ma sincère reconnaissance.

Je remercie les membres du jury, Messieurs les Professeurs Ramesh Pyndiah, de l'E.N.S.T. Bretagne, et Rüdiger Urbanke, de l'E.P.F.L., pour leur lecture attentive de ce manuscrit, leurs remarques constructives et les discussions (parfois animées) que nous avons eues. Leurs réputations scientifiques honorent mon travail.

Je n'oublie évidemment pas les laboratoires d'électroniques Philips (L.E.P.) et Mohammad Haghiri en particulier, qui a été à l'origine du contrat de collaboration entre le département Communication de l'E.N.S.T. et la division 23 du L.E.P. Il est rare qu'une entreprise accorde une telle liberté au thésard qu'elle finance. Je remercie tout particulièrement mon encadrant au sein du L.E.P., Antoine Chouly, dont la gentillesse n'a d'égale que la compétence technique et la discrétion.

Les mots me manquent lorsque j'en viens à mon directeur de thèse. Joseph, sans toi, mon travail et ce rapport, son aboutissement, n'auraient jamais vu le jour. Merci pour ton enthousiasme, ton dynamisme, ton écoute (tu es cependant parfois difficile à convaincre), tes orientations, ton encadrement, mais surtout pour la confiance que tu as su m'accorder.

Les moments que j'ai passés au département Communication de l'E.N.S.T. ont été un parfait equilibre de travail solitaire (le thésard est un coureur de fond), d'échanges scientifiques stimulants, passionnés et parfois impromptus, devant un tableau blanc ou un café noir, et d'enrichissement personnel. Merci au Professeur Philippe Gallion de m'y avoir accepté, aux secrétaires Jany Batz, Laurence Monnot et Danielle Childz pour leur aide efficace, merci à tous les enseignants. Merci à tous les élèves de l'E.N.S.T. qui ont assisté à mes cours, ou que j'ai encadrés, pour l'approfondissement des connaissances et les progrès de clarté, de pédagogie et de synthèse que vous m'avez forcé à accomplir.

Et surtout merci à tous mes compagnons de fortune : Christine (tabernac'), Mono (hola, feo), Bahram (et sa cravache), Amal (3 ans de vie commune), Loïc (trop à dire en une parenthèse), Catherine (vraiment trop), F-X (caliméro des sys-admins), Sandrine (la relève), Céline (de souche), Cédric (c'est pour bientôt ?), X-tof (Champs de Mars, demi-pression), Sabine (cendrée verzionaise), Stefan (et ses coloc'), Christophe (cachax) et tous les autres, pour tout ce que vous m'avez apporté.

Merci à tous mes amis de m'avoir accompagné et supporté.

Merci à Marie.

Merci à mes parents.

Paris, le 22 février 2000

Résumé

Cette thèse propose et analyse des techniques dont l'objet est la compréhension des schémas de codage de canal composites existant et la construction de nouveaux schémas au moyen de représentations graphiques.

Ces travaux ont été motivés par les récents progrès en codage de canal, et plus particulièrement l'invention en 1993 des turbo codes. Ces derniers font partie d'une classe plus étendue de codes, les codes composites.

Les codes composites consistent en l'association de codes constituants liés. Chacun de ces codes posséde un décodeur élémentaire à entrée et sortie souples. Le décodage des codes composites s'effectue de manière coopérative à l'aide de décodeurs élémentaires qui échangent des informations probabilistiques, au cours d'un processus itératif. Les critères classiques d'évaluation des codes, sous l'hypothèse d'un décodage à maximum de vraisemblance, montrent dès lors leur limite dans l'analyse de ces codes. Le décodage itératif peut cependant être interprété comme une propagation de probabilités le long d'un graphe représentant le code composite.

Dans une première partie, une représentation graphique spécifique, le graphe de dépendance, est introduite pour les turbo codes. Elle se révèle être un outil utile dans la compréhension et l'analyse du décodage itératif.

Dans une seconde partie, nous élargissons l'utilisation de représentations graphiques à la construction de nouveaux codes composés, les codes GLD (generalized low density), qui se trouvent à la croisée des chemins des codes à faible densité de Gallager, et des codes graphiques de Tanner. Une étude théorique et pratique de ces codes est menée. Enfin, des entrelaceurs spécifiques aux codes GLD, et à leur décodage itératif, sont présentés.

Compound codes based on graphs and their iterative decoding

Olivier Pothier

Ecole Nationale Supérieure des Télécommunications, Paris
Digital communication group, COMELEC Department

January 27, 2000

Contents

List of Figures

List of Tables

Acronyms and Notations

We present here the main acronyms and notations used in this document. We tried to keep consistent and homogeneous notations in the entire report. However, in some rare places, notations do not have their general meaning summarized here. In these cases, they are redefined before their use. Normally, acronyms are recalled the first time they appear in any chapter.

Acronyms

APP	*A Posteriori* Probability
AWGN	Additive White Gaussian Noise (channel)
BCH	Bose-Chaudhuri-Hocquenghem (code)
BER	Bit Error Rate
BPSK	Binary Phase Shift Keying (modulation)
BSC	Binary Symmetric Channel
cdf	cumulative (probability) density function
DG	Dependency Graph
DMC	Discrete Memoryless Channel
iid	independent and identically distributed
FB	Forward-Backward (algorithm)
FER	Frame (Block) Error Rate
GLD	Generalized Low Density (code)
IOWEF	Input-Output Weight Enumerator Function
LDPC	Low Density Parity Check (code)
LLR	Log Likelihood Ratio
MAP	Maximum *a posteriori* (decoding)
ML	Maximum Likelihood (decoding)
PCCC	Parallel Concatenated Convolutional Code
pce	parity check equation
pdf	probability density function
SCCC	Serial Concatenated Convolutional Code

spcc	single parity check code
SISO	Soft-Input Soft-Output (decoder)
SOVA	Soft-Output Viterbi Algorithm
SNR	Signal-to-Noise Ratio
VA	Viterbi Algorithm
VG	Varshamov-Gilbert (bound)

Notations

c_i	If not specified, the i-th bit of the considered codeword
$\underline{c}_j$	If not specified, the j-th codeword of the considered code
c_ℓ	If not specified, a codeword of weight ℓ
$\mathcal{C}$	If not specified, the considered compound code
$\mathcal{C}_0$	If not specified, the considered constituent code
d_{Hmin}	The minimum Hamming distance of the considered code
E_b	Carrier Frequency Energy per information bit
E_b/N_0	Signal to Noise Ratio per information bit
k	If not specified, the dimension of the considered constituent code
K	If not specified, the dimension of the considered compound code
ℓ	If not specified, the weight of the considered codeword
LR_p	log ratio of the probabilities of any random variable a that can take two values : $\ln(p(a=1)/p(a=0))$
n	If not specified, the length of the considered constituent code
N	If not specified, the length of the considered compound code, or the length of the interleaver of the considered turbo code
$N(\ell)$	Weight distribution of the considered code
$\overline{N(\ell)}$	Average weight distribution of the considered ensemble of codes
$N_0/2$	Two sided Power Spectral Density of the AWGN
P_{eb}	Bit Error Probability (or BER)
P_{ew}	Word Error Probability (or FER)
r	If not specified, the rate of the considered constituent code
R	If not specified, the rate of the considered compound code

Introduction

This thesis proposes and analyses channel coding techniques using graphical representations in order to construct and understand existing and new compound coding schemes.

We were motivated by recent progress in channel coding, especially the invention of turbo codes in 1993. From 1948, when Claude Shannon layed the mathematical foundation of the information theory, till the empirical breakthrough due to the outstanding performance of turbo codes, the two major research directions taken by scientists working on communication over the binary symmetric and Gaussian channels were the construction of powerful codes approaching the capacity limit on one side, and (jointly or not) the development of efficient decoding algorithms on the other side. Forty five years of research led to a large amount of knowledge and numerous successful applications. Codes based on algebraic constructions were used for high rate (greater than 1/2) and low noise applications. Their decoding takes benefit of their algebraic properties. Convolutional codes and their trellis-based decoding algorithm, using a maximum likelihood and probabilistic approach, were applied for low rates (less than 1/2) and noisy channels. The best performance was obtained by the serial concatenation of these two types of codes. The graph theory was confined to the algebraic part of channel coding.

Turbo codes, *i.e.* parallel concatenation of convolutional codes *and* their iterative “turbo” decoding, can be seen as an instance of a more general class of codes, called compound codes. Compound codes can be defined as a collection of interacting constituent codes. Each of these constituent codes must be easily decodable. The decoding of the compound code is performed by means of cooperative elementary decoders that exchange information between the constituent codes. Several decoding steps are necessary, leading to an iterative process.

The first compound scheme was introduced thirty five years ago by Gallager. In his thesis, Gallager introduced the low density parity check (LDPC) codes, and presented an iterative "probabilistic" decoding algorithm. However, for complexity reasons, LDPC codes had almost fallen into oblivion. Recent results indicate that *irregular* LDPC codes can even beat turbo codes for extremely large code length.

Conventional evaluation criteria of channel codes, *i.e.* minimum Hamming distance and Maximum Likelihood (ML) decoding bounds, turn out to be of limited usefulness to understand and evaluate the performance of compound codes and their decoding. Graph theory, that stays on the fringe of the channel coding community interests, gained a renewed attention and contributes to the understanding and evaluation of iterative decoding. Indeed, the latter can be interpreted as a probability propagation along a graph representing the compound code.

Thesis outline

This thesis is divided into two parts. The first part (Chapters 1 and 2) is devoted to turbo codes. Chapter 1 describes their structure and iterative decoding, and briefly recalls classical results. It analyses the "ingredients" that make turbo codes so powerful. Chapter 2 introduces a graphical representation of turbo codes, called the dependency graph, which is a useful tool in the analysis of iterative decoding.

The second part (Chapters 3, 4, and 5) enlarges the use of graphical representations to the construction of new compound codes, called generalized low density (GLD) codes, that are also iteratively decodable. GLD codes stand at the crossroad of Gallager's LDPC codes and Tanner's graph codes. LDPC codes definition, properties, and decoding method are briefly presented in Chapter 3. Chapter 4 extensively studies GLD codes and their iterative decoding. Chapter 5 is an even deeper intrusion of the graph theory into channel coding theory, where we propose algebraic graphs that lead to GLD codes with good iterative decoding properties.

Chapters 2, 4 and 5 constitute the original work that has been carried through. The two parts of this document can be read almost independently. For this reason, several notions may have been introduced twice, for example the notion of dependency graphs, however with consistent definitions.

Note that we restrict our study to the following frame : we assume an additive white Gaussian noise (AWGN) channel in almost all this work, and a binary antipodal phase shift keying (BPSK) modulation. we focus our attention to small length codes suitable for frame transmissions. Furthermore, in this report, the soft-input soft-output decoding of parallel turbo-, low-density parity-check-, product- and generalized low-density parity-check codes uses the forward-backward algorithm and does not use weighting factors during the extrinsic information propagation. We aimed to avoid the influence of any simplication or empirical modification of the decoding algorithm.

Chapter 1

Turbo codes and their decoding

Turbo codes (parallel concatenation of convolutional codes) have been introduced in 1993 by Berrou, Glavieux and Thitimajshima [22], who presented an iterative decoding scheme based on soft-input soft-output (SISO) decoding of the constituent convolutional codes[1]. Their outstanding performance raised astonishment, curiosity and even incredulity in the coding community. Independently and at the same time, Lodge, Young, Hoeher and Hagenauer [67] [68] introduced a similar iterative decoding of product codes.

The concatenation of several constituent codes dates back to the earlier ages of channel coding : iterated product codes were introduced by Elias in 1954 [38], and serially concatenated codes have been studied by Forney [41]. The great success of turbo codes came from their decoding procedure.

Battail [10] pointed out that turbo codes take benefit of three major concepts : random-like coding, concatenation of several constituent codes into a compound code, and successive soft-input soft-output (SISO) decoding of the constituent codes. These concepts were already used thirty years earlier by Gallager in his low-density parity-check (LDPC) codes [47], before they sink into oblivion. Chapter 3 summarizes the major characteristics of LDPC codes.

Since 1993, a huge amount of work has been carried out on turbo codes. The major research directions taken by scientists throughout the world in-

[1]"Turbo code" is a misleading name. It is actually intended to mean parallel concatenated convolutional codes (PCCC) and their "turbo" decoding, as pointed out in [55].

clude :

- Understanding and analyzing turbo code performance :
  - The average "effective" minimum distance and the input-output weight enumerator function (IOWEF) of the ensemble of turbo codes sharing the same constituent codes have been studied in [15] [35] [104], leading to an assessment of their theoretical maximum likelihood (ML) performance, using the union bound (UB). The growth rate of the minimum distance as a function of the interleaver length has been further studied in [60].
  - Improved bounds on ML performance, using Gallager's bound [37] or the tangential sphere bound (TSB) [93] [92] allow to overcome the weakness of the UB, that diverges below the cutoff rate.
  - The turbo decoder is however not an ML decoder. Hence, even if the ML bounds gave insights on the performance, specific analysis of iterative decoding performance had to be done. Hokfelt *et al.* [57] [58] studied it in term of the correlation of extrinsic probabilities, and Ten Brink [98] introduced the mutual information transfer characteristics of SISO decoders as a tool to better understand the iterative decoding convergence.
  - The representation of turbo codes as compound codes based on graphs led to fruitful developments made by Frey, Kschischang, Loeliger, MacKay, McEliece and Wiberg. These works are referred to in Section 2.1. Chapter 2 of this report is a contribution to this topic.
- Finding design criteria for turbo codes :
  - The choice of the constituent codes have been investigated by Benedetto and Montorsi [14] [15]. It appears that the convolutional codes must be recursive, and have a primitive feedback polynomial. The choice of the direct polynomial(s) has no influence, provided it (they) has (have) the same degree as the primitive feedback polynomial.
  - The first method to build improved random interleavers for turbo codes was designed by Divsalar and Dolinar [36]. This so-called S-random interleaver aims at spreading as much as possible the pairs of adjacent information bits. Hokfelt *et al.* [57] [58] showed

however that these interleavers do not always present the best decorrelation properties. The influence of the interleaver cycles on the iterative decoding is extensively studied in this report, as regards turbo codes (Chapter 2), LDPC (Chapter 3) and GLD codes (Chapter 5). Algebraic constructions of deterministic interleavers for GLD codes that outperform random interleavers are presented in the last chapter.

- Extending the turbo decoding scheme to other compound systems :

  - Serial concatenation of convolutional codes is a straightforward extension of conventional turbo codes [16] [17]. Serial turbo codes exhibit better "effective" minimum distance and higher interleaving gain[2] than parallel turbo codes [16]. However, the performance of their iterative decoding is farther from ML decoding than that of parallel turbo codes.
  - Iterative decoding of block product codes has been studied by Pyndiah *et al.* [84] [85] [86].
  - The "turbo" principle has been adapted to iterative soft-output equalization and channel decoding by Glavieux *et al.* [49], and called turbo equalization.
  - Iterative joint demodulation and decoding schemes were also studied in [29] [77].

- The SISO decoding of individual codes presented in [22] involves maximum *a posteriori* (MAP) symbol-by-symbol decoding by the forward-backward (FB) algorithm. Reducing its complexity by allowing a sub-optimal implementation is a major task to widespread the use of turbo codes in all-day applications. The work of Hagenauer *et al.* [52] and Robertson *et al.* [89] [90] contributed to it.

In this short introductory chapter, we only describe the structure of turbo codes. We briefly recall the major analytical results and explain the iterative turbo decoding as an APP passing procedure independent of the specific SISO algorithm applied on the constituent codes.

[2]The notion of "interleaving gain" is introduced in Section 1.2.

1.1 Structure of turbo codes

In this section, we present the general structure of parallel and serial turbo codes.

1.1.1 Parallel turbo codes

A parallel turbo code $\mathcal{C}$ with J levels is a linear binary *block* code of dimension N. It results from the parallel concatenation of J convolutional codes $\mathcal{C}^j$, $1 \leq j \leq J$, as depicted in Figure 1.1.

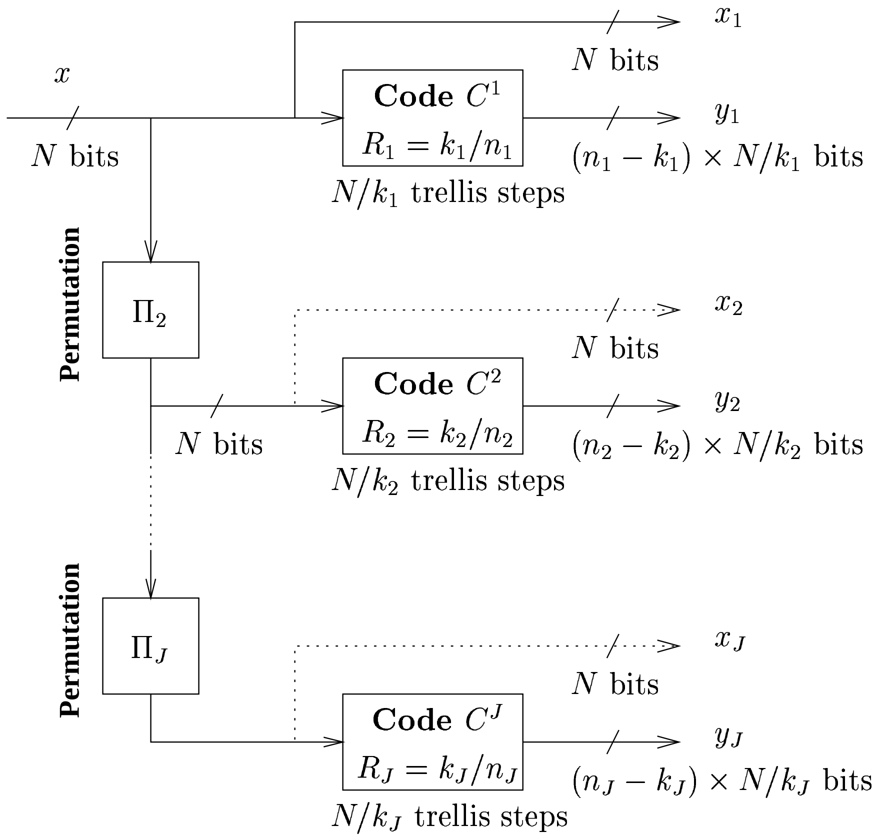


Figure 1.1: Encoder of a parallel turbo code with J levels.

The convolutional codes $\mathcal{C}^j$ are either recursive systematic convolutional (RSC) codes of rate $R_j = k_j/(k_j + 1)$ or non-recursive non-systematic convolutional (NRNSC) codes of rate $R = k_j/n_j$. Their constraint length is L_j and their memory $\nu_j = L_j - 1$. Benedetto *et al.* [14] [15] proved theoretically that turbo decoding works only if the constituent codes are RSC. Hence, we focus our attention on these codes.

The trellis of each code $\mathcal{C}^j$ starts at the zero-state and ends after $N/k_j + T_j$ trellis transitions again at the zero-state. Each code needs at most T_j transitions to return to the zero-state, where :

$$T_j = L_j - 1 = \nu_j \quad \text{transitions} \quad \text{(NRNSC code)} \tag{1.1}$$

or

$$T_j = \lceil \frac{L_j - 1}{k_j} \rceil \quad \text{transitions} \quad \text{(RSC code)} \tag{1.2}$$

Practically, we often chose $J = 2$ and $\mathcal{C}^1 = \mathcal{C}^2$, and always recursive systematic codes.

The N information bits x are encoded by $\mathcal{C}^1$, and, for any j, interleaved through an interleaver Π_j and then encoded by $\mathcal{C}^j$. Each RSC code $\mathcal{C}^j$ produces $(n_j - k_j)N/k_j$ parity bits denoted by y_j. The information bits $x = x_1$ and $x_2, \cdots, x_J$ are the same, apart from the permutation Π_j. Hence, only x is transmitted so as to increase the global rate R of $\mathcal{C}$.

For $J = 2$, and if we neglect the trellis endings, the rate R satisfies :

$$R = \frac{R_1 R_2}{R_1 + R_2 - R_1 R_2} \qquad = \frac{R_1}{2 - R_1} \quad \text{if} \quad R_1 = R_2 \tag{1.3}$$

Taking into account *both* the trellis endings, we have :

$$\begin{aligned} R &= \frac{R_1 R_2}{R_1 + R_2 - R_1 R_2 + \frac{n_1 T_1 + n_2 T_2}{N}} && (1.4) \\ &= \frac{R_1}{2 - R_1 + \frac{2 n_1 T_1}{R_1 N}} \quad \text{if} \quad R_1 = R_2 && (1.5) \end{aligned}$$

The most frequently used turbo codes have $J = 2$ levels, two identical RSC convolutional codes $\mathcal{C}^1 = \mathcal{C}^2$ and either $R = 1/3$ (with $R_1 = 1/2$), or $R = 1/2$ if every second parity bit of each constituent code of rate $R_1 = 1/2$ is punctured. It is also possible to avoid puncturing by directly using RSC codes of rate $R_1 = 2/3$ which however results in more complex trellises.

Some recent techniques allow us to avoid the loss (in terms of rate) due to the trellis endings, by the use of specific RSC convolutional codes and tail-biting.

1.1.2 Serial turbo codes

A serial turbo code $\mathcal{C}$ is also an (M, K) linear binary block code (where M is the length and K the dimension). It results from the serial concatenation of two convolutional codes $\mathcal{C}^1$ and $\mathcal{C}^2$, as depicted in Figure 1.2.

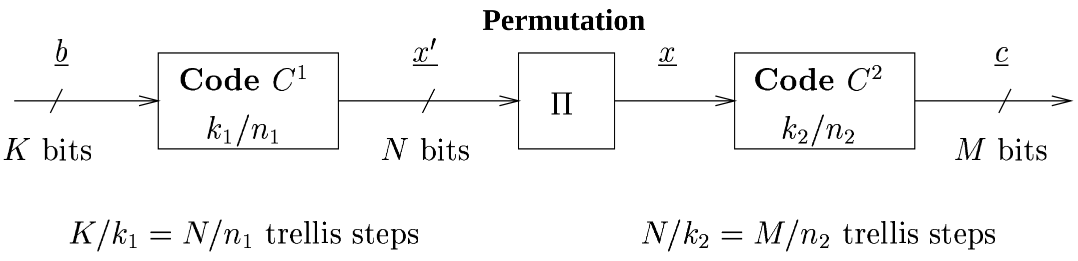


Figure 1.2: Encoder of a serial turbo code.

$\mathcal{C}^1$ is called the *outer* code and $\mathcal{C}^2$ the *inner* code. Similarly to a parallel turbo code, the trellises of both codes start and end at the zero state, after $K/k_1 + T_1$ trellis transitions for $\mathcal{C}^1$ and $N/k_2 + T_2$ for $\mathcal{C}^2$. The outer code can be chosen either systematic or not, but the inner code must be recursive systematic so as to make the iterative decoding efficiently work.

The K information bits b are encoded by $\mathcal{C}^1$ to produce $N = K/R_1$ bits x', which are interleaved by Π and encoded by $\mathcal{C}^2$ to produce $M = N/R_2$ coded bits c. Since $\mathcal{C}^2$ is systematic, c contains the permuted version of x.

Neglecting the endings of both trellises, the rate of the serial turbo code $\mathcal{C}$ is $R = R_1 \times R_2$. Taking them into account, it becomes[3] :

$$R = \frac{K}{M + n_1T_1 + n_2T_2} = \frac{R_1R_2}{1 + (n_1T_1 + n_2T_2)\frac{R_1R_2}{K}} \tag{1.6}$$

The conventional choice of constituent codes for obtaining an overall rate $R = 1/2$ is $R_1 = 2/3$ and $R_2 = 3/4$. Hence, for the same rate R, a serial turbo code uses constituent codes of higher rates and complexity than a parallel turbo code.

[3]we consider that the coded bits corresponding to the ending of the trellis of $\mathcal{C}_1$ go directly to the channel, and are not fed to the interleaver and the inner code.

1.2 Summary of analytical performance

In this section, we describe the properties of turbo codes as derived from their average weight distribution. The union bound on the average bit error probability of ML decoding is expressed as a function of the individual weight distributions of the constituent codes and the length N of the interleaver, for both cases of parallel and serial turbo codes. We discuss the results given in [14] [15] (parallel turbo codes) and [104] (serial turbo code).

1.2.1 Parallel turbo codes

We assume that the considered turbo code has $J = 2$ levels and that its RSC constituent codes $\mathcal{C}^1$ and $\mathcal{C}^2$ are identical, with rate $R_1 = R_2 = 1/2$. In the absence of puncturing, the rate of $\mathcal{C}$ is $R = 1/3$. We neglect the trellis endings. We define the IOWEF of $\mathcal{C}^1$ as :

$$A^{\mathcal{C}^1}(W,Z) = \sum_{\omega=\omega_{min}}^{N} \sum_{j=j_{min}}^{N} A^{\mathcal{C}^1}_{\omega,j} W^{\omega} Z^{j} = \sum_{\omega=\omega_{min}}^{N} W^{\omega} A^{\mathcal{C}_1}(\omega, Z) \qquad (1.7)$$

where

$$A^{\mathcal{C}^1}(\omega,Z) = \sum_{j=j_{min}}^{N} A^{\mathcal{C}^1}_{\omega,j} Z^{j} \qquad (1.8)$$

where $A^{\mathcal{C}^1}_{\omega,j}$ is the number of codewords of $\mathcal{C}^1$ where the input weight (of the information bits) is ω and where the weight of the parity bits is j. $A^{\mathcal{C}_1}(\omega, Z)$ is called the conditional IOWEF, since the input weight ω is fixed. The term 1 which corresponds to the all-zero codeword is excluded from the polynomial. Since the trellis of the RSC $\mathcal{C}^1$ starts and ends at the zero-state, $\omega_{min} = 2$. The minimum weight of the parity bits is denoted by j_{min}. It does not necessarily correspond to an input of weight $\omega_{min} = 2$. We define z_{min} as the minimal parity weight corresponding to an input of weight $\omega_{min} = 2$. Hence, $j_{min} \leq z_{min}$.

The average IOWEF $A^{\mathcal{C}}_{\omega,j}$ of the ensemble of parallel turbo codes (*i.e.* averaging over all the possible interleaver choices) can be expressed as :

$$A^{\mathcal{C}_P}(\omega, Z) = \frac{\left[A^{\mathcal{C}_1}(\omega, Z)\right]^2}{\binom{N}{\omega}} \qquad (1.9)$$

Methods to compute this function are given in [35]. Assuming a BPSK modulation and an AWGN channel, the conventional union bound on the bit error probability under the assumption of ML decoding results in :

$$P_{eb} \tilde{\leq} \frac{1}{2} \sum_{\omega=\omega_{min}}^{N} \frac{\omega}{N} W^{\omega} A^{\mathcal{C}_P}(\omega, Z) \quad \text{with} \quad W = Z = e^{-RE_b/N_0} \tag{1.10}$$

where the function erfc(x) has been upper bounded by $\frac{1}{2}e^{-x}$. This formula is often used to predict the performance of turbo codes although it suffers from two major drawbacks : it is not accurate and even diverges for low values of the SNR, and it does not reflect the iterative decoder performance, which is not an ML decoder. The first problem has been studied by Duman and Saheli [37] who proposed an improved bound based on the Gallager bound, and Sason and Shamai [92] [93] who proposed an even more accurate bound based on the tangential sphere bound [81]. These improved bounds are presented in the context of GLD codes in Section 4.2.5.

Defining a more detailed conditional IOWEF of the constituent codes that counts the number of *simple* error events[4] that constitute any error event, Benedetto and Montorsi [15] were able to prove that any parallel turbo code with NRNSC constituent codes does not exhibit *interleaver gain*, *i.e.* the error probability does not decrease as the interleaver length increases. In the case of RSC constituent codes, the error probability can be roughly expressed as :

$$P_{eb} \tilde{\leq} \frac{1}{2} \sum_{i=1}^{\lfloor N/2 \rfloor} \frac{2i}{N} \frac{(H^{2+2z_{min}})^i}{(1 - H^{z_{min}-2})^{2i}} \tag{1.11}$$

where $H = e^{-RE_b/N_0}$. Hence turbo codes built with RSC codes exhibit an interleaving gain of $1/N$, and an asymptotic approximation of their minimum Hamming distance is :

$$d_{\text{eff}} = 2 + 2z_{min} \tag{1.12}$$

Hence, z_{min} has to be maximized, which corresponds to the choice of a primitive feedback polynomial for the constituent codes.

1.2.2 Serial turbo codes

The same analysis has been carried out for serial turbo codes in [16]. We do not detail it here. Let d_1 be the minimum Hamming distance of $\mathcal{C}^1$, the *outer*

[4]A *simple* error event is a path in the trellis that diverges from the zero-state and returns to it only once.

constituent code, and d_2, the minimum distance of $\mathcal{C}^2$, the *inner* constituent code. The major results are :

- The inner code $\mathcal{C}^2$ must be RSC to produce an interleaving gain, which equals $1/N^{(d_1+1)/2}$. This value is higher than the interleaving gain of parallel concatenation. $\mathcal{C}^1$ is often chosen as NRNSC so as to maximize the gain.
- the effective minimum distance is roughly $d_{\text{eff}} = \frac{d_1 d_2}{2}$, which is also larger than that of parallel concatenation.

From this theoretical analysis, serial turbo codes should exhibit better performance than parallel turbo codes, at the cost of a higher complexity due to larger trellises.

1.3 Iterative decoding

1.3.1 Soft-input soft-output decoding of a convolutional code

We describe the principle of the SISO decoding of a convolutional code $\mathcal{C}^1$ of rate $R_1 = k_1/n_1$. This code starts and ends at the zero-state after $N/k_1 = M/n_1$ trellis steps. Hence, it can be considered as an (M, N) block code.

Let $\underline{c} = (c_1, \cdots, c_M)$ be a transmitted codeword of $\mathcal{C}^1$. Let $\underline{r} = (r_1, \cdots, r_M)$ be the vector of the received samples as the decoder input. The *a posteriori* probability of the i-th bit c_i is equal to :

$$\text{APP}(c_i) = \text{P}(c_i|\underline{r}) = \sum_{\underline{c} \in \mathcal{C}^1} \text{P}(c_i, \underline{c}|\underline{r}) = \sum_{\underline{c} \in \mathcal{C}^1} \frac{p(\underline{r}|c_i, \underline{c})\text{P}(c_i, \underline{c})}{p(\underline{r})} \tag{1.13}$$

The *a priori* probability $\text{P}(c_i, \underline{c})$ is zero if c_i does not correspond to the value of the i-th bit of the considered codeword $\underline{c}$. Furthermore, if we assume that **the *a priori* probabilities on the bits are independent** and that the channel is memoryless, the likelihood $p(\underline{r}|c_i, \underline{c})$ can be expressed as the product of the likelihoods of the symbols :

$$\mathrm{APP}(c_i) \propto \sum_{\underline{c} \in \mathcal{C}^1/c_i} p(\underline{r}|\underline{c})\mathrm{P}(\underline{c}) = \sum_{\underline{c} \in \mathcal{C}^1/c_i} \prod_{j=1}^{M} p(r_j|c_j)\pi(c_j) \tag{1.14}$$

where $\pi(c_j)$ is the *a priori* probability of the j-th bit of the codeword.

Any practical SISO decoder (such as the forward-backward algorithm, described in detail in Appendix B) computes the APP of the bits given their observations (their likelihoods) and their *a priori* probabilities. The SISO decoders used in turbo decoding (either parallel or serial) are fed with M observations[5], and either N or M *a priori* informations (on all the bits or only the information bits) depending on their availability. They can compute N or M APP and also deliver N or M updated *a priori* probabilities, called *extrinsic* informations.

The SISO decoder of a linear block code (M, K) can be implemented using other algorithms, but they all share the same inputs and outputs. This common structure is depicted in Figure 1.3

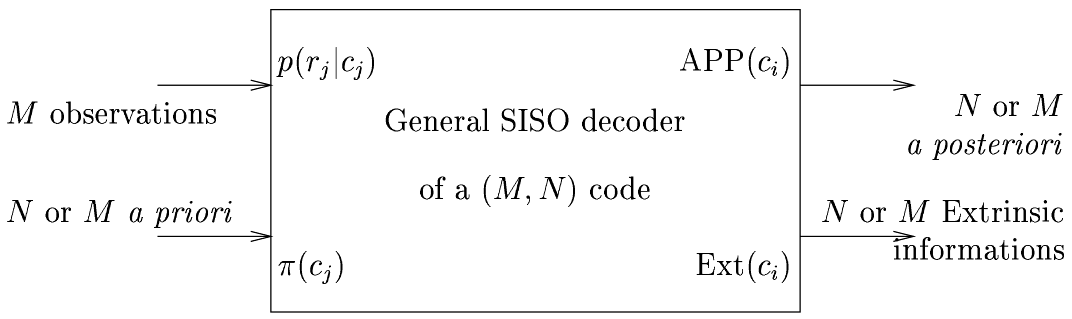


Figure 1.3: General SISO decoder scheme for a (M, N) code.

1.3.2 Iterative decoding of a parallel turbo code

Since a parallel turbo code combines two RSC convolutional codes, its decoding uses two instances of SISO decoder for an RSC code, cooperating

[5]If puncturing is implemented, the likelihood corresponding to any punctured bit is set to 1/2.

together. Hence, we first describe the quantities (APP and extrinsic informations) that the SISO decoder of an RSC code has to compute.

Let $(b_1, \cdots, b_N)$ be the information bits of the RSC code $\mathcal{C}^1$ and $(c_{N+1}, \cdots, c_M)$ its parity bits. Hence, the codeword $\underline{c}$ is constructed as follows :

$$\underline{c} = (b_1, \cdots, b_N, c_{N+1}, \cdots, c_M) \tag{1.15}$$

The SISO decoder receives the M observations, together with N *a priori* probabilities $\pi(b_l)$, $1 \leq l \leq N$ of the information bits. Since it has no *a priori* informations on the parity bits, their corresponding *a priori* probabilities are set to $1/2$. The decoders computes the APP and extrinsic probabilities of the information bits b_i, $1 \leq i \leq N$. The APP of b_i can be expressed as :

$$\begin{aligned} \text{APP}(b_i) &\propto \sum_{\underline{c} \in \mathcal{C}^1/b_i} \prod_{l=1}^{N} p(r_l|b_l)\pi(b_l) \prod_{j=N+1}^{M} p(r_j|c_j) \qquad (1.16) \\ &\propto p(r_i|b_i) \times \pi(b_i) \times \sum_{\underline{c} \in \mathcal{C}^1/b_i} \prod_{l=1, l\neq i}^{N} p(r_l|b_l)\pi(b_l) \prod_{j=N+1}^{M} p(r_j|c_j) \end{aligned}$$

The extrinsic probability of b_i is defined as the *novelty* introduced by the decoder, *i.e.* the part of the APP of b_i that was not known before decoding $\mathcal{C}^1$. Hence, we have :

$$\text{Ext}(b_i) \propto \prod_{l=1, l\neq i}^{N} p(r_l|b_l)\pi(b_l) \prod_{j=N+1}^{M} p(r_j|c_j) \tag{1.17}$$

These formulas are theoretical, and a practical decoder implements either the FB or any other optimal or suboptimal algorithm to compute them. Moreover, it often works in the log domain, and more precisely using the log-ratio (LR) defined as : $\text{LR}(\text{APP})(b_i) = \text{APP}(b_i = 1)/\text{APP}(b_i = 0)$, so as to simplify the computation and avoid that of normalization factors if the communication medium is modelled as an AWGN channel.

Two decoders, described by formulas (1.16) and (1.17), are used in the iterative decoding of a parallel turbo code with $J = 2$ RSC constituent codes $\mathcal{C}^1$ and $\mathcal{C}^2$. The extrinsic outputs of the first decoder (*i.e.* the novelty on the information bits computed by the SISO decoder of $\mathcal{C}^1$) are injected through an interleaver (which is identical to the one at the encoder part) to the *a priori* inputs of the second decoder. The extrinsic outputs of the latter are fed through an inverse interleaver to the *a priori* inputs of the first decoder. This process is iterated, starting with *a priori* probabilities $1/2$ at the first

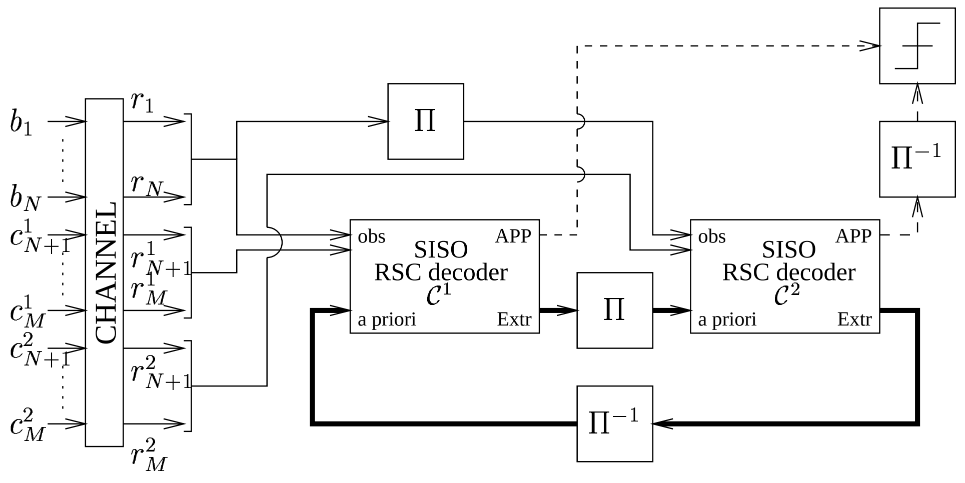


Figure 1.4: Iterative decoding of a parallel turbo code. Solid lines correspond to values that do not vary during the iterative process, bold lines to values that vary (extrinsic/*a priori* probabilities), and dashed lines to APP values used to decode the bits.

decoding of $\mathcal{C}^1$. This iterative decoding, named "turbo" decoding, is depicted in Figure 1.4.

After a fixed or variable number of iteration steps (a step consists of decoding a single constituent code), the last APP values are used to decide upon the bits.

1.3.3 Iterative decoding of a serial turbo code

We assume that the outer code $\mathcal{C}^1$ is an NRNSC code, equivalent to a (N, K) code, and that the inner code $\mathcal{C}^2$ is an RSC code, equivalent to a (M, N) code. The inputs of $\mathcal{C}^1$, *i.e.* the information bits, are denoted by $\underline{b} = b_1, \cdots, b_K$, and its outputs by $\underline{x'} = x'_1, \cdots, x'_N$. Since $\mathcal{C}^1$ is not systematic, the information bits are not part of $\underline{x'}$. The interleaved version $\underline{x}$ of $\underline{x'}$ is the input of $\mathcal{C}^2$, which is systematic. Hence, its output is : $\underline{c} = (x_1, \cdots, x_N, c_{N+1}, \cdots, c_M)$. The corresponding received symbols are denoted by $\underline{r} = (r_1, \cdots, r_M)$.

The SISO decoder of $\mathcal{C}^2$ is almost the same as the one used for parallel turbo codes. However, since, $\mathcal{C}^2$ does not directly handle the information bits $\underline{b}$, it is not necessary to compute the APP of $\underline{x}$. But, the extrinsic probabilities can be computed for all the bits in $\underline{x}$. We have for $1 \leq j \leq N$:

$$\begin{aligned}
\mathrm{APP}(x_j) &\propto \sum_{\underline{c} \in \mathcal{C}^2/x_j} p(\underline{r}|\underline{c})\mathrm{P}(\underline{c}) && (1.18)\\
&\propto \sum_{\underline{c} \in \mathcal{C}^2/x_j} \prod_{l=1}^{N} p(r_l|x_l)\pi(x_l) \prod_{m=N+1}^{M} p(r_m|c_m) && (1.19)\\
&\propto p(r_j|x_j) \times \pi(x_j) \times \sum_{\underline{c} \in \mathcal{C}^2/x_j} \prod_{l=1,l\neq j}^{N} p(r_l|x_l)\pi(x_l) \prod_{m=N+1}^{M} p(r_m|c_m)
\end{aligned}$$

The corresponding extrinsic information is :

$$\mathrm{Ext}(x_j) \propto \sum_{\underline{c} \in \mathcal{C}^2/x_j} \prod_{l=1,l\neq j}^{N} p(r_l|x_l)\pi(x_l) \prod_{m=N+1}^{M} p(r_m|c_m) \qquad (1.20)$$

This single equation defines the SISO decoder of $\mathcal{C}^2$. It computes the N extrinsic probabilities corresponding to the "middle" bits, given the all the observations and *a priori* probabilities of the middle bits.

The SISO decoder of $\mathcal{C}^1$ works differently. The observations corresponding to its coded bits $\underline{x'}$ are directly the channel outputs properly de-interleaved, since $\mathcal{C}^2$ is systematic. However, $\mathcal{C}^1$ is not systematic, and hence the APP of the information bits is now different. On the other hand, the extrinsic informations it has to compute correspond to its coded bits $\underline{x'}$. We have, for $1 \leq i \leq K$:

$$\mathrm{APP}(b_i) \propto \sum_{\underline{c} \in \mathcal{C}^1/b_i} \prod_{l=1}^{N} p(r_l|x'_l(b_i))\pi(x'_l(b_i)) \qquad (1.21)$$

where $x'_l(b_i)$ is the l-th bit of $\underline{c}$, the codeword of $\mathcal{C}^1$ that corresponds to a sequence of information bits $\underline{b}$ with value b_i at position i.

We also have :

$$\begin{aligned}
\mathrm{APP}(x'_j) &\propto \sum_{\underline{c} \in \mathcal{C}^1/x'_j} \prod_{l=1}^{N} p(r_l|x'_l)\pi(x'_l) && (1.22)\\
&\propto p(r_l|x'_j) \times \pi(x'_j) \times \sum_{\underline{c} \in \mathcal{C}^1/x'_j} \prod_{l=1,l\neq j}^{N} p(r_l|x'_l)\pi(x'_l)
\end{aligned}$$

so the extrinsic probability of the coded bits x'_j is :

$$\mathrm{Ext}(x'_j) \propto \sum_{\underline{c} \in \mathcal{C}^1/x'_j} \prod_{l=1,l\neq j}^{N} p(r_l|x'_l)\pi(x'_l) \qquad (1.23)$$

Equations (1.21) and (1.23) define the decoder of the NRNSC $\mathcal{C}^2$. The structure of the turbo decoder of a serial turbo code is represented in Figure 1.5.

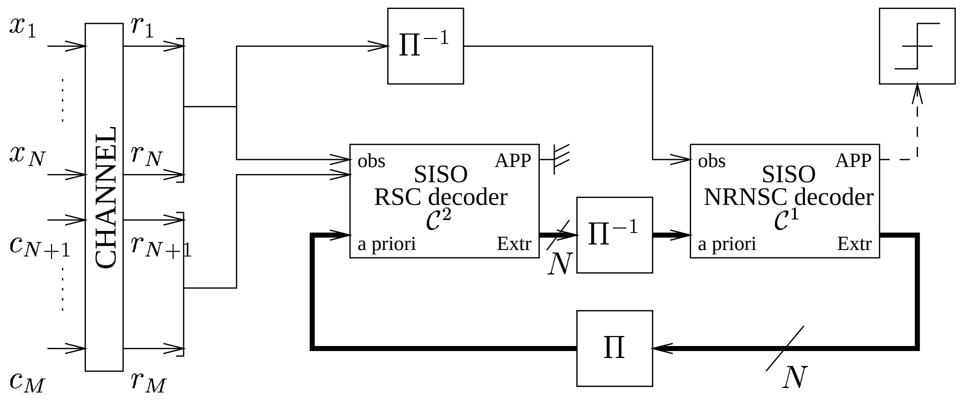


Figure 1.5: Iterative decoding of a serial turbo code. Solid lines correspond to values that do not vary during the iterative process, bold lines to values that vary (extrinsic/*a priori* probabilities), and dashed lines to APP values used to decode the bits.

Important remark : In either the parallel or serial turbo decoding, we were able to decompose the APP formula by assuming that the different *a priori* probabilities are independent. The interleaver, which separates the encoders in both schemes, has this role : the extrinsic probabilities at the output of any SISO decoder are correlated. They are fed to the next SISO decoder as *a priori* information through the interleaver. The latter breaks the dependencies in the extrinsic stream.

1.4 Simulations results

In this section, we present some results on the turbo codes performance. Figure 1.6 plots computer simulations of the performance of a parallel turbo code over the AWGN channel (BPSK modulation). The code has two identical constituent code with generator polynomial $(37, 21)$ and a randomly chosen interleaver of length $N = 65536$. The iteration involved 20 steps. We can clearly see the "error floor" of parallel turbo codes, due to the comparatively small minimum Hamming distance of these codes.

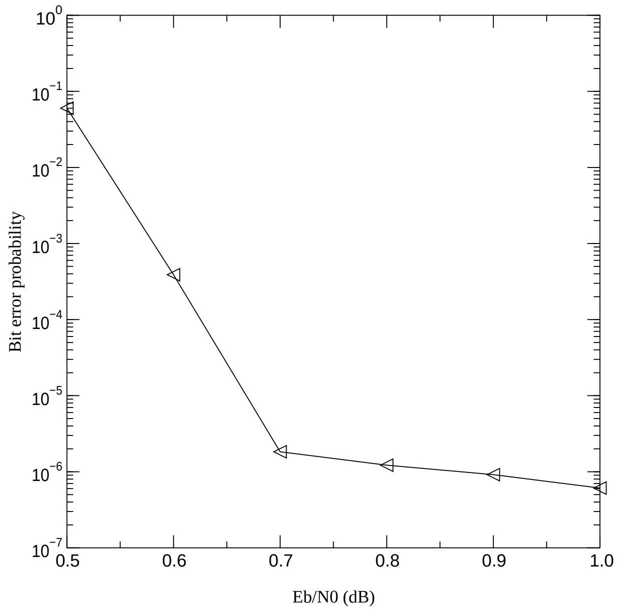


Figure 1.6: Performance of a parallel turbo code with $J = 2$ identical RSC codes $(37, 21)$, overall rate $R = 1/2$ and $N = 65536$.

Figure 1.7 compares the ML union bound (1.9) with the simulation results of turbo decoding of a parallel turbo code with $J = 2$ identical RSC codes $(7, 5)$, overall rate $R = 1/2$ and $N = 400$. We observe that the ML bound diverges for low values of the SNR, and that, in this case, the iterative decoding is very close to the *upper* bound ML decoding where it is significant. However, this is not always true, because iterative decoding does not implement ML decoding, and the bound is computed for the average ensemble of turbo codes generated by random interleaving, whereas the simulation is performed for a specific interleaver which can be better or worst than the average. In the context of GLD codes, we present a case where, for a specific interleaver choice, the iterative decoding performance is better than the ML union bound (see Figure 4.27).

Figure 1.8 presents the same type of comparison for a serial turbo code of overall rate $R = 1/2$. The outer code $\mathcal{C}^1$ is NRNSC, with rate $R = 2/3$ and polynomials $(236, 155, 337)$.The outer code $\mathcal{C}^1$ is RSC, with rate $R = 3/4$ and polynomials $(23, 25, 33, 37)$. The length of the interleaver is $N = 600$, and the length of the turbo code $M = 800$. The performance of iterative decoding is here very far from the ML union bound performance. It appears from our simulation results that the iterative decoding performs better (*i.e.*

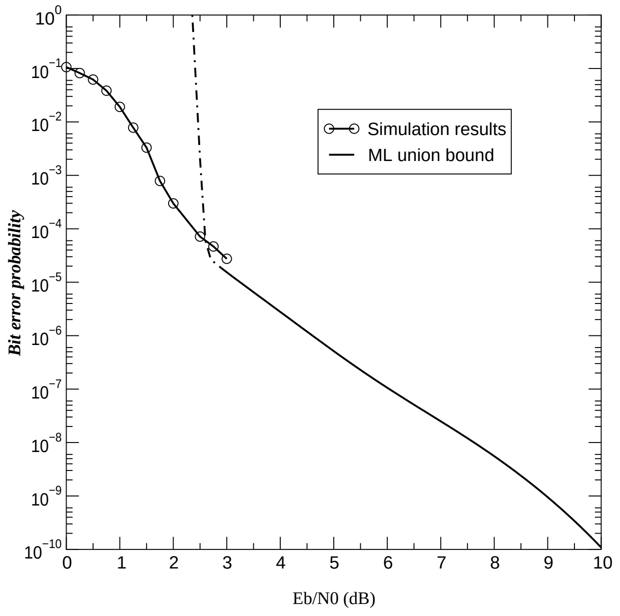


Figure 1.7: Comparison between simulation of turbo decoding and ML union bound for a parallel turbo code with $J = 2$ identical RSC codes $(7, 5)$, overall rate $R = 1/2$ and $N = 400$.

is closer to the ML UB) for parallel turbo codes than for serial ones.

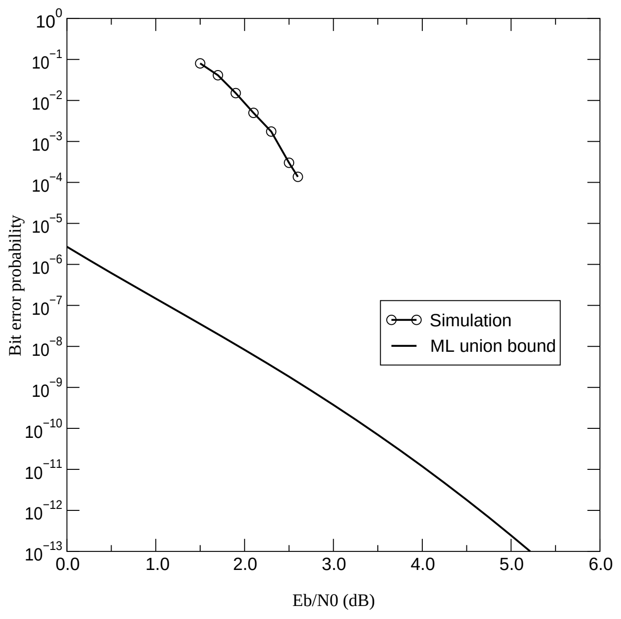


Figure 1.8: Comparison between simulation of turbo decoding and ML union bound for a serial turbo code. Code length : 800, rate : $R = 1/2$. See text for the other parameters.

Chapter 2

Iterative decoding convergence of turbo codes*

This chapter introduces a special graph representation of turbo codes[1], called the *dependency graph*, which is useful to understand the iterative decoding process as propagating probabilities along this graph. It also gives information on the number of iteration steps needed for convergence and checks the efficiency of the interleaver separating the two constituent codes.

2.1 Graphical representation of turbo codes

Graphical representation of error control codes dates back to Gallager (1963) [47] and has been studied by authors such as Tanner [97]. The graph theory is gaining more and more interest in the coding community, especially after the recent works of Sipser and Spielman [95], Wiberg, Loeliger and Kötter [105] [106], Frey and Kschischang [44] [45] [61]. Indeed, the latter shows a single graphical model framework that can represent compound codes like turbo codes [22], serially concatenated codes [17] [18], Gallager's Low Density Parity Check (LDPC) codes and product codes in a unified manner.

*Parts of this chapter have been presented at the Canadian Workshop on Information Theory, 1997 and also submitted to IEEE Trans. on Information Theory.

[1]In this Chapter, "turbo code" refers only to *parallel* concatenated recursive systematic convolutional codes.

In [44], Frey and Kschischang observed that the iterative decodings of compound codes are instances of probability propagation algorithms that operate on a graphical model of the codes. Pearl's "belief propagation" algorithm [78] which operates on graphs and has been developed in the context of expert systems is strongly connected with Gallager's decoding algorithm of LDPC codes [73] and turbo decoding [76].

In this section, we introduce a graphical representation of turbo codes, based on their Bayesian network. A Bayesian network is a directed acyclic graph. The paradigm of channel coding can be represented by a Bayesian graph, as shown in Figure 2.1. The information bits are denoted by $\mathbf{u}$, the encoder states by $\mathbf{s}$ (if there are any), the encoder outputs by $\mathbf{x}$ and the channel output by $\mathbf{y}$. The edges are oriented so as to indicate the causality : the states $\mathbf{s}$ are directly driven by the information bits $\mathbf{u}$, and the encoder output $\mathbf{x}$ is jointly influenced by the states $\mathbf{s}$ and the inputs $\mathbf{u}$.

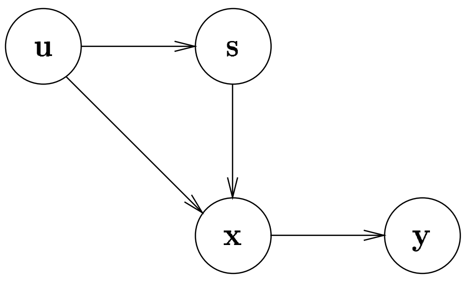


Figure 2.1: Bayesian network of a general channel code.

The relationship described by a directed edge between a "parent" p and a "child" c is usually the conditional probability $P(c|p)$. Hence, the joint distribution of all the variables in the network can be expressed from the parent-child probabilities by following the edges. For the channel decoding paradigm, we have :

$$P(\mathbf{u},\mathbf{s},\mathbf{x},\mathbf{y}) = P(\mathbf{u})P(\mathbf{s}|\mathbf{u})P(\mathbf{x}|\mathbf{u},\mathbf{s})P(\mathbf{y}|\mathbf{x}) \tag{2.1}$$

This kind of global joint probability can always be written if the graph does not contain any directed cycle.

Kschischang and Frey [61] presented a general algorithm that is used to compute marginal probabilities in graphical models by *propagating the probabilities*. Applied to Bayesian networks, it gives Pearl's belief propagation algorithm. The maximum *a posteriori* (MAP) decoding is an instance of the latter when applied to the general network depicted in Figure 2.1. It consists of computing the *a posteriori* probabilities (APPs) $p(u_i|\mathbf{y})$. The authors showed that the APPs are computed thanks to a marginalization of (2.1) performed by probability propagation along the graph.

Figure 2.2 shows the Bayesian network of a convolutional code. The numerical indices represent the time. We copied the information bits into the states vertices to simplify the graph. We can further simplify it by merging the three vertical vertices u, (u, s) and (x, y) into a *super-vertex*. The final network is a simple chain.

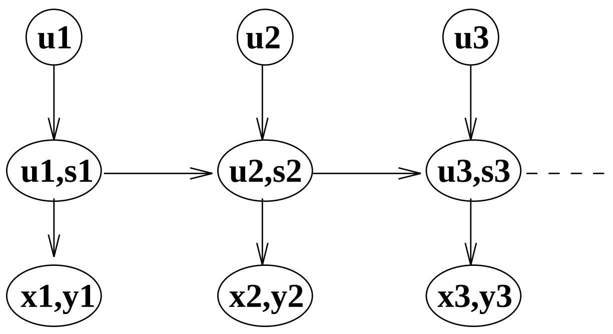


Figure 2.2: Bayesian network of a convolutional code.

The conventional MAP decoding algorithm of convolutional codes (the forward-backward algorithm [3]) is graphically equivalent to a forward propagation of the probabilities along the code chain, followed by a backward propagation. It guarantees that all the observations are distributed over all the vertices and that the APPs are exactly computed. It is depicted in Figure 2.3. Briefly and without introducing any notation, the probabilities propagate from the channel output vertices to the states vertices in the first step. The next two steps correspond to the forward and backward recursions. The last step propagates the probabilities to the information bit vertices.

For simplicity's sake, we assume that a turbo code is made of only $J = 2$ constituent convolutional codes linked by an interleaver of size N. Its Bayesian network can be drawn as follows.

Each convolutional code is represented by a simple chain with N vertices and $N-1$ edges. A vertex is a super-node containing the information bit, the

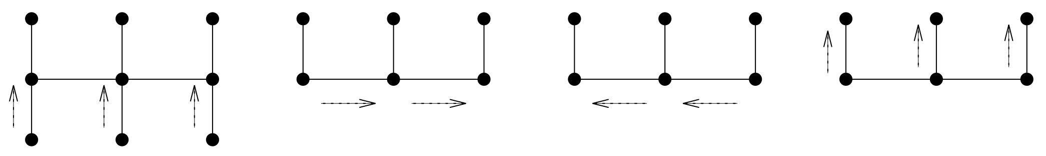


Figure 2.3: Belief propagation algorithm in a convolutional code network.

encoder state and output, and the channel output, as previously stated. The network of the turbo code is obtained by linking the two chains representing each of the two constituent codes by a random matching (the interleaver). The example of a Bayesian network for a turbo code with interleaver size $N = 16$ is shown in Figure 2.4. Wiberg *et al.* [105] [106] were the first to

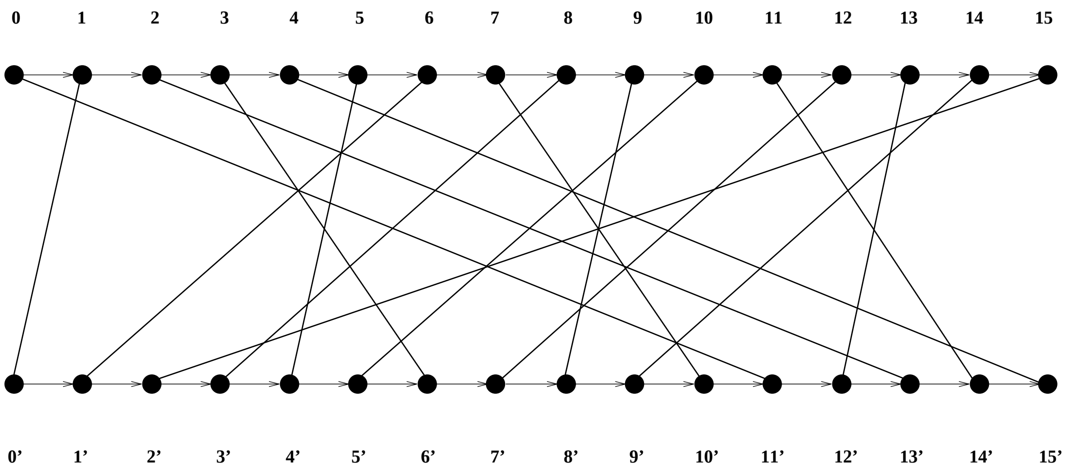


Figure 2.4: Bayesian network of a turbo code with interleaver size $N = 16$.

introduce this graphical representation of turbo codes. We do not orient the edges representing the interleaver since they do not introduce any causality.

The soft-snput soft-sutput (SISO) iterative decoding of turbo codes can be easily explained using this graph. The first decoding step consists of appling the forward-backward (FB) algorithm to the upper constituent code. It gives estimates of the distributions of the information bits given the observations related to the upper code. The information (more precisely the *extrinsic* probability in the turbo literature, or the *parent-child* probability in the probability propagation literature) associated with each bit u_i is passed to the second chain by the edges representing the interleaver. The same FB algorithm is then applied to the lower chain in a second step, taking into account the observations related to the lower code as well as those coming from the first decoding step. The extrinsic probabilities computed at this step are passed through the interleaver edges to be used by the upper decoder together with its own observations at the next iteration step.

However, this decoding procedure is not *stricto sensu* the belief propagation algorithm. Indeed, the network has cycles (*e.g.* $(7, 8, 9, 8', 9', 10', 7)$) and the probabilities propagation is only an *approximation* for computing them. We use an efficient probability propagation algorithm (the FB algorithm) for each constituent code, but we ignore the cycles.

The harmful influence of a cycle can be easily understood using a simple example derived from [44], the *army network*[2]. Let us assume an army, composed of a general in his headquarters (HQ) and a hierarchical structure of soldiers. It is depicted as a Bayesian network in Figure 2.5. Each soldier (including the general) is represented by a vertex, and each hierarchical relationship by an edge.

The first army to be considered has no cycle in its network. The task to be achieved is to let the headquarters know the total number of soldiers, and then to propagate this information to all the soldiers. This is depicted in figure 2.7.

In a first step, each soldier s (beginning from the leaves of the network) transmits to his superior $Sup(s)$ the sum of the number of soldiers he received from his subordinates $Sub_i(s)$ plus one (himself). This is depicted in Figure 2.6. At the end of this step, the general counts $6+4+1$ soldiers plus himself, *i.e.* 12 soldiers.

In a second step, each soldier s (beginning with the general) transmits to each of his subordinates $Sub_i(s)$ the number of soldiers he received from his superior $Sup(s)$, plus the sum of the number of soldiers that all the other subordinates $Sub_j(s)$, $j \neq i$ gave him[3] in the first step plus one (himself)[4]. At the end of this second step, each soldier knows the exact number of soldiers in the army by adding the number he sent in the first step with the number he received in the second step.

[2]this example is closer to the iterative SISO decoding of turbo codes than it may look at first glance :

[3] – it is exactly the notion of *extrinsic* observation used as an *a priori* in the next step,

[4] – it corresponds to his own observation.

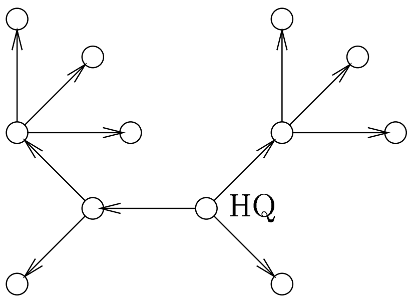


Figure 2.5: Army network without cycle.

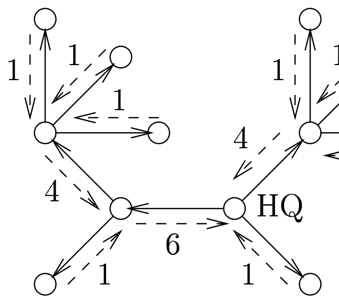


Figure 2.6: Number of soldier count. First step.

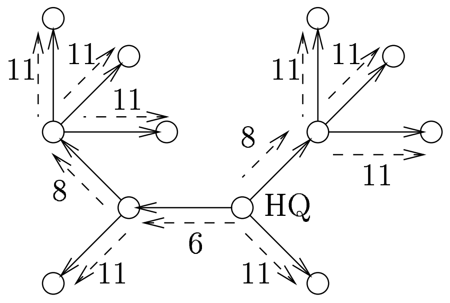


Figure 2.7: Number of soldier count. Second step.

The network of the second army presents a cycle, as depicted in Figure 2.8. The same information propagation algorithm is applied to this network.

In the first step, the soldier represented by a black vertex reports twice to his superiors. Hence, at the end of this step, the number of soldiers counted by the general is $6 + 4 + 1 + 1$ plus himself, that is, 13. This error occurs because the same information (that from the black vertex) propagates by two different paths to the general.

The erroneous number of soldiers propagates to the whole army during the second step of the algorithm. The error would have been greater if the black vertex were not a leaf, since all the information coming to it would have been propagated twice. Furthermore, how would he calculate the total number of soldier if the values computed from its two edges gave different results ?

Using the SISO iterative decoding for turbo codes means choosing to ignore the presence of cycles. If the network had no cycles, the belief propagation algorithm would stop when each vertex has been processed once. In fact, because the turbo codes have cycles, the propagation procedure never self-terminates.

Hopefully, the situation is not so desperate, and furthermore, the practical implementation of turbo decoding gives astonishingly good results. This leads to question the influence of cycles on iterative decoding. We partially answer in the following sections, and answer in Chapter 5 as regards generalized low density (GLD) codes.

2.2 Dependency graph

As previously stated, if the turbo code Bayesian network had no cycles, the iterative turbo decoder would be an *exact* MAP decoder, and would converge in a fixed number of iteration steps. By construction, the presence of cycles in the network can however not be avoided.

Intuitively, large cycles have a smaller influence on the decoding process. This is confirmed by the performance of improved interleavers for turbo codes, such as the S-random interleaver developed by Dolinar and Divsalar

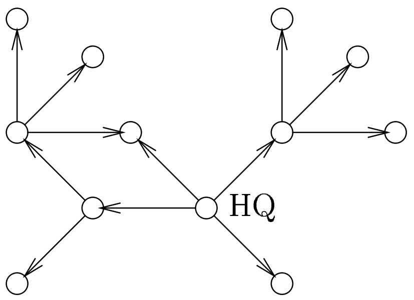


Figure 2.8: Army network with a cycle.

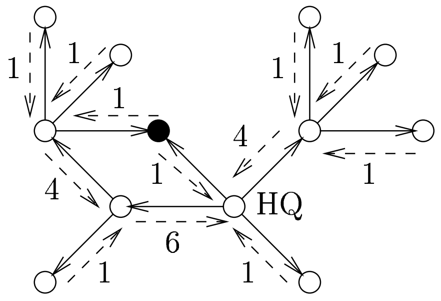


Figure 2.9: Erroneous umber of soldier count. First step.

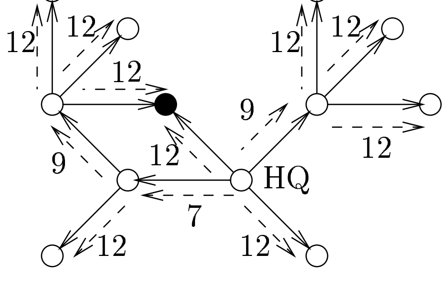


Figure 2.10: Erroneous number of soldier count. Second step.

[36] : since for any two information bits u_i and u_j, their distances d_1 in the upper chain and d_2 in the lower chain satisfy : $d_1 + d_2 \geq S + 1$, the length of the minimal cycle[5] in the graph depicted in Figure 2.4 is lower-bounded by $S + 3$.

We will construct a graph, called the dependency graph, that allows us to identify the cycles and to analyze the convergence of turbo code iterative decoding.

We can distinguish two types of edges in the network of a Turbo code : *strong* and *weak* links. An edge linking two vertices via the interleaver is called a *strong* link. An edge linking two vertices via the convolutional code chain is called a *weak* link. When propagating through strong links, the information is never attenuated. On the contrary, each weak link acts as a "forgetting factor" and the information is faded when passing through it. This notion of weak link is supported by both theory and empirical results.

We prove that the observation (and the *a priori* information) associated with a vertex in the convolutional code chain will not propagate indefinitely, so it has an influence limited by the constraint length to neighboring nodes.

Our aim is to find the "propagation limit" of the observations in the forward-backward algorithm. For a complete description of this algorithm, and the definitions of the relevant quantities, please refer to Appendix B.

Starting from time $t + 1$, let us assume that no more observations and *a priori* informations are available. We look for time τ when the forward recursion of the algorithm becomes stable, i.e. $\alpha_{t+\tau}(m) = K, \forall m \in [1..M]$, where K is a constant value and M is the number of states in the trellis. Letting $j \geq 1$, we have :

$$\begin{aligned}\alpha_{t+j}(m) &= \text{Prob}(S_{t+j} = m, Y_1^{t+j}) = \text{Prob}(S_{t+j} = m, Y_1^t) \\ &= \sum_{m'=1}^{M} \text{Prob}(S_{t+j-1} = m', Y_1^t).\text{Prob}(S_{t+j} = m | S_{t+j-1} = m', Y_1^t)\end{aligned}$$

Given the state S_{t+j-1}, S_{t+j} is independent from the previous observations, so we can write :

$$\alpha_{t+j}(m) = \sum_{m'=1}^{M} \underbrace{\text{Prob}(S_{t+j-1} = m', Y_1^t)}_{\alpha_{t+j-1}(m')}.\text{Prob}(S_{t+j} = m | S_{t+j-1} = m') \quad (2.2)$$

[5]It is called the *girth* of the graph.

Assuming (without loss of generality) the convolutional code rate to be $1/n$, we have since no *a priori* probabilities are available :

$$\text{Prob}(S_{t+j} = m | S_{t+j-1} = m') = \frac{1}{2} B(m', m) \tag{2.3}$$

where $B(m', m) = 1$ if there is a transition $m \to m'$, otherwise $B(m', m) = 0$. Then we have :

$$\alpha_{t+j}(m) = \frac{1}{2} \sum_{m'=1}^{M} \alpha_{t+j-1}(m').B(m', m) \tag{2.4}$$

By iterating downward to zero on j, we obtain :

$$\alpha_{t+j}(m) = \frac{1}{2^j} \sum_{m_1=1}^{M} \sum_{m_2=1}^{M} \cdots$$

$$\sum_{m_j=1}^{M} \alpha_t(m_1) B(m_1, m_2) B(m_2, m_3) \cdots B(m_j, m)$$

Let $B^{(2)}(m', m) = \sum_{m''=1}^{M} B(m', m'') B(m'', m)$ and let us define recursively $\forall u > 2$, $B^{(u+1)}(m', m) = \sum_{m''=1}^{M} B(m', m'') B^{(u)}(m'', m)$, we can rewrite $\alpha_{t+j}(m)$ as follow :

$$\alpha_{t+j}(m) = \frac{1}{2^j} \sum_{m'=1}^{M} B^{(j)}(m', m) \alpha_t(m') \tag{2.5}$$

where $B^{(j)}(m', m)$ denotes the number of paths in the trellis which connect the states m and m' in exactly j steps.

Obviously, the smallest j for which $B^{(j)}(m', m)$ becomes constant for all couples (m, m') is the memory $\nu = L - 1$ of the encoder. Thus, let K' be that constant value. We can write :

$$\alpha_{t+\nu}(m) = \frac{K'}{2^\nu} \sum_{m'=1}^{M} \alpha_t(m') = K, \ \forall m \in [1..M] \tag{2.6}$$

The same computation holds for the backward recursion, considering the β's. Hence, we conclude that an *isolated* information propagates in the trellis in a window of width $W = 2\nu + 1$.

However, in a real decoding scheme, the observations and *a priori* informations are not isolated so the previous result gives us only an order of magnitude. The effective window size W of the influence of informations can be estimated by simulation. We transmit over the AWGN channel with BPSK modulation the coded bits of a convolutional code, seen as a block code since we terminate the trellis. At the decoder side, we implement the FB algorithm on the whole block length, but we are only interested in the APP value of the central information bit. We truncate the observations to a window of width W centered on the middle symbol. All other observations are erased. Figure 2.11 shows the results in terms of BER for a recursive systematic code of memory $\nu = 4$ and generator polynomials $(1, 35/23)$.

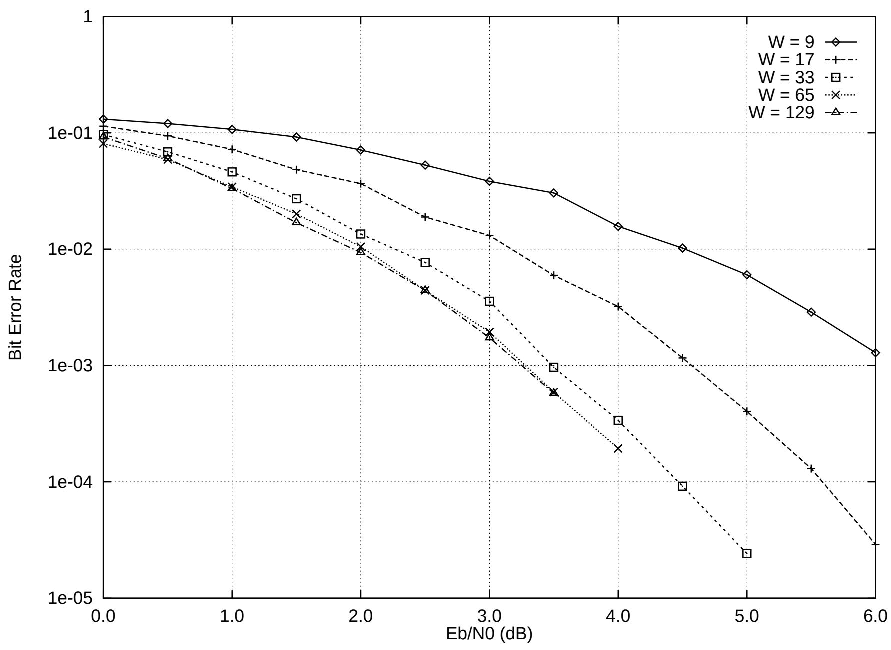


Figure 2.11: Bit Error Rate of the central information bit of a recursive systematic convolution code $(1, 35/23)$ for different observations window size W.

We observe that truncating the observations up to a window of width $8\nu + 1$ almost does not affect the results in terms of BER. Truncation to narrower widths $((4\nu + 1)$ or $(2\nu + 1))$ lead to significant degradations.

Even if these observations are useful and can lead to an improved implementation of the FB algorithm (by the use of a sliding observation window of appropriate length), it is not completely accurate in the context of SISO iterative decoding, since we do not take hard decisions at the output of the constituent code decoder, but we transmit extrinsic values to the other decoder. However, finding a valid criterion for the degradation introduced by an observation truncation in terms of extrinsic probabilities is not an easy task, and further investigations need to be carried out to find a more precise way for characterizing the influence of the observation width W.

However, we can take into account the *weakness* of the link introduced by the convolutional codes between vertices by drawing a *dependency clump* of size W around each vertex, and by stating that, as a first approximation, a vertex does not depend on other vertices outside its clump. As already stated, the value of W is not exactly known.

We are now able to draw a graph called *dependency graph*, denoted by $\mathrm{DG}(u_i)$, for each information bit u_i, $i = 1 \dots N$. This graph stems from vertex u_i at level 1. For any vertex at any level, taking into account the information attenuation in each constituent code, we consider the clump of size W containing the vertex and its $W - 1$ neighbors. Then, we consider a new level by following the strong links of each vertex in all clumps using the following additional rules :

- We discard any vertex already present at any preceding level (this case is represented by a cross in Figure 2.12).
- We do not follow the strong link from the center of any clump (it would always give its parent vertex which is already used) except at the first level where a bit b_i has no parent.

This construction leads to Figure 2.12 drawn for the bit u_8 of the turbo code represented in Figure 2.4, where we consider a dependency clump of length $W = 3$.

The dependency graph is thus organized in levels, each level corresponding to 1/2 iteration step of the turbo decoding. Vertices of the same level are linked together using weak links only. The strong links connect vertices belonging to two adjacent levels. Iterative SISO decoding of the bit u_i can be interpreted as follows : the forward-backward algorithm is applied on the population of each level, starting from the lowest one and propagating upward until it reaches the first one containing the information bit u_i.

We can further note that *cycles* are present in the dependency graph $DG(u_i)$, as soon as a "clump collision" occurs (as shown in Figure 2.12 at

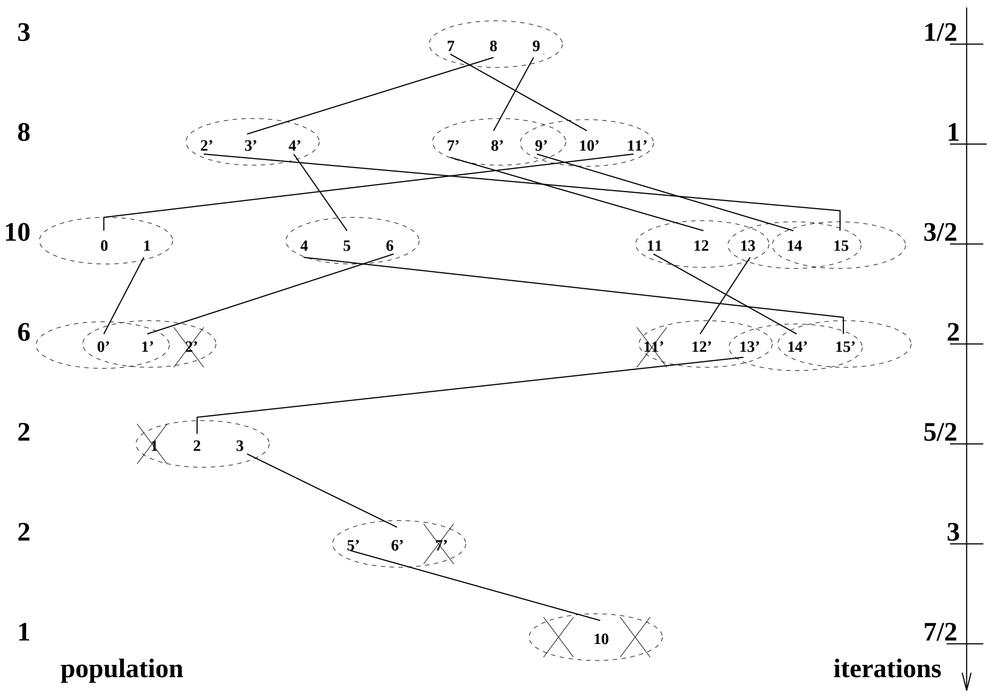


Figure 2.12: Dependency graph DG(u_8) for the turbo code depicted in Figure2.4

level 2 for vertex $9'$) or as soon as we eliminate a vertex. The influence of the dependency graph height and cycles on the performance of turbo decoding is addressed in the next section.

2.3 Convergence analysis

We introduce two quantities related to the dependency graph that are linked to the iterative decoding, namely the *height* of the graph, and the level where a cycle closes itself for the *first time*. Since the dependency graphs of a turbo code depend almost only on its interleaver (only the memory of the constituent codes is involved in the dependency graph through the clump size W, and furthermore in an unclear way), dependency graphs provide us a tool to analyze the influence of the interleaver in a turbo coding scheme.

2.3.1 Height of the dependency graph

The height of the dependency graph DG(u_i) gives the maximum number of iteration steps to reach iterative decoding convergence[6] for bit b_i. Hence, the maximum number of iteration steps necessary to reach total convergence is obtained by finding the dependency graph of maximal height. The average number of iteration steps can be computed ignoring the levels with a negligible population.

Height of theoretical dependency graphs without cycles

If there are no clump collisions, the theoretical height of any dependency graph can be calculated theoretically. Figure 2.13 shows this theoretical dependency graph for a clump length of $W = 3$.

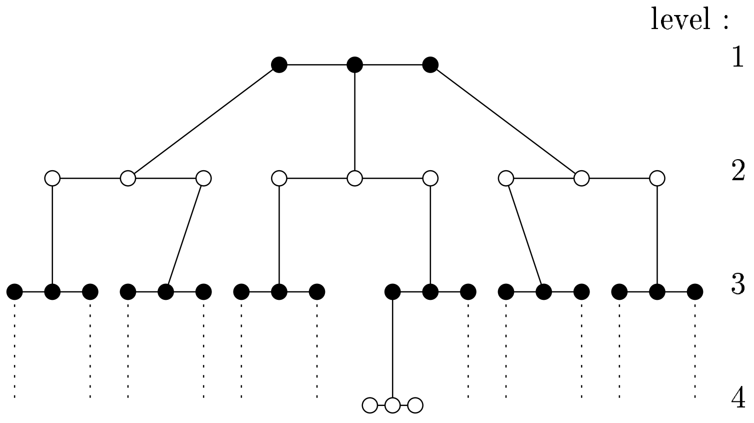


Figure 2.13: Dependency graph without clump collision. $W = 3$.

The first level has W vertices, and W strong links will descend to the second level. The second level has W^2 vertices, and $W(W-1)$ strong links will descend to the third level. The third level has $W^2(W-1)$ vertices, and $W(W-1)^2$ strong links will descend from it. Generally, the level i has $V(i) = W^2(W-1)^{i-2}$ vertices, and $E(i) = W(W-1)^{i-1}$ descending strong links. The black vertices belong to the upper convolutional chain (odd levels)

[6]We define the *convergence* as the number of iteration steps for which the APP of a bit b_i is computed given all the observations at the channel output.

and the white vertices belong to the lower convolutional chain (even levels), if the top vertex b_i belongs to the upper chain[7].

We can count the total number of vertices in this graph, in order to estimate its height h. The first estimate results from considering the total number of vertices that are present in the graph, without distinguishing the upper and lower vertices, that is $2N$. The total number of vertices at levels strictly smaller than h is strictly lower than $2N$, and the theoretical total number of vertices at level h is greater than, or equal to, $2N$. We have :

$$2N \left\{ \begin{array}{ll} > & W + W^2 + W^2(W-1) + W^2(W-1)^2 + \cdots + W^2(W-1)^{h-3} \\ \leq & W + W^2 + W^2(W-1) + W^2(W-1)^2 + \cdots + W^2(W-1)^{h-2} \end{array} \right. \tag{2.7}$$

which leads to :

$$1{+}\log_{W-1}\left[1 + \frac{(2N-W)(W-2)}{W^2}\right] \leq h < 2{+}\log_{W-1}\left[1 + \frac{(2N-W)(W-2)}{W^2}\right] \tag{2.8}$$

A more accurate estimation of h takes separately into account the N upper and N lower vertices.

even height

Let us first assume that the height of the dependency graph is even. The number of upper vertices can be bounded by counting the number of vertices in the even levels :

$$N \left\{ \begin{array}{ll} > & W^2 + W^2(W-1)^2 + \cdots + W^2(W-1)^{h-4} \\ \leq & W^2 + W^2(W-1)^2 + \cdots + W^2(W-1)^{h-2} \end{array} \right. \tag{2.9}$$

which leads to :

$$\log_{W-1}\left[1 + \frac{N(W-2)}{W}\right] \leq h < 2 + \log_{W-1}\left[1 + \frac{N(W-2)}{W}\right] \tag{2.10}$$

Bounding the number of lower vertices gives :

$$N \left\{ \begin{array}{ll} > & W + W^2(W-1) + W^2(W-1)^3 + \cdots + W^2(W-1)^{h-5} \\ \leq & W + W^2(W-1) + W^2(W-1)^3 + \cdots + W^2(W-1)^{h-3} \end{array} \right. \tag{2.11}$$

which leads to :

$$2{+}\log_{W-1}\left[1 + \frac{(N-W)(W-2)}{W(W-1)}\right] \leq h < 4{+}\log_{W-1}\left[1 + \frac{(N-W)(W-2)}{W(W-1)}\right] \tag{2.12}$$

[7]We always draw dependency graphs starting from an upper vertex.

odd height

Let us now assume that the height of the dependency graph is odd. The number of upper vertices can be bounded by counting the number of vertices in the odd levels :

$$N \left\{ \begin{array}{ll} > & W + W^2(W-1) + W^2(W-1)^3 + \cdots + W^2(W-1)^{h-4} \\ \leq & W + W^2(W-1) + W^2(W-1)^3 + \cdots + W^2(W-1)^{h-2} \end{array} \right. \tag{2.13}$$

which leads to :

$$1+\log_{W-1}\left[1+\frac{(N-W)(W-2)}{W(W-1)}\right] \leq h < 3+\log_{W-1}\left[1+\frac{(N-W)(W-2)}{W(W-1)}\right] \tag{2.14}$$

Bounding the number of lower vertices gives :

$$N \left\{ \begin{array}{ll} > & W^2 + W^2(W-1)^2 + \cdots + W^2(W-1)^{h-5} \\ \leq & W^2 + W^2(W-1)^2 + \cdots + W^2(W-1)^{h-3} \end{array} \right. \tag{2.15}$$

which leads to :

$$1+\log_{W-1}\left[1+\frac{N(W-2)}{W}\right] \leq h < 3+\log_{W-1}\left[1+\frac{N(W-2)}{W}\right] \tag{2.16}$$

These bounds have been calculated for the two different block sizes $N = 65536$ and $N = 1024$, and odd values of W from 3 to 9. The tightest values (between (2.8), (2.10) / (2.12), and (2.14) / (2.16)) are summarized in Tables 2.1 and 2.2.

$W = 3$	$15.83 < h \leq 16.41$
$W = 5$	$8.63 < h \leq 8.97$
$W = 7$	$7.00 < h \leq 7.30$
$W = 9$	$6.21 < h \leq 6.48$

Table 2.1: Bounds on the height of a cycle-free dependency graph with $N = 65536$

At a fixed length N, these bounds are clearly decreasing functions[8] of W. Hence, the smallest value of W, *i.e.* $W = 3$, leads to the highest dependency graph. Its height is an upper bound on the number of decoding iterations needed to reach convergence for a code that presents cycle-free dependency graphs for all its bits u_i.

[8]and also increasing functions of the length N.

$W = 3$	$9.41 < h \leq 9.83$
$W = 5$	$5.63 < h \leq 5.97$
$W = 7$	$4.68 < h \leq 4.98$
$W = 9$	$4.21 < h \leq 4.48$

Table 2.2: Bounds on the height of a cycle-free dependency graph with $N = 1024$

Dependency graph height of realistic interleavers

However, any finite interleaver and turbo code exhibit dependency graphs with cycles. As the presence of a cycle in a dependency graph results in decreasing the number of vertices at the level where it appears, it also increases the total height of the dependency graph.

As an example, Figures 2.14 & 2.15 show the average population, *i.e.* the number of vertices per level, (computed for the N dependency graphs, $W = 3$) of two different interleavers of size $N = 65536$. The first one is chosen at random and the second one is the conventional block row-column interleaver. We see from the population distribution that a maximum of $32/2 = 16$ iteration steps[9] and an average of $22/2 = 11$ are needed to converge in the case of a random interleaver. This number is much larger, a maximum of $258/2 = 129$ and $254/2 = 126$ on the average, in the case of a row-column interleaver. This behavior will be explained in the next Section, where we prove that any row-column interleaver has minimal cycles of length 3.

We empirically remark, on several examples of purely random interleavers with different lengths N, that the maximal height of their dependency graphs is twice the height of the corresponding theoretical cycle-free dependency graph.

Our first conclusion is that *a good interleaver must produce compact (short) dependency graphs to enable a fast propagation of informations.* The fact that we do not know the actual clump size W is not important : we just compare the average/maximal heights of the dependency graphs of the competing interleavers for different values of W.

[9]An iteration step corresponds to two decoding steps.

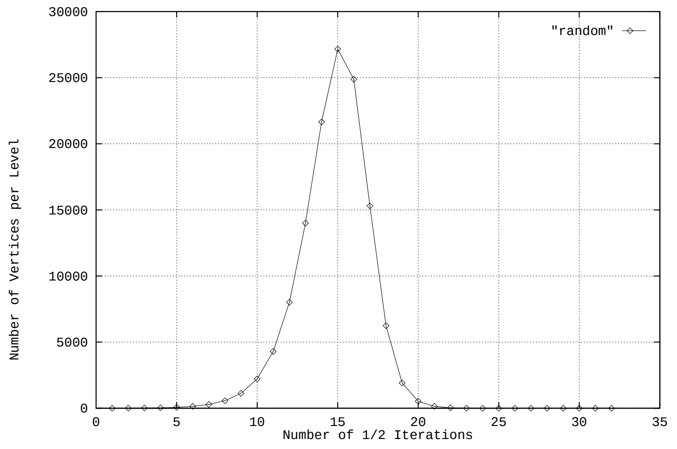


Figure 2.14: Average population of the dependency graphs for $N = 65536$. Random interleaving. $W = 3$.

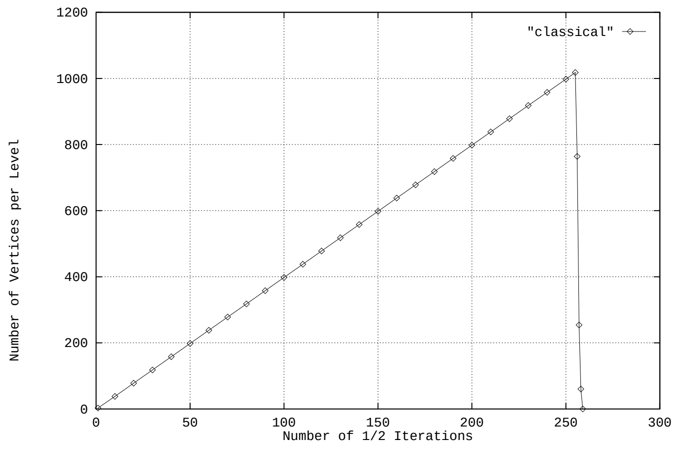


Figure 2.15: Average population of the dependency graphs for $N = 65536$. Row-column interleaving. $W = 3$.

2.3.2 First-cycle level in the dependency graph

We saw that a cycle including the DG summit is generated by a collision between two clumps. These cycles contain strong links and weak links. For short cycles, the number of weak links is not as large as to make the information fade before reaching the summit, and so the information propagating upward in the dependency graph will reach the summit via two different routes and will modify the APP of the summit bit.

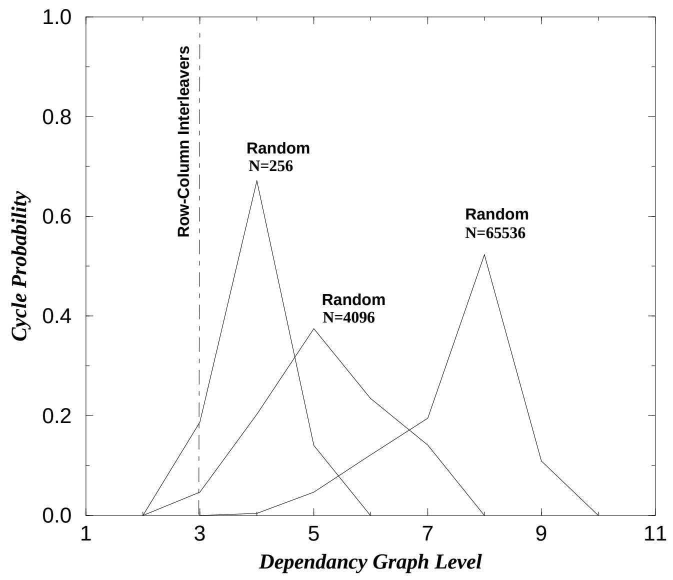


Figure 2.16: Minimal cycles distribution of different interleavers.

Figure 2.16 illustrates the first cycles distribution for different interleaver sizes. It can be easily shown that all minimal cycles of a classical row-column interleaver start from the summit and terminate at level 3 : see Appendix A.

Our second conclusion is that *a good interleaver must produce dependency graphs with cycles as large as possible to guarantee the lowest dependency of incoming probabilities.*

These conclusions involve two contradictory properties : dependency graphs can not be "compact" and have large cycles at the same time. Indeed, the diameter[10] d of any graph is linked to its girth g by the inequality :

$$g \leq 2d + 1 \tag{2.17}$$

Intuitively, the maximal height of all the dependency graphs of a turbo code is linked to the diameter of the Bayesian network of the code (the relation depends on the chosen clump size W), and the minimum first cycle level is liked to the girth (and also to W).

This fact explains why decoding iteration steps remain useful even if convergence (as we defined it) has been reached. All the observations have been conveyed to each bit, but are corrupted by the presence of cycles.

The performance comparison of two turbo codes with same rate $R = 1/3$, interleaver length $N = 4096$, and $J = 2$ constituent codes with generators $(1, 21/37)$, but using two different interleavers (row-column and purely random) validates our conclusions. It is depicted in Figure 2.17.

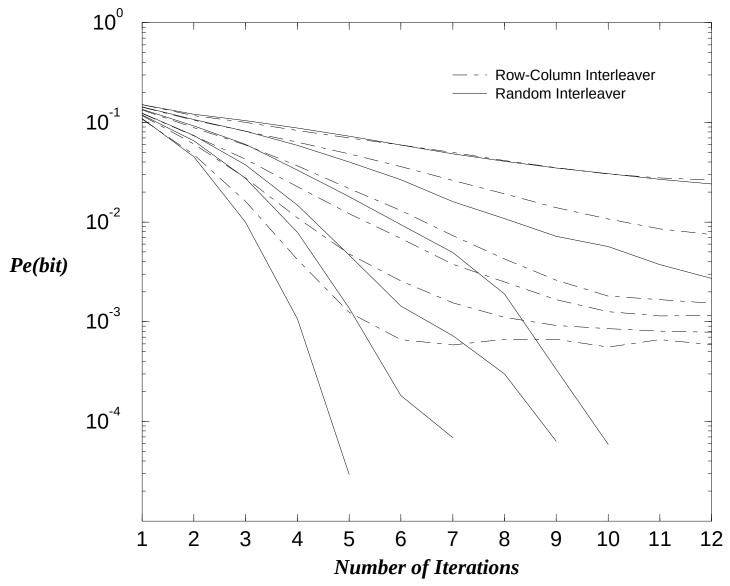


Figure 2.17: Comparison of row-column and random interleaving, $N = 4096$. $Eb/N0$ from 0.0 dB to 0.5 dB with 0.1 dB step.

[10]It is the greatest distance between any two vertices.

Indeed, the dependency graph of the row-column interleaver is much higher than the one of the random interleaver (hence convergence is slower), and furthermore its first cycle distribution (*i.e.* a Dirac measure at level 3) is worst as shown in Figure 2.16 (hence extrinsic informations are more dependent).

2.4 Concluding remarks

In this chapter, we introduced a graphical tool, the dependency graph, that can be used to compare interleavers for turbo codes, regardless of the chosen constituent codes. Since dependency graphs with either large height or short cycles can be identified, the idea of expurgating the bits having such “bad” DGs should be studied further on.

Chapter 3

Low density parity check codes

Gallager introduced in 1962/1963 error-correcting codes based on low-density parity-check matrices. These low-density parity-check (LDPC) codes [46] [47] exploited the following fruitful ideas :

- the use of random permutations linking simple parity-check codes to build an efficient low complexity code that imitates random coding,
- an iterative decoding technique where *a priori* informations and channel observations are both used to compute *a posteriori* and new *a priori* informations.

Unfortunately, Gallager's work has been forgotten by the majority of the scientific community during the past three decades, until the recent invention of turbo codes [22] which share the same above ingredients with LDPC codes.

LDPC codes then gained great attention again, and several scientists work on them. Among these recent works, MacKay [73] showed that Gallager's decoding algorithm is related to Pearl's belief propagation algorithm [78], Luby *et al.* [69] [70], Richardson *et al.* [88] and MacKay *et al.* [72] studied *irregular* LDPC schemes. They exhibit performance extremely close to the capacity limit, and outperform turbo codes for large code lengths.

In this chapter, we briefly recall the construction of the original LDPC codes, their analytical properties as derived by Gallager and their iterative decoding. We present their graphical representation as Tanner codes [97] on random graphs. We show that turbo codes can be represented as block codes with a special low-density parity-check matrix. This chapter does not claim to be self-contained but its aim is rather to be an introduction for the generalized low-density (GLD) codes presented in Chapter 4, whose construction is inspired by LDPC codes, and that share common properties with them.

3.1 LDPC structure and decoding

3.1.1 Definition of low density-parity check-codes

An LDPC code $\mathcal{C}$ with parameters (N, j, k) is a binary linear block code of length N whose parity-check matrix H has j ones in each column, k ones in each row, and thus zeros elsewhere. The numbers j and k have to remain small with respect to N in order to obtain a sparse matrix H. Such a matrix is represented in Figure 3.1.

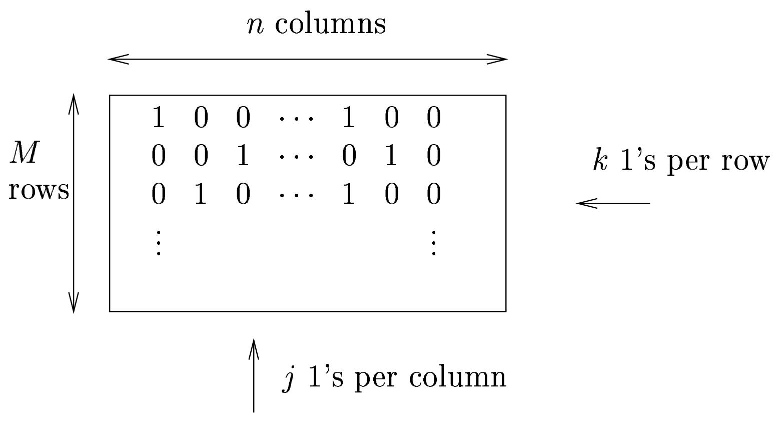


Figure 3.1: Properties of the parity check matrix H of LDPC codes.

This matrix has hence $M = N - K = Nj/k$ rows *i.e.*, the number of single *parity-check equations* (pce) of the code. Each coded bit belongs to j pces, and any pce involves k coded bits. The rate of the code is : $R \geq 1 - j/k$. The inequality holds because we do not constraint the matrix H to have independent rows.

Gallager's construction of a low-density parity-check matrix H with parameters (N, j, k) consists of dividing it into j sub-matrices $H^1, \cdots, H^j$, each containing a single one in each of its columns. The first of these, H^1, looks like a "flattened" identity matrix (that is, an identity matrix where each one is replaced by k ones in a row, and where the number of columns is multiplied accordingly). The $j-1$ other sub-matrices $H^2, \cdots, H^j$ are derived from H^1 by $j-1$ column-wise random permutations $\pi_2, \cdots, \pi_j$ of H^1. Table 3.1 shows the parity-check matrix of a particular $(20, 3, 4)$ LDPC code. It can be noted that summing in each sub-matrices all the rows leads to an all-one row. Hence, there are at least $j-1$ dependent rows and hence the rate is greater than $1 - j/k$.

1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0
0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0
0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	1	0	0	0
0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1

Table 3.1: Gallager's construction of the parity-check matrix of a $(20, 3, 4)$ LDPC code $\mathcal{C}$.

$\mathcal{C}$ can be seen as the intersection of j *super-codes* $\mathcal{C}^1, \cdots, \mathcal{C}^j$ whose respective parity-check matrices are the sub-matrices $H^1, \cdots, H^j$. Since each sub-matrix consists of N/k single parity-check equations of k bits that are mutually exclusive, each super-code is the direct sum of N/k *independent* single-parity-check $(k, k-1)$ codes (spcc) $\mathcal{C}_0$. We have :

$$\mathcal{C} = \bigcap_{j=1}^{J} \mathcal{C}^j \tag{3.1}$$

and :

$$\mathcal{C}^1 = \bigoplus_{l=1}^{N/k} \mathcal{C}_0 \tag{3.2}$$

It is important to note that the matrix H is not in systematic form. Therefore a systematization has to be done, for two reasons. The first one is to compute the actual dimension, and rate, of the code. The second reason is the ease of encoding. Let us denote by $H' = [P|I]$ the result of the systematization of H, and Π_S the column permutation applied. H' is no longer a low density parity check matrix. $G = \left[I|P^T\right]$ is the systematic generator matrix associated with H'. If b is the information bits vector, the corresponding codeword is $c = bG$. We have $cH'^T = 0$, but also $\Pi_S^{-1}(c)H^T = 0$. Since Gallager's probabilistic decoding (see Section 3.1.3) takes benefit of the sparsity of matrix H, an LDPC coding scheme uses the systematic generator matrix G at the encoder and the LDPC matrix H at the decoder. The received symbols must be interleaved by Π_S^{-1}. For a large length, the encoding complexity is high, which is a practical drawback of LDPC codes. There exist ways to get around the encoding problem : they use the graphical representation of LDPC codes (see Section 3.2) to reduce the encoding complexity [70].

3.1.2 Analytical properties of LDPC codes

Gallager presented several analytical results on LDPC codes. We only summarize here those related to the minimum Hamming distance of LDPC codes, since the methods used to find them are also used in our original work.

LDPC codes with $j = 3$ are asymptotically good

Gallager compared the asymptotic *average*[1] minimum Hamming distance properties of the LDPC codes to the ones of the whole ensemble of binary linear block codes of same rate R. By the random coding argument, it is well known that the latter reaches the Varshamov-Gilbert (VG) bound ([11] p. 251) : asymptotically, when the length N of the parity-check codes

[1]"average" means that we consider the ensemble of LDPC codes with same parameters (N, j, k), constructed with all the possible choices for the interleavers $\pi_2, \cdots, \pi_j$, and we average the results on this ensemble.

tends to infinity, the average normalized Hamming distance of their ensemble $\delta_0 = \overline{d_{Hmin}}/N$ satisfies :

$$H(\delta_0) = (1 - R)\log(2) \tag{3.3}$$

where H denotes the natural entropy function.

Computing the average number of codewords $\overline{N_{jk}(\ell)}$ of weight ℓ of the LDPC code ensemble of parameters (N, j, k), Gallager proved that when N is large enough :

$$\overline{N_{jk}(\ell)} \leq C(\lambda, N)\exp\left(-NB_{jk}(\lambda)\right) \tag{3.4}$$

where B_{jk} and $C(\lambda, N)$ are defined as follows :

$$\begin{aligned} B_{jk}(\lambda) &= (j-1)H(\lambda) - \frac{j}{k}\left[\mu(s) + (k-1)\ln 2\right] + js\lambda & (3.5) \\ C(\lambda, N) &= \left[2\pi N\lambda(1-\lambda)\right]^{\frac{j-1}{2}} \exp\left(\frac{j-1}{12N\lambda(1-\lambda)}\right) & (3.6) \end{aligned}$$

where $\lambda = \ell/N$, $\mu(s)$ is a function depending only on k and s is a parameter that has to be optimized. We omit the details since they can be found in [47] and since we use the same approach in the next chapter to prove that GLD codes are also asymptotically good.

Asymptotically, the sign of the exponent function $B_{jk}(\lambda)$ determines the behavior of $\overline{N_{jk}}(l)$: if $B_{jk}(\lambda) > 0$, then $\overline{N_{jk}}(l)$ tends to zero when N tends to infinity. The highest value of λ such that $B_{jk}(\lambda) > 0$ obtains gives us an asymptotic lower bound on the average normalized Hamming distance δ_{jk}. We computed the values of δ_{jk} for different choices of (j, k) and compared them with the VG bound. The results are presented in Figure 3.2.

We observe that $\delta_{jk} > 0$ for all the computed values with $j \geq 3$. Hence, the LDPC codes are *asymptotically good*[2] if $j \geq 3$. Furthermore, LDPC codes are close to the VG bound when j increases.

LDPC codes with $j = 2$ are not asymptotically good

Gallager used a graphical argument to prove that LDPC with $j = 2$ are not asymptotically good. We reproduce his argument here, since he used a concept similar to the dependency graph.

[2] *i.e.* their minimum Hamming distance increases linearly with the length of the code.

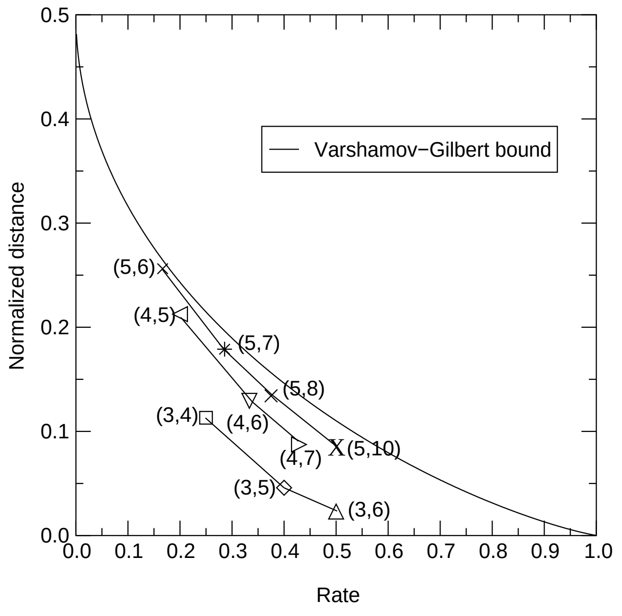


Figure 3.2: Average asymptotic values of the normalized Hamming distance of the ensemble of LDPC codes.

Let us consider a LDPC code with parameters $(N, 2, k)$. Let us associate a vertex with each coded bit, and consider the N/k single-parity-check equations as “super” edges linking the k vertices representing the coded bits that they involve. Since $j = 2$, any vertex belongs to two “super” edges.

Let us choose any vertex and build the equivalent of the dependency graph which stems from it : we follow its two edges, and put its $2(k-1)$ neighbors in the first level. Let us iterate this process with the bits of the first level as shown in Figure 3.3 for $k = 3$. We plot the two different “super” edges of any vertex with two different types of lines in order to distinguish them easily.

Let us assume that the shortest cycle passing through the summit arises at level c. The i-th level $(i < c)$ contains $2(k-1)^i$ vertices. We can roughly bound the number of vertices at level c-1 as :

$$2(k-1)^{c-1} \leq N \tag{3.7}$$

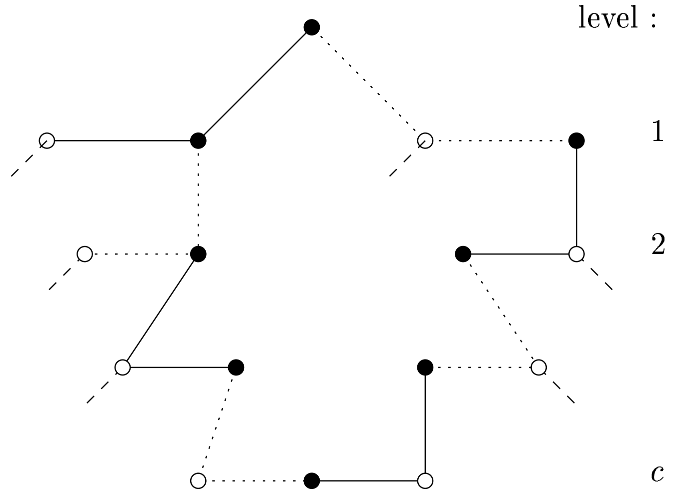


Figure 3.3: Dependency graph of an LDPC code with $j = 2$ and $k = 3$.

which leads to :

$$c \leq 1 + \log_{k-1}\left(\frac{N}{2}\right) \tag{3.8}$$

For the shortest cycle, we consider the set of vertices that are at the intersection of the "super" edges in the cycle. They are represented in black in Figure 3.3. The word with all coded bits set to zero, except the ones corresponding to the black vertices that we set to one, is clearly a codeword of $\mathcal{C}$. Its Hamming weight d satisfies :

$$d = 2c \tag{3.9}$$

since there are exactly two black vertices on levels 1 to $c-1$ plus the summit and the last vertex. Hence, we have :

$$d \leq 2 + 2\log_{k-1}\left(\frac{N}{2}\right) \tag{3.10}$$

We just found a nonzero codeword that has a Hamming weight that increase only logarithmically with N. Hence, LDPC codes with $j = 2$ can not be asymptotically good.

3.1.3 Gallager's probabilistic decoding algorithm

In this section, we slightly modify the notations of the LDPC codes : j is replaced by J and k by K.

APP decoding of a single parity-check equation

Let b_l, $1 \leq l \leq K$, be the coded bits belonging to a particular pce. Let p_l, $1 \leq l \leq K$, be the probability that the bit b_l equals 1. We assume these bits are **independent**. Let S be the event that the pce is verified and $\overline{S}$ the complementary event. The probability of S (respectively $\overline{S}$) corresponds to the probability to have an *even* (respectively *odd*) number of bits b_l equal to 1. It can be written [47] :

$$\mathrm{P}(S) = \frac{1 + \prod_{l=1}^{K} (1 - 2p_l)}{2} \quad \text{and} \quad \mathrm{P}(\overline{S}) = \frac{1 - \prod_{l=1}^{K} (1 - 2p_l)}{2} \tag{3.11}$$

Assuming that the coded bits are known *via* their observed values y_l, $1 \leq l \leq K$, at the output of a memoryless channel, the probability that the pce is verified given the observations is :

$$\mathrm{P}(S|\, \{y\}_{l=1}^{K}) = \frac{1 + \prod_{l=1}^{K} \left[1 - 2\mathrm{P}(b_l = 1|y_l)\right]}{2} \tag{3.12}$$

Fixing the value of the coded bit b_l, and applying the Bayes rule, we have :

$$\begin{aligned} \mathrm{P}\left(b_l = 1|S, \{y\}_{l=1}^{K}\right) &\propto \mathrm{P}(b_l = 1|y_l)\mathrm{P}\left(S|b_l = 1, \{y\}_{l=1}^{K}\right) \qquad (3.13) \\ &\propto \mathrm{P}(b_l = 1|y_l)\frac{1 - \prod_{l'=1, l' \neq l}^{K} \left[1 - 2\mathrm{P}(b_{l'} = 1|y_{l'})\right]}{2} \end{aligned}$$

Similarly, we have :

$$\mathrm{P}\left(b_l = 0|S, \{y\}_{l=1}^{K}\right) \propto \mathrm{P}(b_l = 0|y_l)\frac{1 + \prod_{l'=1, l' \neq l}^{K} \left[1 - 2\mathrm{P}(b_{l'} = 1|y_{l'})\right]}{2} \tag{3.14}$$

Iterative decoding of LDPC codes

Considering a complete (N, J, K) LDPC code, each bit belongs now to J pces. For each pce m, let us denote by $\mathcal{L}(m)$ its set of K coded bits, and by $S(m)$ the event that it is verified. For each coded bit b_l, let us denote by $\mathcal{M}(l)$ the set of J pces that involve it. We define :

$$\begin{aligned} \mathrm{APP}_l^x &= \mathrm{P}\left(b_l = x|S(m), m \in \mathcal{M}(l), \{y\}\right) \qquad (3.15) \\ Q_{m,l}^x &= \mathrm{P}\left(b_l = x|S(m'), m' \in \mathcal{M}(l), m' \neq m\right) \qquad (3.16) \\ V_{m,l}^x &= \mathrm{P}\left(S(m)|b_l = x, Q_{m,l'}, l' \in \mathcal{L}(m), l' \neq l\right) \qquad (3.17) \end{aligned}$$

The APP is defined in the usual way. $Q^x_{m,l}$ represents the partial *a posteriori* probability of the bit b_l, since it does not take the m-th pce into account. Hence, it does not depend on the pce m, and can be used as an enhanced observation probability inside $V^x_{m,l}$, the probability that this pce m is verified.

Using (3.11), we have :

$$V^0_{m,l} = \frac{1 + \prod_{l' \in \mathcal{L}(m), l' \neq l} \left[1 - 2Q^1_{m,l'}\right]}{2} \tag{3.18}$$

and similarly :

$$V^1_{m,l} = \frac{1 - \prod_{l' \in \mathcal{L}(m), l' \neq l} \left[1 - 2Q^1_{m,l'}\right]}{2} \tag{3.19}$$

At the first iteration, $Q^x_{m,l}$ is initialized to the likelihood of b_l.

The $V^x_{m,l}$ can now be used to update the $Q^x_{m,l}$. Indeed, since for a fixed b_l, the set of bits involved in all the pces $m \in \mathcal{M}(l)$ are statistically independent, we can rewrite (3.16) using the Bayes rule as we did in (3.13), but taking now into account $J-1$ pces :

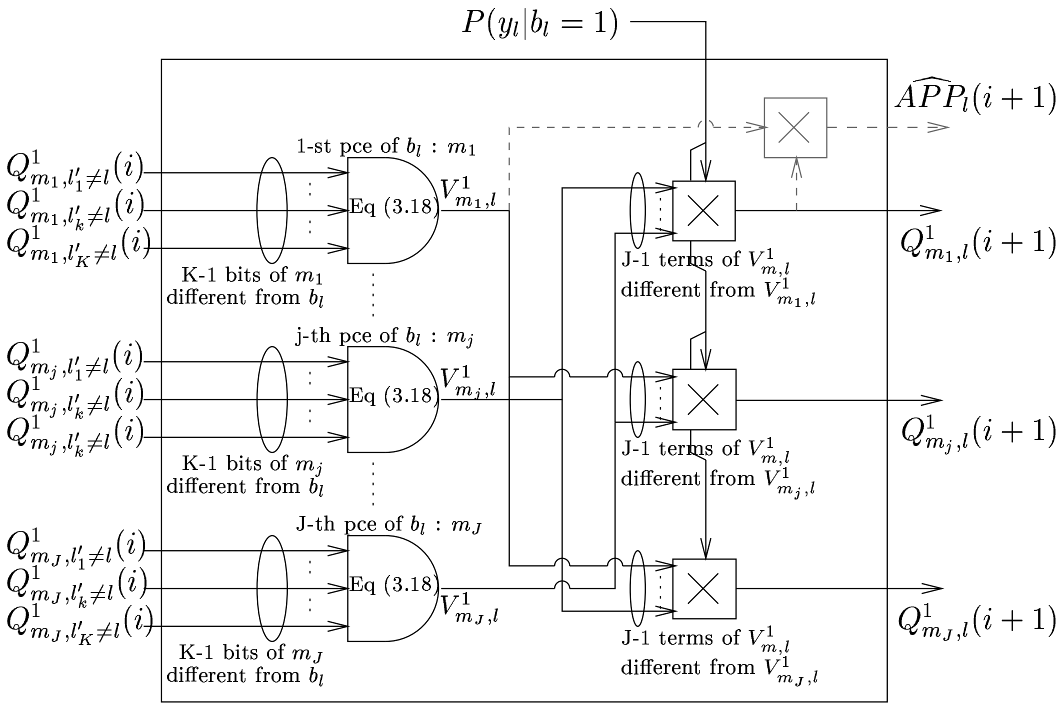
$$\begin{aligned} Q^x_{m,l} &\propto \mathrm{P}(b_l = x | y_l) \prod_{m' \in \mathcal{M}(l), m' \neq m} V^x_{m',l} \\ &\propto \mathrm{P}(y_l | b_l = x) \pi(b_l = x) \prod_{m' \in \mathcal{M}(l), m' \neq m} V^x_{m',l} \\ &\propto \mathrm{P}(y_l | b_l = x) \prod_{m' \in \mathcal{M}(l), m' \neq m} V^x_{m',l} \end{aligned} \tag{3.20}$$

The *a priori* term $\pi(b_l = x)$ was deleted, since there is no *a priori* information on the bit b_l that comes from somewhere else than its parity check equations. An estimate of the APP of all the bits can be derived at this step as :

$$\widehat{APP}(b_l = x) \propto \mathrm{P}(y_l | b_l = x) \prod_{m \in \mathcal{M}(l), m} V^x_{m,l} \tag{3.21}$$

The process is iterated by introducing the new values (3.20) into (3.18) and (3.19).

Figure 3.4 represents the SISO decoding for the bit b_l, at the i-th iteration step. The V values are computed in a middle stage and the partial APPs Q are updated. A complete decoding iteration step consists of applying this SISO decoding to all the coded bits b_l, $1 \leq l \leq N$.



$P(y_l|b_l = 1)$
$\widehat{APP}_l(i+1)$
$Q^1_{m_1,l'_1\neq l}(i)$
$Q^1_{m_1,l'_k\neq l}(i)$
$Q^1_{m_1,l'_K\neq l}(i)$
1-st pce of b_l : m_1
Eq (3.18)
$V^1_{m_1,l}$
K-1 bits of m_1 different from b_l
J-1 terms of $V^1_{m,l}$ different from $V^1_{m_1,l}$
$Q^1_{m_1,l}(i+1)$
$Q^1_{m_j,l'_1\neq l}(i)$
$Q^1_{m_j,l'_k\neq l}(i)$
$Q^1_{m_j,l'_k\neq l}(i)$
j-th pce of b_l : m_j
Eq (3.18)
$V^1_{m_j,l}$
K-1 bits of m_j different from b_l
J-1 terms of $V^1_{m,l}$ different from $V^1_{m_j,l}$
$Q^1_{m_j,l}(i+1)$
$Q^1_{m_J,l'_1\neq l}(i)$
$Q^1_{m_J,l'_k\neq l}(i)$
$Q^1_{m_J,l'_K\neq l}(i)$
J-th pce of b_l : m_J
Eq (3.18)
$V^1_{m_J,l}$
K-1 bits of m_J different from b_l
J-1 terms of $V^1_{m,l}$ different from $V^1_{m_J,l}$
$Q^1_{m_J,l}(i+1)$

Figure 3.4: Diagram of the LDPC SISO decoder of bit b_l at the i-th iteration step

Two scheduling strategies are possible. At the i-th iteration step, J new partial APP values $Q^1_{m_j,l}(i+1)$ are computed by the SISO decoding of bit b_l. At the same iteration i, the SISO decoding of any bit $b_{l'}$, $l' > l$, belonging to m, one of the pces of b_l, can use as input either $Q^1_{m_j,l}(i)$ or $Q^1_{m_j,l}(i+1)$, previously computed, since $l' > l$. The first strategy is called *parallel* scheduling, since the SISO decodings of all the bits are independent, and the second *serial* scheduling. Serial scheduling should improve the iterative convergence speed, since at a fixed iteration step i, the last bits benefit from the partial APP values of the first bits that have been improved. However, we do not observe any significant gain compared with parallel scheduling.

Two LDPC codes have been simulated, using BPSK modulation over the AWGN channel. The first one has as parameters : $(510, 3, 6)$. Its actual dimension is $K = 257$ hence its rate is $R = 0.503$. We used the parallel scheduling version of the iterative SISO decoding with a maximum of 40 iteration steps. The second one has parameters $(48000, 3, 6)$. Its theoretical rate is also 1/2. We limited the same decoding algorithm to 30 iterations. The results are plotted in Figures 3.5 and 3.6.

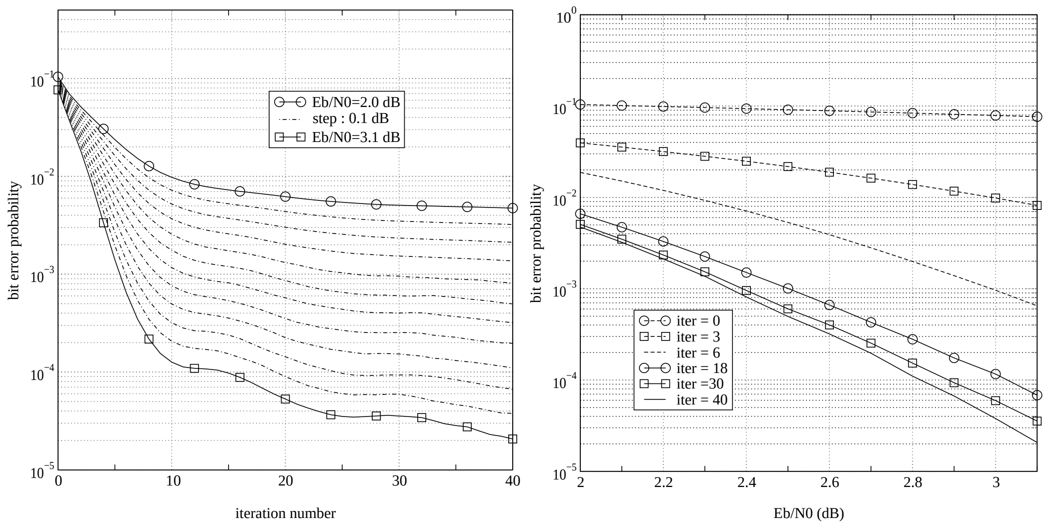


Figure 3.5: Simulation results of an LDPC code with parameters : $(510, 3, 6)$.

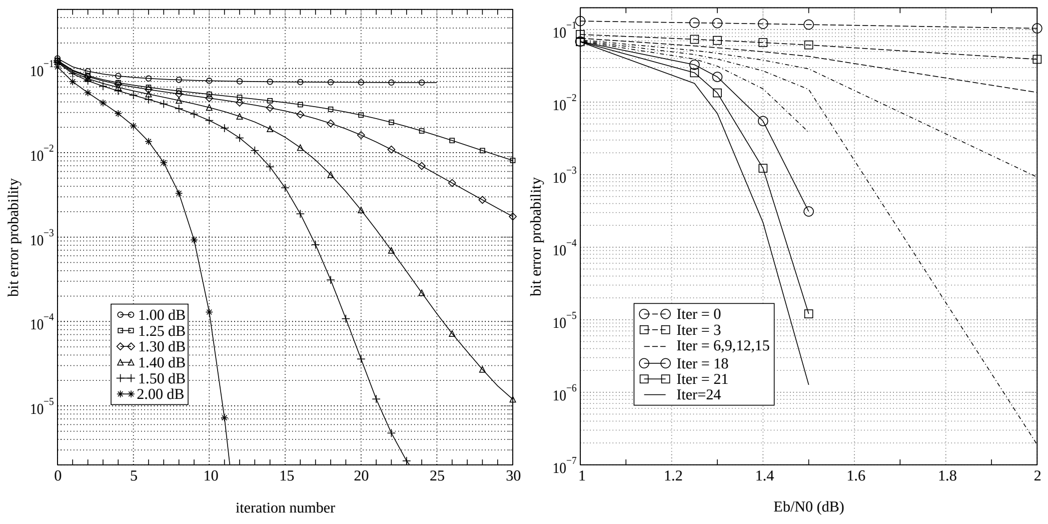


Figure 3.6: Simulation results of an LDPC code with parameters : $(48000, 3, 6)$.

3.2 Graphical representation

Any LDPC code $\mathcal{C}$ with parameters (N, j, k) can be represented as a regular bipartite graph : the Tanner graph. Its properties are the following : its left part has N vertices, representing the coded bits. Its right part has $M = Nj/k$ vertices, representing the single-parity-check codes. There is an edge connecting a bit vertex to a spcc vertex if this bit belongs to the spcc. Hence, the degree[3] of the left part is j, and the degree of the right part is k. Figure 3.7 shows this representation.

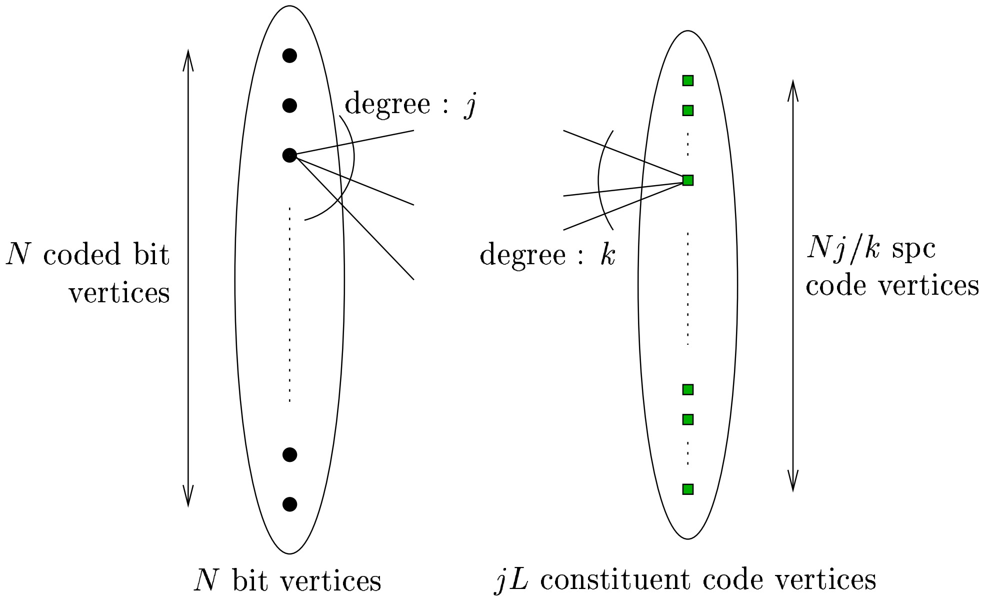


Figure 3.7: Graphical representation of an LDPC code with parameters (N, j, k) as a Tanner random code.

If the parity-check matrix is chosen at random, just satisfying the weight conditions on the rows and columns, the resulting graph is also purely random : the edges are chosen at random, just satisfying the degree conditions on the bit and spcc vertices. Gallager's construction (see Section 3.1.1 and

[3]The degree of a vertex is the number of edges connecting this vertex to others. All the vertices of a regular graph have the same degree. We then speak of the degree of the graph. A regular bipartite graph has two degrees : one for its left part, and one for its right part. For further definitions, and for an introduction to the graph theory, please refer to *e.g.* [34]

Table 3.1) is slightly more specific. Since each coded bit belongs to one and only one spcc $\mathcal{C}_0$ of any of the j super-codes $\mathcal{C}^1, \cdots, \mathcal{C}^j$, the right part of the graph is divided in j clumps of N/k spcc vertices. Any coded bit vertex has hence a single edge connecting it to any of the clumps. This scheme is detailed in the context of GLD codes in the next chapter (see Figure 4.2).

Tanner derived bounds linking the minimum Hamming distance, the number of vertices and the girth of the graph, for any compound code defined by a graph. Applying these results to LDPC codes is straightforward. We derived such results for GLD codes in Section 5.1.1.

3.2.1 Dependency graph

The notion of dependency graph, already introduced for the turbo codes, is even more natural in the context of LDPC (and GLD) codes. Basically, it consists of choosing a coded bit vertex b_l in the graph of the LDPC code, pulling it upwards and observing the vertices that go up level by level.

The j spcc vertices which b_l depends on go up first, followed by the $j(k-1)$ coded bits vertices that belong to these spccs, and so on. The closer from b_l a vertex is, the more important influence on b_l it has. Figure 3.8 represents the first levels of such a dependency graph drawn for a specific bit b_l belonging to an LDPC code with $j = 3$ and $k = 4$.

Gallager's probabilistic decoding algorithm, presented in the previous section, can be interpreted as performing probability propagation along this dependency graph.

At the first iteration step, the estimated *a posteriori* probability of b_l, $\widehat{APP}_l(1)$ is computed taking into account only the probabilities that the pces of the first spcc vertex level are verified (thanks to (3.21)). These probabilities are computed with the $Q(1)$ values of the $j(k-1)$ coded bits at the first bit level (3.19), initialized to the likelihoods of the bits. Hence, this first estimate APP takes into account only a small part of all the available observations $\{y\}$.

At the end of the first iteration, where all the coded bits are processed once, their partial APPs are updated with $Q(2)$, computed at the first iteration step. Hence, the next decoding of b_l takes now into account the

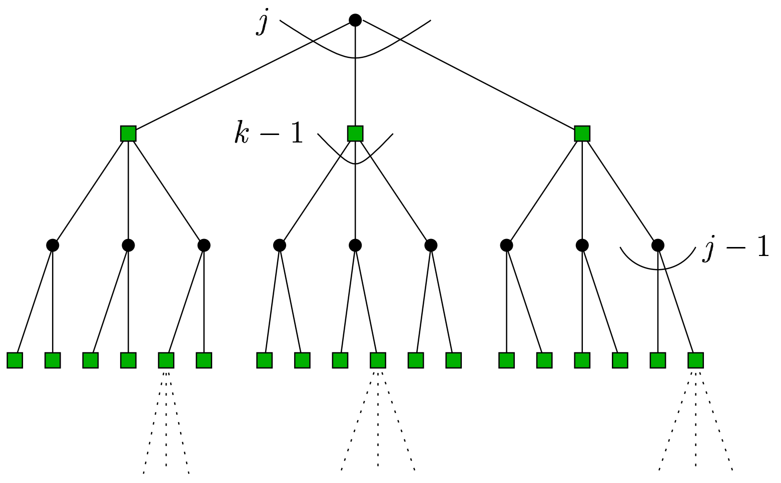


Figure 3.8: Dependency graph of b_l for an LDPC code with parameters $(N, 3, 4)$.

observations coming from the first two bit vertex levels of its dependency graph. This probability propagation is represented in Figure 3.9, with notations that are consistent with Figure 3.4.

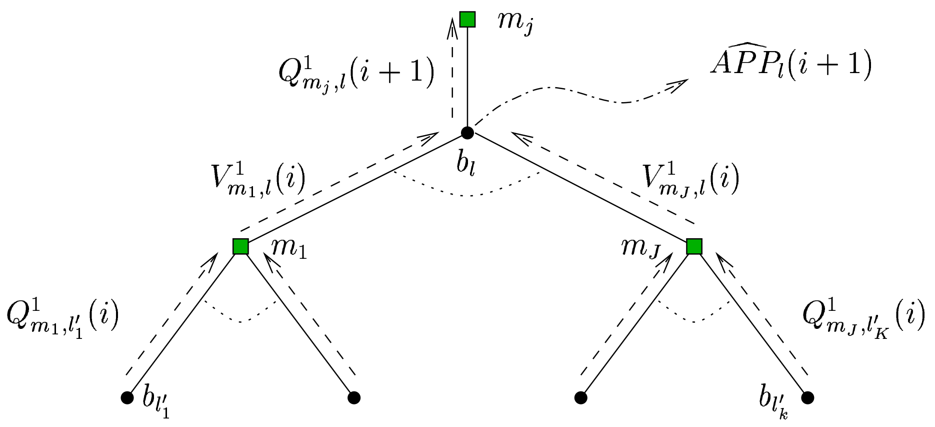


Figure 3.9: Probability propagation in the dependency graph of an LDPC code during the SISO decoding of b_l at iteration step i.

When the iteration steps number is equal to the height of the dependency graph of b_l, the SISO decoding b_l takes into account all the available observations. Hence, the maximum height of all the dependency graphs has the same meaning as for the turbo code DG. It is the number of iteration steps needed for *convergence*.

However, this decoding is not perfect. Indeed, (3.20) holds only if the sets of bits involved in the V probabilities are independent, which is no longer true if one bit (or more) appears twice. Hence, as soon as a cycle is present in the DG, the SISO iterative decoding suffers from dependencies. The first cycle level is therefore the maximum number of iteration steps where decoding is perfect. Unfortunately, it is always lower than the height of the DG. The first cycle distribution in the dependency graphs represents the iterative decoding "quality" of the randomly chosen parity-check matrix H.

But, whereas for turbo codes the DG characteristics highly depend on the clump size W, which is not known, the DGs of LDPC codes do not depend on an arbitrary parameter. The results found are accurate. We drew all the DGs of a $(500, 3, 4)$ LDPC code built using Gallager's construction, and two randomly chosen interleavers. The DGs of 8 bits exhibited cycles of length 4, namely $l = 71, 111, 242, 285, 292, 297, 359, 379$. All the other dependency graphs have no cycles of length 4. We simulated decoding this LDPC code over an AWGN channel (BPSK modulation) and estimated the BER of the individual bits. The results obtained are plotted in Figure 3.10. The black arrows represent the positions of the bits involved in short cycles.

We notice an astonishing agreement between the bits involved in short cycles and their poor protection compared with the other bits. As predicted, their degradation appears at the second iteration step. This result proves that the dependency graph is a valid and useful tool to analyze the iterative SISO decoding process. It can be used to predict the behavior of particularly chosen interleavers.

An algorithmic method to increase the length of the shortest cycles has already been presented by Gallager, working directly on the parity-check matrix H. Chapter 5 investigates the same problem in the more general context of GLD codes, by the use of algebraic interleavers.

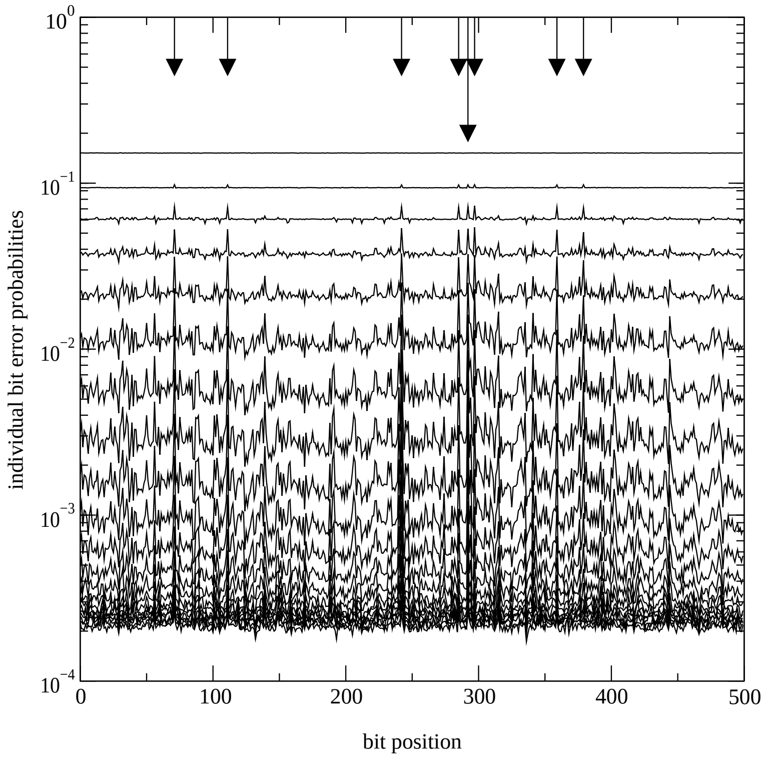


Figure 3.10: Individual bit error probabilities of a $(500, 3, 4)$ LDPC code, at Eb/N_0=3.25 dB. Iterations from 1 to 20.

3.3 Low-density interpretation of turbo codes

To conclude this introduction to LDPC codes, and to emphasis their link with turbo codes, we show on a simple example that turbo codes (parallel concatenation of recursive systematic convolutional codes) have a low-density parity-check matrix H.

Let us consider a non-systematic non-recursive convolutional code $\mathcal{C}_0$ of rate $R = 1/2$. It has two generator polynomials $g_1(X)$ and $g_2(X)$ of degree $\nu + 1$, where ν is the memory of the encoder. Let $u(X)$ be the input of the encoder, and $x_1(X)$ and $x_2(X)$ its outputs. We consider this code as a block code, and write the encoder output as a matrix product :

$$\begin{bmatrix} x_1(X) & x_2(X) \end{bmatrix} = u(X) \begin{bmatrix} g_1(X) & g_2(X) \end{bmatrix} \tag{3.22}$$

Hence we can define an equivalent generator matrix $\mathbf{G}$:

$$\mathbf{G} = \begin{bmatrix} g_1(X) & g_2(X) \end{bmatrix} \tag{3.23}$$

and similarly, an equivalent parity check matrix for $\mathcal{C}_0$:

$$\mathbf{H} = \begin{bmatrix} g_2(X) & g_1(X) \end{bmatrix} \tag{3.24}$$

It is easy to check that :

$$\begin{bmatrix} x_1(X) & x_2(X) \end{bmatrix} \mathbf{H}^T = 0$$

for any output of $\mathcal{C}_0$.

The matrix $\mathbf{H}$ is also the parity-check matrix of $\mathcal{C}'_0$, the recursive systematic convolutional code sharing the same generator polynomials with $\mathcal{C}_0$. Indeed, the outputs $x'_1(X)$ and $x'_2(X)$ of $\mathcal{C}'_0$ are :

$$\begin{bmatrix} x'_1(X) & x'_2(X) \end{bmatrix} = u(X) \begin{bmatrix} 1 & g_2(X)/g_1(X) \end{bmatrix} \tag{3.25}$$

and we can define $\mathbf{G}'$ as :

$$\mathbf{G}' = \begin{bmatrix} 1 & g_2(X)/g_1(X) \end{bmatrix} \tag{3.26}$$

The natural form of $\mathbf{H}'$, the parity check matrix of $\mathcal{C}'_0$ is therefore :

$$\mathbf{H}' = \begin{bmatrix} g_2(X)/g_1(X) & 1 \end{bmatrix} \tag{3.27}$$

But we have :

$$\begin{aligned} & \left[\begin{array}{cc} x_1'(X) & x_2'(X) \end{array}\right]\mathbf{H}'^T = 0 \\ \Leftrightarrow\ & \left[\begin{array}{cc} x_1'(X) & x_2'(X) \end{array}\right]\mathbf{H}'^T g_1(X) = 0 \\ \Leftrightarrow\ & \left[\begin{array}{cc} x_1'(X) & x_2'(X) \end{array}\right]\mathbf{H}^T = 0 \end{aligned} \tag{3.28}$$

More generally, we can show that any convolutional code of rate k/n can be described by a linear block code of length n and dimension k over GF(2)[X].

Let us now consider the special case of a recursive systematic convolutional code $\mathcal{C}_0$ of rate $R = 1/2$ and whose input $u(X)$ has a finite degree $N-1$ (*i.e.* the input vector has size N). Its parity check matrix $\mathbf{H}$ can now be written as an $N \times 2N$ matrix over GF(2) whose coefficients are fixed by its generator. The first (respectively second) $N \times N$ part of $\mathbf{H}$ consists of shifted rows representing the coefficients of $g_2(X)$ (respectively $g_1(X)$). For example, choosing $g_1 = (7)$, $g_2 = (5)$ and $N = 8$, we have :

$$\mathbf{H} = \left[\begin{array}{cccccccc|cccccccc} 1&0&1&0&0&0&0&0&1&1&1&0&0&0&0&0 \\ 0&1&0&1&0&0&0&0&0&1&1&1&0&0&0&0 \\ 0&0&1&0&1&0&0&0&0&0&1&1&1&0&0&0 \\ 0&0&0&1&0&1&0&0&0&0&0&1&1&1&0&0 \\ 0&0&0&0&1&0&1&0&0&0&0&0&1&1&1&0 \\ 0&0&0&0&0&1&0&1&0&0&0&0&0&1&1&1 \\ 0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&1 \\ 0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1 \end{array}\right] \tag{3.29}$$

Please note that for simplification purpose, we do not terminate the trellis.

We consider a conventional turbo code $\mathcal{C}$, resulting from the parallel concatenation of two identical recursive systematic convolutional codes $\mathcal{C}_0$ and $\mathcal{C}_0'$ whose common inputs are separated by an interleaver of length N and matrix[4] Π.

Let u be the information bit vector (of length N), and u' its interleaved version :

$$u' = u\Pi \tag{3.30}$$

Let v be the vector of the parity bits of $\mathcal{C}_0$, and v' the vector of the parity bits of $\mathcal{C}_0'$. We denote by $\mathbf{H_1}$ (resp. $\mathbf{H_2}$) the $N \times N$ matrix consisting of

[4]Π is a matrix of size $N \times N$ with exactly 1 one per row and column

shifted rows representing the coefficients of $g_1(X)$ (resp. $g_2(X)$). Applying (3.28), we have :

$$\begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} \mathbf{H_2} & \mathbf{H_1} \end{bmatrix}^T = 0 \tag{3.31}$$

for $\mathcal{C}_0$ and :

$$\begin{aligned} & \begin{bmatrix} u' & v' \end{bmatrix} \begin{bmatrix} \mathbf{H_2} & \mathbf{H_1} \end{bmatrix}^T = 0 \\ \Leftrightarrow \quad & \begin{bmatrix} u\Pi & v' \end{bmatrix} \begin{bmatrix} \mathbf{H_2} & \mathbf{H_1} \end{bmatrix}^T = 0 \\ \Leftrightarrow \quad & u\Pi\mathbf{H_2}^T + v'\mathbf{H_1}^T = 0 \\ \Leftrightarrow \quad & \begin{bmatrix} u & v' \end{bmatrix} \begin{bmatrix} \mathbf{H_2}\Pi^T & \mathbf{H_1} \end{bmatrix}^T = 0 \end{aligned} \tag{3.32}$$

for $\mathcal{C}'_0$. Since the codewords of the turbo code $\mathcal{C}$ have the form : $\begin{bmatrix} u & v & v' \end{bmatrix}$, the parity-check equation that they satisfy is :

$$\begin{bmatrix} u & v & v' \end{bmatrix} \begin{bmatrix} \mathbf{H_2} & \mathbf{H_1} & 0 \\ \mathbf{H_2}\Pi^T & 0 & \mathbf{H_1} \end{bmatrix}^T = 0 \tag{3.33}$$

Hence, the parity-check matrix of the whole turbo code is :

$$\mathbf{H} = \begin{bmatrix} \mathbf{H_2} & \mathbf{H_1} & 0 \\ \mathbf{H_2}\Pi^T & 0 & \mathbf{H_1} \end{bmatrix} \tag{3.34}$$

It is represented in Figure 3.11. The shaded areas in the matrix are those where the ones can be found. The number of ones per row and per column in the "diagonal" sub-matrices $\mathbf{H_1}$ and $\mathbf{H_2}$ is upper bounded by $L = \nu + 1$, the constraint length of the constituent codes, which is always very small compared with the length of the interleaver. The interleaver does not change the weights of the sub-matrix $\mathbf{H_2}\Pi^T$. The upper bounds on the weights in the different areas of $\mathbf{H_1}$ are pointed by arrows in the Figure.

Therefore, this turbo code can be seen as an irregular LDPC code, since the weight of ones per row and column is not strictly constant, but always very small compared with the size of the parity-check matrix.

The iterative decodings of LDPC and turbo codes are different : the conventional turbo decoding takes advantage of the trellis structure of each constituent code to decode it in a single step. It does not process each row of the equivalent LDPC matrix separately, as the LDPC decoding scheme does. From the "diagonal" sub-matrices, it is easy to observe that the LDPC decoding of the turbo code matrix will not yield good results : there is a lot of very short cycles (square patterns in the matrix).

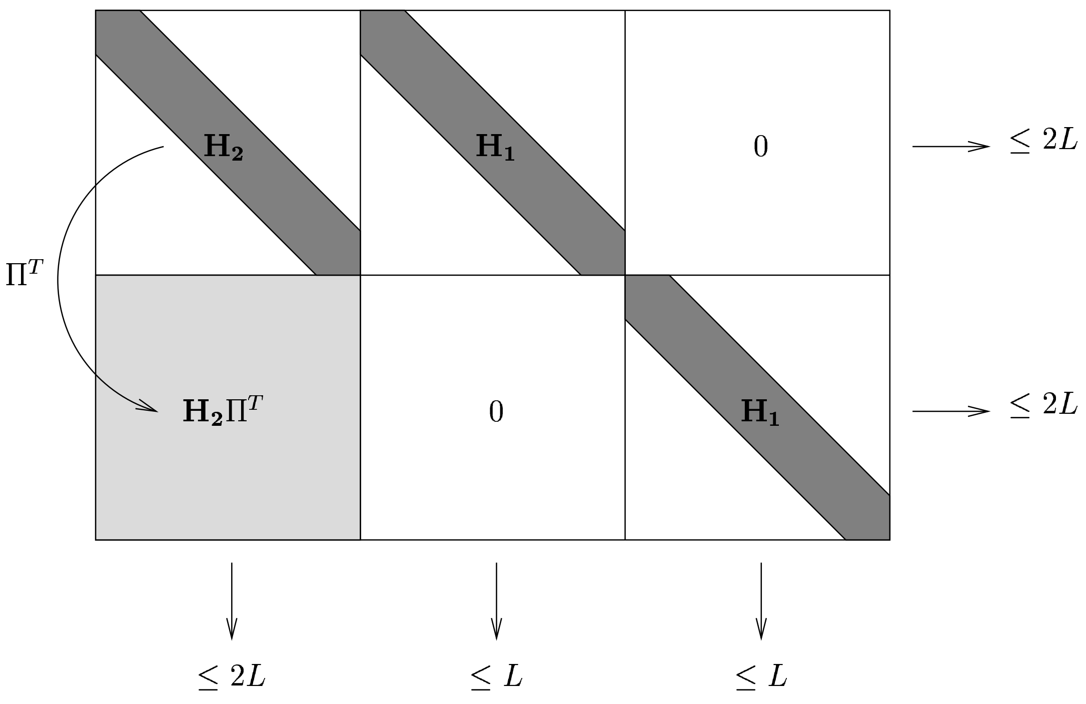


Figure 3.11: Parity check matrix of a turbo code of rate $R = 1/3$ composed of two constituent codes of rate $r = 1/2$.

Finding convolutional codes that lead to "diagonal" sub-matrices without short cycles leads to *self-orthogonal* convolutional codes. Their memory increases rapidly, and classical SISO decodings (FB or SOVA) become intractable. However, we can use their LDPC matrix representation, and Gallager's probabilistic decoding. This has been done in [30], and also in [39] using time-varying convolutional codes. Hence, a *single* well-chosen convolutional code (*i.e.* without the use of any interleaver) with high memory can be iteratively decoded.

Chapter 4

The Generalized Low Density codes*

If I had more time I could write a shorter letter.
Blaise Pascal

In this Chapter, we present a class of codes called *generalized low density* (GLD) codes. This family of pseudo-random error correcting codes is built as the intersection of randomly permuted binary BCH codes. It is a direct generalization of LDPC codes (see Chapter 3). GLD codes can also be seen as Tanner codes based on random bipartite graphs. Independently, Lentmaier and Zigangirov [63] introduced the same construction.

We study the ensemble performance of these codes and prove that they are asymptotically good. Upper and lower bounds on their minimum Hamming distance are provided, together with their maximum likelihood decoding error probability.

An iterative soft-input soft-output decoding for any GLD code is presented, and a method to estimate the performance of *perfect*[1] iterative decoding of GLD codes is derived.

Simulations of GLD codes with $J = 2$ levels were performed over the

*Parts of this chapter have been presented at the ICC'99 conference (Vancouver) and the VTC'99 conference (Houston).

[1]By perfect, we mean iterative decoding on a code whose graph has no cycle. This assumption also corresponds to an infinite length of the code.

AWGN channel with binary antipodal modulation (BPSK) using the iterative decoding scheme presented. We compare the results in terms of performance and complexity with those of product codes, and sketch the main lines of a case study involving transmission of short frames.

4.1 Structure of GLD codes

We will describe the structure of GLD codes in two different ways. The first one uses their parity check matrix so they appear as an extension of the LDPC codes and the second one their graphical representation as an extension of Tanner codes.

4.1.1 Parity-check matrix description

The generalization of LDPC codes is straightforward. We based it on Gallager's construction of LDPC codes. Let us describe the structure of an (N, K) GLD code $\mathcal{C}$:

- Each single-parity-check code $(k, k-1)$ of an LDPC code is replaced by a linear block code $\mathcal{C}_0(n, k, d)$ called the *constituent code.*
- As an LDPC code was the intersection of j super-codes, a GLD code is the intersection of J super-codes[2] $\mathcal{C}^j$:

$$\mathcal{C} = \bigcap_{j=1}^{J} \mathcal{C}^j. \tag{4.1}$$

- As the first super-code of an LDPC code was the direct sum of N/k independent SPC codes, the first super-code $\mathcal{C}^1$ of a GLD code is the direct sum of $L = N/n$ independent constituent codes $\mathcal{C}_0(n, k, d)$:

$$\mathcal{C}^1 = \bigoplus_{l=1}^{L} \mathcal{C}_0 \tag{4.2}$$

[2]we change the number of super-codes from j to J because it no longer corresponds to the column weight.

- L is called the *repetition factor* and J the *number of levels* of the code.
- The super-codes $\mathcal{C}^j$, $j \in \{2 \cdots J\}$ are merely bitwise permutations of the first one :

$$\mathcal{C}^j = \pi_j\left(\mathcal{C}^1\right) \tag{4.3}$$

- The complete number of constituent codes involved in a GLD code is thus $JN/n = JL$.

In terms of the parity-check matrix, each row of the matrix depicted in Table 3.1 is replaced by $n-k$ rows including a single copy of H_0, the parity check matrix of $\mathcal{C}_0$, the remainder being filled with zeros. The full parity-check matrix H of the GLD code is the concatenation of the sub-matrices H^j $j \in \{1 \cdots J\}$ of the super-codes. All the sub-matrices are obtained by pseudo-random column-wise permutations $\pi_j(H^1)$. We have $H^2 = \pi_2(H^1) \cdots H^J = \pi_J(H^1)$. Figure 4.1 shows the parity-check matrix of a GLD code.

Even if the property of constant weight per row and column is no longer satisfied for GLD codes, the property of low density still holds. If we denote by $\bar{k}$ (resp. $\bar{j}$) the average number of ones per row (resp. per column) in H_0, the average number of ones per row for the GLD code is $\bar{k}$ and the average number of ones per column is $J\bar{j}$. These quantities are constant once $\mathcal{C}_0$ and J are chosen, and do not depend on N, the length of the GLD code.

Note that it is not possible to define in the general case the GLD codes as a serial (neither parallel, nor hybrid) concatenation of two or multiple constituent codes. Section 4.1.5 exhibits a special case of GLD codes : the product codes.

4.1.2 Graphical description

The GLD code $\mathcal{C}$ can also be described using a random regular bipartite graph with the following properties. In this sense, it is also a generalization of Tanner codes [97] to a random matching, as depicted in Figure 4.2.

1. The left part contains the N bit vertices (*i.e.,* the coded bits or the codeword). The degree of all the bit vertices is J (*i.e.,* each bit is involved in J constituent codes).

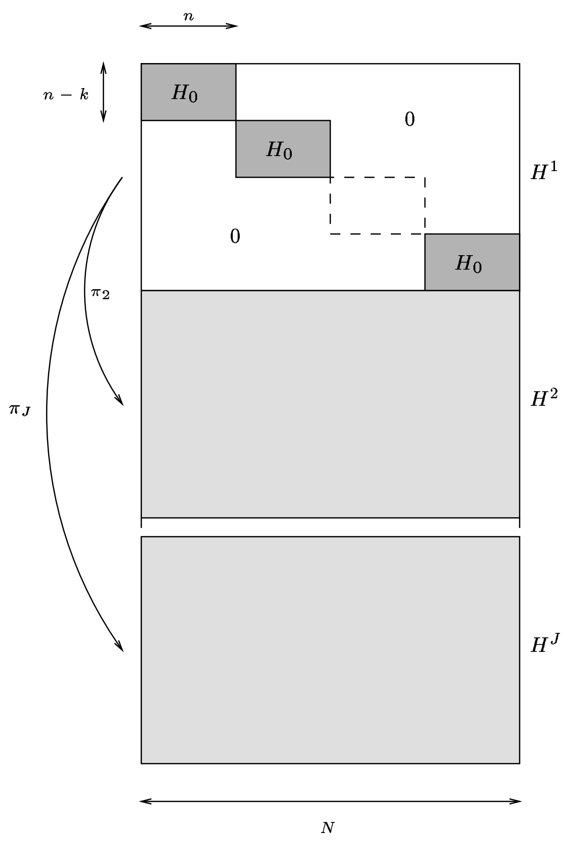


Figure 4.1: Parity-check matrix H of a GLD code

2. The right part contains the JL constituent code vertices. The degree of all the constituent code vertices is n (its length).

3. An edge between a bit vertex and a code vertex means that this bit belongs to that constituent code.

4. The constituent code vertices are grouped into J clumps of L vertices (a clump corresponds to a super-code). The J edges stemming from every single bit vertex are connected to code vertices belonging to different clumps (it corresponds to the condition that the constituent codes of a super-code are independent).

5. The set of edges characterizes the interleaver.

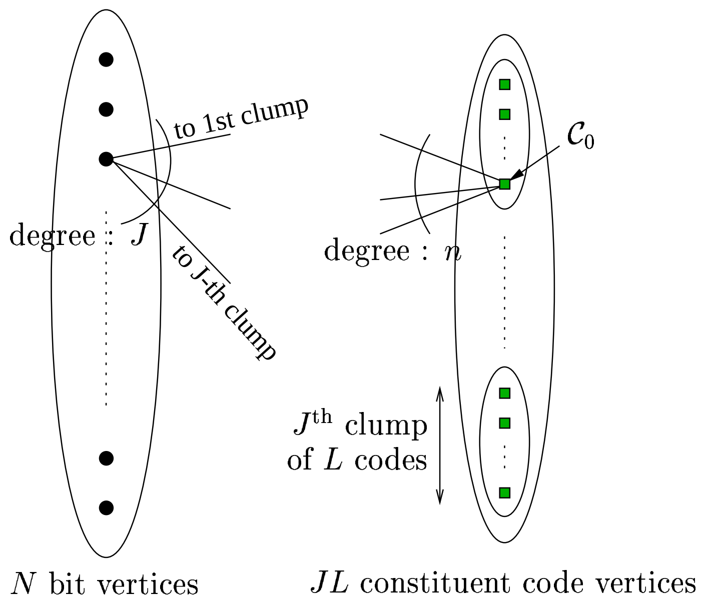


Figure 4.2: The GLD code as a Tanner random code.

This description is equivalent to the one presented in Section 4.1.1, but a little bit more general. Indeed, the bits should not belong to the first super-code (*i.e.,* the first clump of codes in the graph) in the strict order induced by the direct sum.

Let us notice that nothing in the properties stated above forbids that two (or more) bit vertices are connected to the same J constituent codes. However, we try to avoid this situation that creates cycles of length 4 in the graph[3] since Chapter 5 proves that short cycles have a negative influence on iterative decoding.

4.1.3 Coding Rate

The rate of a GLD code can be lower-bounded by observing its parity-check matrix as follows. Let us first assume that the rows of H are independent. The number of rows in each sub-matrix corresponds to $(n-k)L$. Thus, the total number of rows $N-K'$ in H satisfies :

$$N - K' = J(n-k)L = \frac{J(n-k)N}{n} = J(1-r_0)N \tag{4.4}$$

where r_0 denotes the rate of $\mathcal{C}_0$. Since the number of columns in H is N, we have :

$$\frac{K'}{N} = 1 - J(1-r_0). \tag{4.5}$$

The rate R of the GLD code should be K'/N. But nothing guarantees that the rows of H are independent. Hence, the actual dimension K of the code can be greater than K'. Consequently, we found a lower bound on the rate of a GLD code :

$$R = \frac{K}{N} \geq 1 - J(1-r_0). \tag{4.6}$$

In the following, even if $K > K'$, we will neither discard the redundant rows in H nor modify some constituent codes, so as to keep the well structured construction of the matrix and the graph. Some constraints might just be used more than once.

4.1.4 A special case : GLD codes with $J = 2$ levels

We will prove in Section 4.2 that GLD codes based on Hamming constituent codes are asymptotically good even if $J = 2$, and that iterative decoding (presented in Section 4.3) is very simple to implement in this case. Moreover, as the rate decreases linearly with J (see (4.6)), the case of 2-level GLD codes is of particular interest.

[3] Cycles of length 4 are the shortest a GLD graph can have

A GLD code is then the intersection of two super-codes, each of them being composed of L independent constituent codes. The constituent codes belonging to the first super-code are called the *upper codes*, and the ones belonging to the second super-code are the *lower codes.* There are exactly $2L$ constituent codes. A simple application of (4.6) shows that :

$$R_{J=2} \geq 2r_0 - 1 \tag{4.7}$$

which indicates that we have to choose in the general case constituent codes with rate greater than 1/2. Table 4.1 shows the rate bound for 2-level GLD codes based on different constituent codes.

Constituent code				GLD Code
type	n	k	r_0	Minimum Rate
Hamming	7	4	0.57	0.14
Hamming	15	11	0.73	0.46
Shortened Hamming	12	8	2/3	1/3
Extended Hamming	16	11	0.69	0.38
Hamming	31	26	0.84	0.68
Shortened Hamming	20	15	3/4	1/2
Ext. and Shortened Hamming	24	18	3/4	1/2

Table 4.1: Minimum rates of 2-level GLD codes and different constituent codes

The effective rate of a GLD code depends not only on the rate of its constituent code, but also on its repetition factor and the interleaver chosen, as shown in the next section.

A 2-level GLD code has also a special graphical representation. It can be described by a bipartite biregular random graph we call the *compact graph* of the 2-level GLD code.

Its left part contains the *upper* code vertices and the right part the *lower* code vertices. Consequently, the left part represents the super-code $\mathcal{C}^1$ and the right part $\mathcal{C}^2$. As each bit is connected to exactly two constituent codes, an edge is drawn between every two vertices when the corresponding constituent codes share a bit. As the super-codes are composed of *independent* constituent codes, there is no edges in-between the vertices of one part. Thus, the graph is bipartite. As all the constituent codes have the same length n, the graph is biregular of degree n. Figure 4.3 represents this graph.

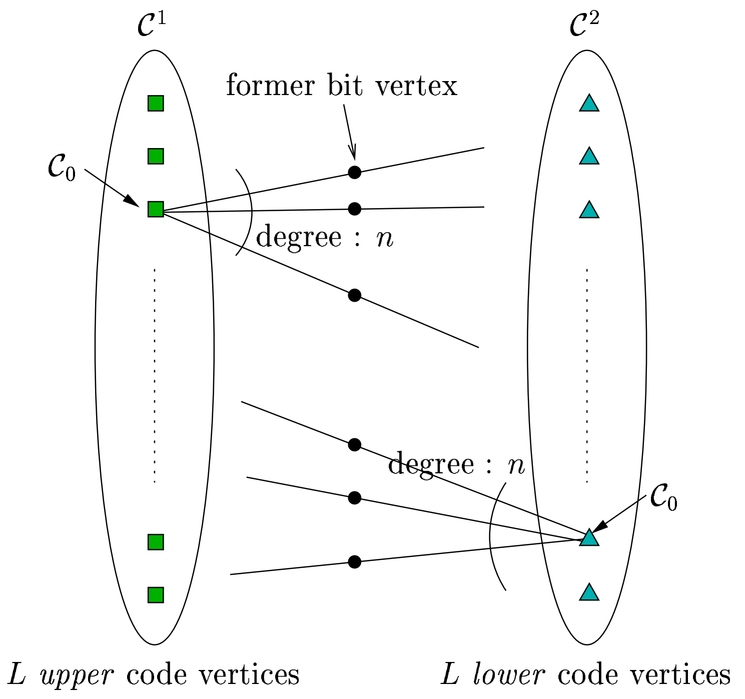


Figure 4.3: Two-level GLD code compact graph representation.

Avoiding that bits belong to exactly the same set of constituent codes in the general case corresponds here to avoid parallel edges in the compact graph.

4.1.5 Another special case : Product Codes

Let us examine 2-level GLD codes with a repetition factor L equal to n. The length of these GLD codes is $N = Ln = n^2$. Each of the two super-codes $\mathcal{C}^1$ and $\mathcal{C}^2$ is thus composed of n constituent codes.

Let us forbid any parallel edge in the compact graph of these codes. As each constituent code vertex is of degree n, and as there is exactly n code vertices in the other part, this implies that the compact graph is connected[4]. It corresponds to the definition of a product code, where each bit belongs to two codes (the *horizontal* and the *vertical* ones).

Example :

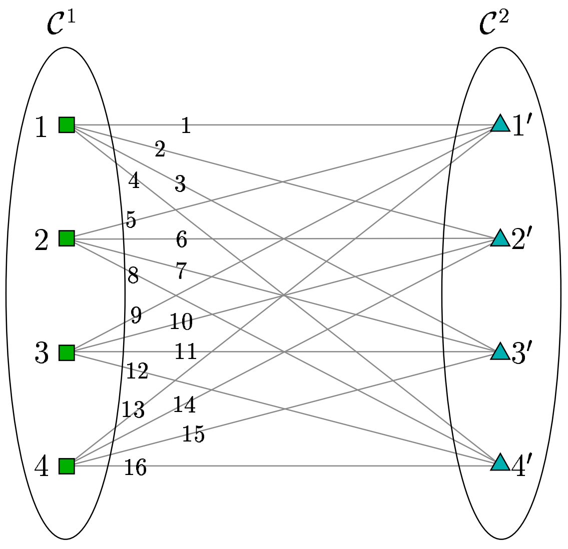


Figure 4.4: Connected compact graph of a block product code

[4]it means that there is an edge connecting each pair of vertices that belong to different parts.

Figure 4.4 shows the compact graph of a GLD code of length $N = 16$, constituent code length $n = 4$ and repetition factor $L = n$. The numbers on the edges label the bits, the *upper* codes are labeled from 1 to 4 and the *lower* codes from $1'$ to $4'$. Let us assume that the constituent codes are $(4, 3, 2)$ single-parity-check codes. This leads to the parity-check matrix description of the code presented in Table 4.2. After systematization (involving one row

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	1	1	1												
2					1	1	1	1								
3									1	1	1	1				
4													1	1	1	1
$1'$	1				1				1				1			
$2'$		1				1				1				1		
$3'$			1				1				1				1	
$4'$				1				1				1				1

Table 4.2: Parity-check matrix of the code

deletion and column reordering), this matrix has the form depicted in Table 4.3.

1	2	3	5	6	7	9	10	11	4	8	12	13	14	15	16
1	1	1							1						
			1	1	1					1					
						1	1	1			1				
1			1			1						1			
	1			1			1						1		
		1			1			1						1	
1	1	1	1	1	1	1	1	1							1

Table 4.3: Systematic form of the parity-check matrix of the code

The dimension of the code is thus $K = 9$, that is the rate of the code is $R = (3/4)^2$, a value higher than the bound given in (4.7). The last form of the parity-check matrix is that of the product code based on single-parity-check codes $(4, 3, 1)$ shown in table 4.4.

Further comparisons between GLD codes and product codes are presented in section 4.5, in terms of performance, complexity and flexibility.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Table 4.4: Product code based on $(4, 3, 1)$ single-parity-check codes

4.2 Ensemble performance

In this section, we first study the average weight distribution of GLD codes over all the possible interleavers, as a function of the number of levels J, the constituent code $\mathcal{C}_0$, and the length N.

From this distribution, we deduce an asymptotical upper bound that allows us to prove that GLD codes are asymptotically good and to compare their minimum distance with the Varshamov-Gilbert bound.

We also derive the maximum transition probability of the BSC channel that an asymptotical GLD code with fixed rate can achieve with maximum likelihood decoding, and compare it to the BSC capacity.

Average upper bounds on the minimum distance of GLD codes of fixed length and constituent code are calculated semi-analytically, together with a tight upper bound on the bit error probability of the ML decoding over the AWGN channel.

4.2.1 Average weight distribution of GLD codes

The direct computation of the exact weight distribution of a GLD code becomes rapidly intractable when N (or equivalently L) increases. However, the weight distributions of the constituent codes used in practical GLD codes (Hamming codes, extended Hamming codes, BCH codes with error correction capacity $t = 2$ and extended BCH codes) are well known. See [79] p. 142 for Hamming and extended Hamming codes, see [74] p. 451 & p. 669 and [21] Ch. 16 for the family of BCH codes. The case of the shortened versions of the former codes is different. The weight distribution might depend on the shortening pattern. But it is always possible and simple to get the distribution of the constituent code, either by formulas or computations. Indeed, the rate of the constituent codes must be high so as to obtain a GLD code of reasonable rate (4.6) and thus the dimension of the dual code remains small. Exhaustive search of all the codewords of the dual code and application of the MacWilliams identity ([79] p. 66) lead to the weight distribution of the constituent code.

The *average* weight coefficients of a GLD code can be easily obtained by averaging over all the possible interleavers π_j.

Let us denote by $g_{\mathcal{C}_0}(s)$ (or $g(s)$ for brevity's sake) the moment generating function of $\mathcal{C}_0$, that is the exponential polynomial whose coefficient g_i of degree i is the normalized number of codewords with weight i. For example, the moment generating function of the $(7,4,3)$ Hamming code is :

$$g_{\mathcal{C}_0}(s) = \frac{1 + 7e^{3s} + 7e^{4s} + e^{7s}}{16} \tag{4.8}$$

As the J super-codes $\mathcal{C}_j$ of length N are the direct sum of $L = N/n$ independent constituent codes $\mathcal{C}_0$, their moment generating functions $G_{\mathcal{C}_j}(s)$ are simply a power of $g(s)$:

$$G_{\mathcal{C}_j}(s) = G(s) = g(s)^{N/n} = \sum_{\ell} Q(\ell)e^{\ell s} \tag{4.9}$$

where $Q(\ell)$ is the probability that a codeword of $\mathcal{C}_j$ has a weight ℓ. We assume without loss of generality that all the super-codes are built from the same constituent code.

Since the total number of codewords in $\mathcal{C}_j$ is $(2^k)^L = (2^k)^{N/n}$, the number of codewords of $\mathcal{C}_j$ having weight ℓ is :

$$N_j(\ell) = N(\ell) = 2^{(kN/n)}Q(\ell) \tag{4.10}$$

Thanks to the fact that $\mathcal{C}_1, \cdots, \mathcal{C}_J$ are randomly permuted versions of the same super-code, and thus independent, the probability $P(\ell)$ that a vector of weight ℓ belongs to $\mathcal{C} = \mathcal{C}_1 \bigcap \cdots \bigcap \mathcal{C}_J$ is the product of the probabilities that it belongs to each code, that is $N(\ell)/\binom{N}{\ell}$:

$$P(\ell) = \left(\frac{N_1(l)}{\binom{N}{\ell}} \right)^J \tag{4.11}$$

Finally, the average number of codewords in $\mathcal{C}$ having weight ℓ is :

$$\overline{N(\ell)} = \binom{N}{\ell} \times P(\ell) = \frac{2^{(JkN/n)} Q(\ell)^J}{\binom{N}{\ell}^{J-1}} \tag{4.12}$$

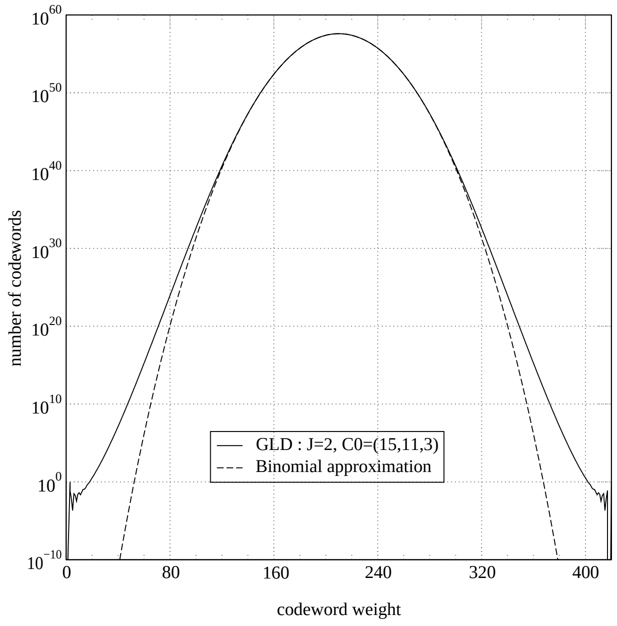


Figure 4.5: Average weight distribution of 2-level GLD codes based on the Hamming $(15, 11)$ as constituent code, compared with the binomial approximation.

This formula is the starting point of different analytical or semi-analytical results presented in the following sections.

4.2.2 Lower bound on the asymptotic minimum distance

We want in a first step to upper-bound $\overline{N(\ell)}$. Using exactly the same bounding technique as in [47], that is upper bounding roughly each coefficient $Q(\ell)$ in (4.9) by the entire function $G(s)$, we obtain :

$$Q(\ell) \leq G(s)e^{-\ell s} \tag{4.13}$$

Defining $\mu(s) = \ln(g(s))$ and the normalized weight $\lambda = \ell/N$, (4.13) can be rewritten as :

$$Q(\ell) \leq \exp\left\{-N\left[\lambda s - \frac{\mu(s)}{n}\right]\right\} \tag{4.14}$$

Let us now lower-bound the denominator in 4.12, namely the binomial coefficient. Extended Stirling bounds[5] on $n!$ are [12] [1] :

$$\sqrt{2\pi z}\; z^z e^{-z} \leq z! \leq \sqrt{2\pi z}\; z^z e^{-z} \exp\left\{\frac{1}{12z}\right\} \tag{4.15}$$

and give for the binomial coefficient :

$$\frac{\exp\left\{NH(\lambda) - \frac{1}{12N\lambda(1-\lambda)}\right\}}{\sqrt{2\pi N\lambda(1-\lambda)}} \leq \binom{N}{\ell} = \binom{N}{\lambda N} \leq \frac{\exp\{NH(\lambda)\}}{\sqrt{2\pi N\lambda(1-\lambda)}} \tag{4.16}$$

where $H(\lambda)$ is the natural entropy function :

$$H(\lambda) = -\lambda\ln(\lambda) - (1-\lambda)\ln(1-\lambda) \tag{4.17}$$

Introducing (4.14) and (4.16) into (4.12) gives :

$$\begin{aligned}\overline{N(\ell)} &\leq \frac{2^{(JkN/n)}\sqrt{2\pi N\lambda(1-\lambda)}^{J-1}\exp\left\{-JN\left[\lambda s - \frac{\mu(s)}{n}\right]\right\}}{\exp\left\{(J-1)\left[NH(\lambda) - \frac{1}{12N\lambda(1-\lambda)}\right]\right\}}\\ &\leq \sqrt{2\pi N\lambda(1-\lambda)}^{J-1}\exp\left\{\frac{J-1}{12N\lambda(1-\lambda)}\right\}\\ &\quad\times\exp\left\{-N\left[(J-1)H(\lambda) - \frac{J}{n}(\mu(s) + k\ln 2) + J\lambda s\right]\right\}\end{aligned}$$

We define :

$$C(\lambda, N) = \sqrt{2\pi N\lambda(1-\lambda)}^{J-1}\exp\left\{\frac{J-1}{12N\lambda(1-\lambda)}\right\} \tag{4.18}$$

and :

$$B(\lambda, s) = (J-1)H(\lambda) - \frac{J}{n}[\mu(s) + k\ln 2] + J\lambda s \tag{4.19}$$

The final upper bound on the average number of codewords of weight ℓ in the GLD codes is :

$$\overline{N(\ell)} \leq C(\lambda, N) \times e^{-NB(\lambda,s)} \tag{4.20}$$

[5]The bounds given in [47], p. 12, equation (2.3) are false.

Asymptotically, when the length N tends to infinity, $\overline{N(\ell)}$ tends to zero if $B(\lambda, s)$ is strictly positive. Our aim is now to find (if it exists) the smallest value δ of λ in $]0 \cdots 1/2[$ satisfying $B(\delta, s) = 0$ and $B(\lambda, s) > 0$ for $\lambda \in]0 \cdots \delta[$. This value of δ would give us an asymptotical lower bound on the normalized minimum distance of the GLD code.

$B(\lambda, s)$ is a function of an arbitrary parameter s we have to optimize in order to get the tightest bound, namely we want $B(\lambda, s)$ as large as possible for each λ. Let us denote by $f_\lambda(s)$ the part of $B(\lambda, s)$ that depends on s :

$$f_\lambda(s) = -\frac{J}{n}\mu(s) + J\lambda s \tag{4.21}$$

Finding the value of s that maximizes $B(\lambda, s)$ is equivalent to finding the value that maximizes $f_\lambda(s)$. When we set the derivative of $f_\lambda(s)$ equal to zero, we get :

$$\lambda_{opt} = \frac{\mu'(s)}{n} \tag{4.22}$$

We do not try to invert this relation. We will cover a large range of values of s, and (4.22) will give us the value λ_{opt} of λ for which $B(\lambda, s)$ is optimal. Before doing this, we have to make sure that (4.22) is one-to-one in order to cover the whole range of values assumed by λ between 0 and 1/2 and that $f_\lambda(s)$ is a convex $\cap$ function of s.

As $g(s)$ is the moment generating function of the constituent code, it can be written as :

$$g(s) = \sum_{i=0}^{n} g_i e^{is} = \frac{1}{2^k}\sum_{i=0}^{n} w_i e^{is} \tag{4.23}$$

where w_i is the number of codewords of $\mathcal{C}_0$ of weight i. We have :

$$\mu'(s) = \frac{g'(s)}{g(s)} = \frac{\sum_{i=0}^{n} i g_i e^{is}}{\sum_{i=0}^{n} g_i e^{is}} \tag{4.24}$$

and :

$$\begin{aligned}\mu''(s) &= \frac{\left(\sum_{i=0}^{n} i^2 g_i e^{is}\right)\left(\sum_{i=0}^{n} g_i e^{is}\right) - \left(\sum_{i=0}^{n} i g_i e^{is}\right)^2}{\left(\sum_{i=0}^{n} g_i e^{is}\right)^2} \\ &= \frac{\sum_{l=0}^{2n} u_l e^{ls}}{\left(\sum_{i=0}^{n} g_i e^{is}\right)^2}\end{aligned}$$

where, for each $l \in [0, \cdots, 2n]$ u_l equals :

$$\begin{aligned}
u_l &= \sum_{j=0}^{l} j^2 g_j g_{l-j} - \sum_{j=0}^{l} j(l-j) g_j g_{l-j} \\
&= \sum_{j=0}^{l} g_j g_{l-j}\; j(2j-l) \\
&= \sum_{j=0}^{\lfloor \frac{l}{2} \rfloor} g_j g_{l-j}\; j(2j-l) + (l-j)(l-2j) \\
&= \sum_{j=0}^{\lfloor \frac{l}{2} \rfloor} g_j g_{l-j}\; (2j-l)^2 \qquad (4.25)
\end{aligned}$$

This proves that each u_l is non-negative, and so is $\mu''(s)$ for all s. Thus, $\lambda_{opt} = \mu'(s)/n$ is a monotonic increasing function of s. We have :

$$\begin{aligned}
\lambda_{opt} &= \frac{\sum_{i=1}^{n} i g_i e^{is}}{n \sum_{i=0}^{n} g_i e^{is}} \\
\lambda_{opt} &\underset{s \to -\infty}{\longrightarrow} 0 \\
\lambda_{opt} &\underset{s \to +\infty}{\longrightarrow} 1
\end{aligned} \qquad (4.26)$$

Figure 4.6 shows $\lambda_{opt}(s)$ for a specific constituent code.

Let us now show that $f_\lambda(s)$ is a convex $\cap$ function of s. We have :

$$f''_\lambda(s) = -\frac{J}{n}\mu''(s) \qquad (4.27)$$

Hence, $f''_\lambda(s)$ is non-positive, and $f'_\lambda(s)$ is a monotonic increasing function, which is negative for all values of s such that $\mu'(s)/n < \lambda$ and positive for all values of s such that $\mu'(s)/n > \lambda$. Consequently, $f_\lambda(s)$ and $B(\lambda, s)$ are convex $\cap$ functions of s, with their maximum value satisfying (4.22).

We can now calculate the asymptotic lower bound on the minimum distance of GLD codes as follows :

1. Start with a negative value of s.
2. Calculate λ_{opt} thanks to (4.22).

3. Calculate $B_{opt}(s) = B(\lambda_{opt}, s)$.

4. If $B_{opt}(s) > 0$, increase the value of s and return to 2, else $\delta = \lambda_{opt}$ is an asymptotical lower bound on the normalized minimum distance of the GLD code.

Constituent code $\mathcal{C}_0$	Rate of the GLD Code	δ
Hamming $(7, 4, 3)$	0.143	0.187
Hamming $(15, 11, 3)$	0.467	0.026
Hamming $(31, 26, 3)$	0.677	5.11e-3
Hamming $(63, 57, 3)$	0.810	1.14e-3
Extended Hamming $(16, 11, 4)$	0.375	0.072
Extended Hamming $(32, 26, 4)$	0.625	0.015
Extended Hamming $(64, 57, 4)$	0.781	3.40e-3
BCH $(t = 2)$ $(31, 21, 5)$	0.355	0.116
BCH $(t = 2)$ $(63, 51, 5)$	0.619	0.031
Extended BCH $(32, 21, 6)$	0.313	0.143
Extended BCH $(64, 51, 6)$	0.594	0.038

Table 4.5: Asymptotic lower bounds on the normalized minimum Hamming distance δ of GLD codes with $J = 2$ levels

Constituent code $\mathcal{C}_0$	Rate of the GLD Code	δ
Hamming $(15, 11, 3)$	0.2	0.231
Hamming $(31, 26, 3)$	0.517	0.080
Hamming $(63, 57, 3)$	0.715	0.030
Extended Hamming $(16, 11, 4)$	0.063	0.354
Extended Hamming $(32, 26, 4)$	0.438	0.115
Extended Hamming $(64, 57, 4)$	0.672	0.043
BCH $(t = 2)$ $(31, 21, 5)$	0.032	0.395
BCH $(t = 2)$ $(63, 51, 5)$	0.429	0.132
Extended BCH $(64, 51, 6)$	0.391	0.153

Table 4.6: Asymptotic lower bounds on the normalized minimum Hamming distance δ of 3-level GLD codes

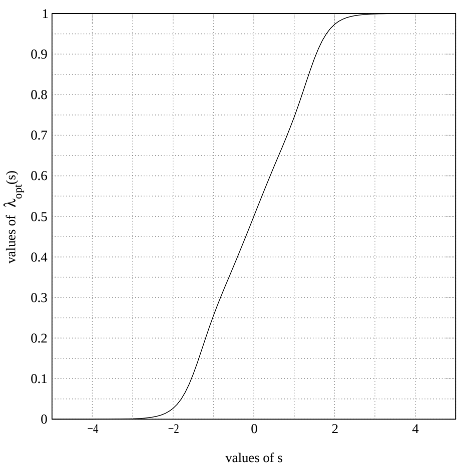


Figure 4.6: $\lambda_{opt}(s)$ for the Hamming $(15, 11, 3)$ constituent code.

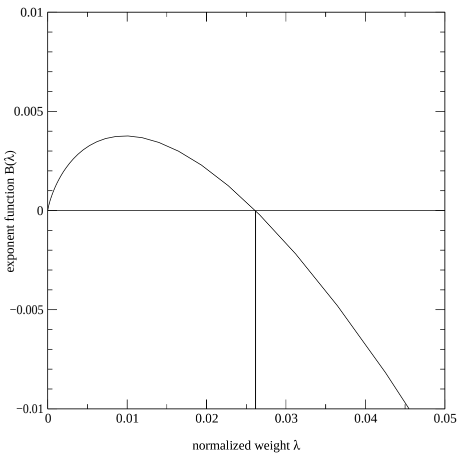


Figure 4.7: $B(\lambda_{opt}, s)$ plotted as a function of λ_{opt} for the 2-level GLD code based on the Hamming $(15, 11, 3)$ constituent code

Tables 4.5 and 4.6 show the values of δ found by this method for different constituent codes, with respectively $J = 2$ and $J = 3$ levels. All the values found are strictly positive, which means that GLD codes are asymptotically good. It is interesting to compare these values with the Varshamov-Gilbert bound ([11] p. 251). Figure 4.8 shows that GLD codes are not far from it, and even become quite close when $J = 3$ or $t = 2$ (double error-correcting BCH constituent codes).

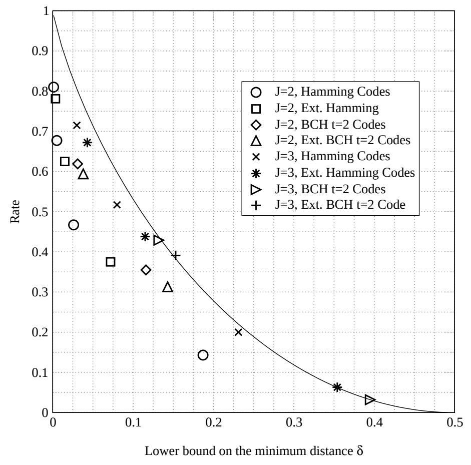


Figure 4.8: GLD asymptotical lower bounds on minimum distance compared with the Varshamov-Gilbert bound

4.2.3 BSC channel threshold

In this section, we consider communication over a BSC channel with transition probability p. Let us compute the maximum value of p such that the word error probability P_{ew} for ML decoding of GLD codes goes to zero when N is arbitrarily large.

In a first step, we will derive an upper bound of P_{ew} for any value of N, depending only on the weight distribution $N(l)$ of the GLD code. In a second step, we will use the result (4.20) on the asymptotic average distribution of GLD codes to compute $\overline{P_{ew}^{\infty}}$, the asymptotic average error probability of ML decoding of GLD codes, and compare it with the BSC capacity.

Let us assume that the all-zero word $\mathbf{0}$ of a linear binary block code $\mathcal{C}$ of length N is transmitted over the BSC channel. Let E be the ensemble of all possible error vectors, that is :

$$E = \mathrm{GF}(2)^N \setminus \mathcal{V}_{\mathbf{0}} \tag{4.28}$$

where $\mathcal{V}_{\mathbf{0}}$ is the Voronoi region of $\mathbf{0}$. Let us partition E with respect to the Hamming weight of its vectors $E = \bigcup_{i=1}^{N} E_i$ as :

$$\begin{aligned} E_i &= \{\underline{e} \in E / w(\underline{e}) = i\} \\ &= \left\{\underline{e} \in \mathrm{GF}(2)^N / w(\underline{e}) = i \text{ and } \exists \underline{c} \neq \mathbf{0} \in \mathcal{C} / d_H(\underline{e}, \underline{c}) \leq d_H(\underline{e}, 0)\right\} \end{aligned}$$

where $w()$ is the Hamming weight of a vector and $d_H(,)$ is the Hamming distance of two vectors. We have :

$$P_{ew} = \sum_{\underline{e} \in E} p(\underline{e}) = \sum_{i=1}^{N} \sum_{\underline{e} \in E_i} p(\underline{e}) = \sum_{i=1}^{N} \sum_{\underline{e} \in E_i} p^i (1-p)^{N-i} = \sum_{i=1}^{N} |E_i|\, p^i (1-p)^{N-i} \tag{4.29}$$

Let us denote by $E_i(\underline{c}_j)$ the set of error vectors of weight i that lead to the decoding of the codeword $\underline{c}_j$ of weight j :

$$E_i(\underline{c}_j) = \left\{\underline{e} \in E_i / d_H(\underline{e}, \underline{c}_j) \leq i\right\} \tag{4.30}$$

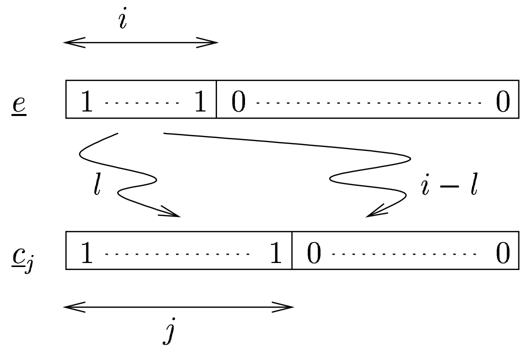


Figure 4.9: Error vector $\underline{e}$ and codeword $\underline{c}_j$. The 1s of each vector are drawn at the first positions of the vectors to help understanding.

Let us denote by l the number of 1s that an element $\underline{e}$ of $E_i(\underline{c}_j)$ has in common with $\underline{c}_j$ (see Figure 4.9). Hence, $\underline{e}$ has $i-l$ ones where $\underline{c}_j$ has zeros.

The Hamming distance between $\underline{e}$ and $\underline{c}_j$ is :

$$d_H(\underline{e}, \underline{c}_j) = j - l \; + i - l = i + j - 2l \tag{4.31}$$

(4.30) leads to the following inequality for l :

$$\lceil \frac{j}{2} \rceil \leq l \leq \min(i, j) \tag{4.32}$$

That means that an error occurs when at least half of the 1-bits of a codeword are covered by an error vector. The same argument is used on the ensemble of the linear binary block codes to derive the random-coding upper bound on the error probability in [79] pp. 92-99. Hence, the cardinality of $E_i(\underline{c}_j)$ equals :

$$\left|E_i(\underline{c}_j)\right| = \sum_{l=\lceil \frac{j}{2} \rceil}^{\min(i,j)} \binom{j}{l} \binom{N-j}{i-l} \tag{4.33}$$

Let us denote by $E_{i,j}$ the set of error vectors of weight i that lead to the decoding of a codeword of weight j. There are $N(j)$ such codewords. Because an error vector $\underline{e}$ of $E_{i,j}$ can be closer to two or more codewords of weight j than to the all-zero codeword, we have the following inequality :

$$|E_{i,j}| \leq N(j) \sum_{l=\lceil \frac{j}{2} \rceil}^{\min(i,j)} \binom{j}{l} \binom{N-j}{i-l} \tag{4.34}$$

We then have :

$$|E_i| \leq \sum_{j=d_{H\,min}}^{N} N(j) \sum_{l=\lceil \frac{j}{2} \rceil}^{\min(i,j)} \binom{j}{l} \binom{N-j}{i-l} \tag{4.35}$$

and :

$$P_{ew} \leq \sum_{i=d_{H\,min}}^{N} p^i (1-p)^{N-i} \sum_{j=d_{H\,min}}^{N} N(j) \sum_{l=\lceil \frac{j}{2} \rceil}^{\min(i,j)} \binom{j}{l} \binom{N-j}{i-l} \tag{4.36}$$

Note that the bounding occurs in and only in (4.34). This last inequality can be used to compute the ML decoding word error probability of any linear code of which we know the weight distribution. It is tighter than the conventional union bound on the error vectors weight combined with the Chernoff-Bhattacharrya bound on the pairwise error probability ([103] p.63) :

$$P_{ew} \leq \sum_{\ell=d_{H\,min}}^{N} N(\ell) \sqrt{4p\,(1-p)}^{\,\ell} \tag{4.37}$$

We are interested in the maximum crossover probability p of the BSC that leads to an asymptotically vanishing word error probability $\overline{P_{ew}^{\infty}}$ in (4.36) when N tends to infinity. In (4.29), we have to focus our attention on the term that occurs with the highest probability. Intuitively, when N grows, the weight i of the error vectors tends to pN. First, compute the probability that the normalized error weight $\theta = i/N$ is :

$$p - \epsilon \leq \theta \leq p + \epsilon \tag{4.38}$$

where ϵ is an arbitrary nonnegative value. It is closely related to the Chernoff bound on the tail of a binomial distribution. If η is a random variable with finite moments of all order and pdf $p(\eta)$ we have :

$$\text{Prob}\,(\eta \geq \tau) \leq \text{E}\left(e^{s(\eta-\tau)}\right) = e^{\Gamma(s)-s\tau} \tag{4.39}$$

where s is any non-negative value and $\Gamma(s) = \ln \sum_{\eta} e^{s\eta} p(\eta)$. Minimizing (4.39) on s results in :

$$\text{Prob}\,(\eta \geq \tau) \leq e^{\Gamma(s)-s\Gamma'(s)} \tag{4.40}$$

where $\tau = \Gamma'(s) = \frac{d\Gamma(s)}{ds}$. We apply this equality to the weight of an error vector of length N. We define

$$\eta = w(e) = \sum_{n=1}^{N} e_n \tag{4.41}$$

where e_n are the *iid* components of the error vector e and take the value 0 with probability $1 - p$ and 1 with probability p. Equality (4.40) can be rewritten in this case :

$$\text{Prob}\,(\eta \geq i) = \text{Prob}\,(\eta \geq \theta N) \leq e^{N[\gamma(s)-s\gamma'(s)]} \tag{4.42}$$

where

$$\gamma(s) = \frac{\Gamma(s)}{N} = \ln \sum_{e_n} e^{se_n} p(e_n) = \ln\left(1 - p + pe^s\right) \tag{4.43}$$

The optimal value of θ is :

$$\theta = \gamma'(s) = \frac{pe^s}{1 - p - e^s} \tag{4.44}$$

and leads to :

$$e^s = \frac{(1-p)\,\theta}{p\,(1-\theta)} \quad \text{and} \quad s = \ln(1-p) + \ln(\theta) - \ln(p) - \ln(1-\theta) \tag{4.45}$$

$$\gamma(s) = \ln(1-p) - \ln(1-\theta) \tag{4.46}$$

and finally (4.42) results in :

$$\text{Prob}\,(\eta \geq \theta N) \leq p^{N\theta}\,(1-p)^{N(1-\theta)}\,e^{NH(\theta)} \quad \text{for} \quad \theta > p \tag{4.47}$$

Replacing η by $N-\eta$, p by $1-p$ and θ by $1-\theta$ leads to the opposite tail bound :

$$\text{Prob}\,(\eta \leq \theta N) \leq p^{N\theta}\,(1-p)^{N(1-\theta)}\,e^{NH(\theta)} \quad \text{for} \quad \theta < p \tag{4.48}$$

The asymptotic exact expression of $\text{Prob}\,(\eta \geq \theta N)$ can be found in [48] pp.188-193. Let us take $\theta = p+\epsilon$ in (4.47) and $\theta = p-\epsilon$ in (4.48) to compute the probability that the normalized error weight is outside the interval $[p-\epsilon, p+\epsilon]$. We have :

$$\begin{aligned} \text{Prob}\,(\eta \geq (p+\epsilon)\,N) \;\leq\; & p^{Np}\,(1-p)^{N(1-p)}\,p^{N\epsilon}\,(1-p)^{N(1-\epsilon)} \\ & \times \exp\left\{N\left[H(p) + \epsilon H'(p) + \frac{\epsilon^2}{2}H''(p) + o(\epsilon^2)\right]\right\} \end{aligned} \tag{4.49}$$

As

$$H'(p) = \ln\left(\frac{1-p}{p}\right) \quad \text{and} \quad H''(p) = -\frac{1}{p\,(1-p)} \tag{4.50}$$

it follows that :

$$\text{Prob}\,(\eta \geq (p+\epsilon)\,N) \leq \exp\left\{-N\frac{\epsilon^2}{2p\,(1-p)} + No(\epsilon^2)\right\} \tag{4.51}$$

and the same bound holds for $\text{Prob}\,(\eta \leq (p-\epsilon)\,N)$. Consequently, we have :

$$\text{Prob}\,(\eta \in [0, p-\epsilon] \cup [p+\epsilon, 1]) \leq 2\exp\left\{-N\frac{\epsilon^2}{2p\,(1-p)} + No(\epsilon^2)\right\} \tag{4.52}$$

Hence, it is always possible to choose an $\epsilon > 0$ such that this bound tends to zero when N goes to infinity . We can now rewrite (4.29) as :

$$\begin{aligned} P_{ew} &= \sum_{i \in N \times [0,p-\epsilon] \cup [p+\epsilon,1]} |E_i|\,p^i(1-p)^{N-i} + \sum_{i \in N \times [p-\epsilon,p+\epsilon]} |E_i|\,p^i(1-p)^{N-i} \\ &\leq 2\exp\left\{-N\frac{\epsilon^2}{2p\,(1-p)} + No(\epsilon^2)\right\} \times 1 + 2N\epsilon\,|E_{pN}|\,p^{pN}(1-p)^{N(1-p)} \end{aligned}$$

Asymptotically, we have :

$$\begin{aligned} P_{ew}^{\infty} &\leq 2N\epsilon p^{pN}(1-p)^{N(1-p)} \sum_{j=d_{Hmin}}^{N} N(j) \sum_{l=\lceil \frac{j}{2} \rceil}^{\min(pN,j)} \binom{j}{l}\binom{N-j}{pN-l} \\ &\leq N\epsilon p^{pN}(1-p)^{N(1-p)} \sum_{\lambda=\delta}^{1} N(\lambda N) \sum_{l=\frac{\lambda}{2}}^{\min(p,\lambda)} \binom{\lambda N}{lN}\binom{N(1-\lambda)}{N(p-l)} \end{aligned} \tag{4.53}$$

The product of the binomial expressions is clearly maximum when $l = \lambda/2$ and it follows :

$$P_{ew}^{\infty} \leq N\epsilon p^{pN}(1-p)^{N(1-p)} \sum_{\lambda=\delta}^{1} N(\lambda N)\frac{\lambda}{2}2^{\lambda N}\binom{N(1-\lambda)}{N(p-\frac{\lambda}{2})} \tag{4.54}$$

Using the asymptotic average distribution $\overline{N(\ell)}$ of GLD codes (4.20) and the asymptotic equivalent of the binomial coefficient (4.18), and focusing on the exponential term, we have :

$$\overline{P_{ew}^{\infty}} \leq \sum_{\lambda=\delta}^{1} D\left(N,\lambda,p,\epsilon\right)\exp\left\{-NE\left(\lambda,s,p\right)\right\} \tag{4.55}$$

where

$$D\left(N,\lambda,p,\epsilon\right) = N\epsilon C\left(\lambda,N\right)\frac{\lambda}{2}\frac{1}{\sqrt{2\pi N\left(p-\lambda/2\right)\left(\frac{1+p-3\lambda/2}{1-\lambda}\right)}} \tag{4.56}$$

and

$$E\left(\lambda,s,p\right) = B\left(\lambda,s\right) - \lambda\ln\left(2\right) - \left(1-\lambda\right)H\left(\frac{p-\lambda/2}{1-\lambda}\right) + H(p) \tag{4.57}$$

The asymptotic average word error probability $\overline{P_{ew}^{\infty}}$ tends to zero if the smallest term $E\left(\lambda,s,p\right)$ in the exponential part is non-negative. Defining $E(p)$ as :

$$E(p) = \min_{\lambda,s}\left\{E\left(\lambda,s,p\right)\right\} \tag{4.58}$$

we have :

$$\overline{P_{ew}^{\infty}} \rightarrow 0 \iff E(p) > 0 \tag{4.59}$$

We can now find the BSC crossover probability threshold p which is the highest value for which $E(p)$ is non-negative, with an algorithm similar to the one described in Section 4.2.2 for $B(\lambda_{opt}, s)$.

1. Start with a small value of p.
2. Calculate the maximal value $E(p)$ of $E(\lambda, s, p)$ thanks to (4.57) and (4.19) by varying λ (and s as they are related).
3. If $E(p) > 0$, increase the value of p and return to 2, else $p_{thres} = p$ is an asymptotical upper bound on the crossover probability of the BSC that leads to a vanishing word error probability of ML decoding of the GLD code.

$E(p)$ is plotted in Figure 4.10 for the 2-level GLD code based on the Hamming $(7, 4, 3)$ constituent code. The BSC crossover probability threshold is $p = 0.109$ whereas the probability threshold for a code achieving the capacity at the same rate is $p(C) = 0.121$. Tables 4.7 and 4.8 show the values of p compared with $p(C)$ for different kinds of 2- or 3-level GLD codes. GLD codes thus achieve near-capacity performance when their length is arbitrarily large.

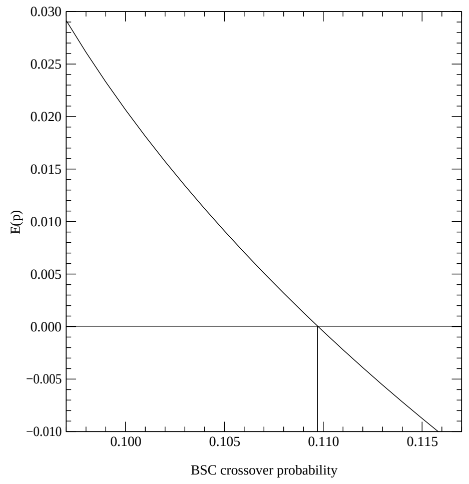


Figure 4.10: $E(p)$ vs. p for the 2-level GLD code based on the Hamming $(15, 11, 3)$ constituent code

4.2.4 Upper bound on minimum distance for a fixed length

In this section, we use the expression (4.12) of the average weight distribution of the GLD codes to compute an upper bound on the minimum Hamming distance of the ensemble of GLD codes *with fixed length.* The major difference with Section 4.2.2 is that we do not use the inequality (4.13).

The probability that the minimum distance d_{Hmin} of a linear block code $\mathcal{C}$ of length N is lower than, or equal to, D is the probability that there exists a sequence $\underline{v}$ of length N and weight l lower than, or equal to, D that is a codeword of $\mathcal{C}$:

$$\text{Prob}\left(d_{Hmin} \leq D\right) = \text{Prob}\left(\exists \underline{v} \in \text{GF}(2)^N / w(\underline{v}) = \ell \leq D \text{ and } \underline{v} \in \mathcal{C}\right) \quad (4.60)$$

Constituent code $\mathcal{C}_0$	p	$p(C)$
Hamming $(7,4,3)$	0.277	0.281
Hamming $(15,11,3)$	0.109	0.121
Hamming $(31,26,3)$	0.047	0.059
Hamming $(63,57,3)$	0.021	0.029
Extended Hamming $(16,11,4)$	0.149	0.156
Extended Hamming $(32,26,4)$	0.063	0.072
Extended Hamming $(64,57,4)$	0.027	0.035
BCH $(t=2)$ $(31,21,5)$	0.164 $\star$	0.165
BCH $(t=2)$ $(63,51,5)$	0.072	0.074
Extended BCH $(32,21,6)$	0.182 $\star$	0.183
Extended BCH $(64,51,6)$	0.080 $\star$	0.081

Table 4.7: BSC crossover probability threshold p of 2-level GLD codes compared with the threshold $p(C)$ of the code of same rate achieving capacity. $\star$ indicates that p is closer than 10^{-3} to $p(C)$.

Constituent code $\mathcal{C}_0$	p	C
Hamming $(15,11,3)$	0.242 $\star$	0.243
Hamming $(31,26,3)$	0.104 $\star$	0.105
Extended Hamming $(16,11,4)$	0.353 $\star$	0.354
Extended Hamming $(32,26,4)$	0.131 $\star$	0.132

Table 4.8: BSC crossover probability threshold p of 3-level GLD codes compared with the threshold $p(C)$ of a code of same rate achieving capacity. $\star$ indicates that p is closer than 10^{-3} to $p(C)$.

This probability is clearly less than the sum of the probabilities that the individual sequences $\underline{v}$ of the considered weight are codewords :

$$\text{Prob}\,(d_{Hmin} \leq D) \leq \sum_{\underline{v}/\ell \leq D} \text{Prob}\,(\underline{v} \in \mathcal{C}) = \sum_{\ell=1}^{D} \binom{N}{l} P(\ell) \tag{4.61}$$

where $P(\ell)$ is defined as the probability that a sequence of weight ℓ is a codeword.

We calculated $P(\ell)$ for the average ensemble of GLD codes of fixed length N, level number J and constituent code $\mathcal{C}_0$ as a function of the weight enumerator function of the constituent code in (4.11). Applying (4.12) in (4.61) leads to :

$$\text{Prob}\,(d_{Hmin} \leq D) \leq \sum_{\ell=1}^{D} \overline{N(\ell)} \tag{4.62}$$

$\overline{N(\ell)}$ is directly connected to $Q(\ell)$, the normalized number of codewords of weight l of a super-code as defined in (4.9). We no longer use the approximation (4.13), but we calculate with symbolic tools its exact value. It is the coefficient of degree l of the N/n-time convolutions of $g(s)$, the moment generating function of the constituent code $\mathcal{C}_0$. Although limited by computer capacity[6], this method can lead to :

- the complete weight distribution $\overline{N(\ell)}$ of the GLD code (see the next section for its application to ML decoding).
- an *upper bound* of its minimum Hamming distance by taking the right hand side of (4.62) equal to 1 as follows.

We have :

$$d_{Hmin} \leq \Delta \tag{4.63}$$

where Δ is the smallest integer such that

$$\lfloor \sum_{\ell=1}^{\Delta} N\,(\ell) \rfloor = 1 \tag{4.64}$$

[6]thanks to a numerical multi-precision library, and symbolic polynomial tools specially developed, it is possible to reach in a reasonable time the exact average weight distribution $\overline{N(\ell)}$ for codes up to length $N = 7000$.

Figure 4.11 shows this bound Δ calculated for two sets of 2-level GLD codes of length N in the range $200 \cdots 600$. The first one is based on the $(20, 15, 3)$ constituent code, a $(31, 26, 3)$ shortened Hamming code. The second one is based on the $(24, 18, 4)$ constituent code, an extended then shortened (32,26,4) Hamming code. Both of them have an overall rate of $R = 1/2$.

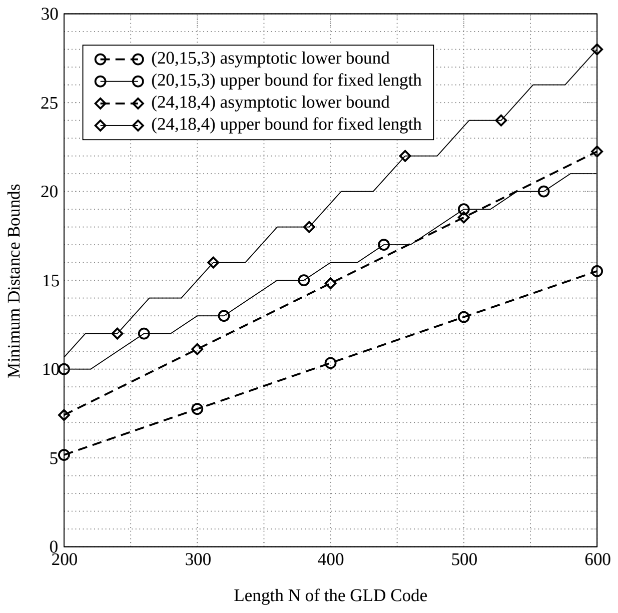


Figure 4.11: Upper bound for fixed length and asymptotic lower bound on the minimum Hamming distance of two GLD codes with $J = 2$ levels

The granularity of the curves is due to the floor function in (4.64) and to the fact that the length of a GLD code is a multiple of the constituent code length. These upper bounds are compared with the asymptotical lower bounds derived in Section 4.2.2 which are $\delta = 2.586e - 2$ for the first GLD code and $\delta = 3.708e - 2$ for the second. The average minimum distance of these codes is clearly a linear function of their length.

4.2.5 ML decoding error probability over the AWGN channel

The exact average weight distribution $\overline{N(\ell)}$ can also be used for computing the average bit error probability of ML decoding of GLD codes. Indeed, the interleaver acts on all the coded bits[7] so that they are equally protected (this statement also has been verified by computer simulation).

Hence, from any bound on the word error probability over the AWGN channel, we can derive an upper bound on the average bit error probability $\overline{P_{eb}}$ without having to compute the input-output weight enumerator function (IOWEF) as in the conventional methods used to bound the ML decoding performance of turbo-codes ([14] for parallel concatenated convolutional codes, [16] and [104] for serially concatenated convolutional codes).

In all this section, we assume that the codewords of the GLD code are transmitted as a set of N antipodal signals of baseband energy $E_s = 1$ (bit 0 associated with symbol -1 and bit 1 with symbol 1). Hence, the E_b term in the signal to noise ratio E_b/N_0 equals $E_b = 1/2R$.

Union Bound

The simpler bounds are derived directly from the union bound :

$$\overline{P_{ew}} \leq \sum_{\ell=1}^{N} \overline{N(\ell)} \times \frac{1}{2}\text{erfc}\left(\sqrt{R\ell\frac{E_b}{N_0}}\right) \tag{4.65}$$

$$\overline{P_{eb}} \leq \sum_{\ell=1}^{N} \frac{\ell}{N} \times \overline{N(\ell)} \times \frac{1}{2}\text{erfc}\left(\sqrt{R\ell\frac{E_b}{N_0}}\right) \tag{4.66}$$

But their classical drawback is that they are not tight and even diverge for low values of the signal-to-noise ratio per information bit.

[7]this is a major difference with classical compound codes such as parallel or serial concatenated convolutional codes or product codes.

Improved bound based on Gallager's bound

In [37], Duman & Salehi derived an upper bound on the word (and bit) error probability of turbo codes with ML decoding using a modified version of Gallager's bound [48] rather than the standard union bound. Their method is directly applicable to GLD codes.

The broad outlines of this bound are the partition of the code to constant-weight sub-codes, the application of Gallager's bound on each sub-code and finally the union bound to get an upper bound on the word error probability of the overall code.

The code $\mathcal{C}$ is partitioned in the set of sub-codes $\mathcal{C}_\ell$, $\ell = 1, \cdots, N$ defined as the collection of the all-zero codeword together with all the codewords of weight ℓ. Note that $\mathcal{C}_\ell$ is not necessarily linear. Let us denote by D_i the Voronoi region associated with $\underline{c}_i$, $i = 1, \cdots, N$ considered as a codeword of $\mathcal{C}$, and denote by D_i' its Voronoi region when it is considered as a codeword of $\mathcal{C}_\ell$. It is clear that for all i :

$$D_i \subseteq D_i' \tag{4.67}$$

Assuming the all-zero codeword $\underline{c}_0$ is emitted, the union bound gives :

$$P_{ew} = \sum_{i=1}^{N} \int_{y \in D_i} p(y|\underline{c}_0)dy \leq \sum_{\ell=1}^{N} \sum_{\substack{\underline{c}_i \in \mathcal{C}_\ell \\ \underline{c}_i \neq \underline{c}_0}} \int_{y \in D_i'} p(y|\underline{c}_0)dy \triangleq \sum_{\ell=1}^{N} P_{ew}^{(\ell)} \tag{4.68}$$

where $p(y|\underline{c}_0)$ is the likelihood of the codeword $\underline{c}_0$. Duman & Saheli applied Gallager's bound to $P_{ew}^{(\ell)}$, and found after some manipulations :

$$\begin{aligned} P_{ew}^{(\ell)} \quad &\leq \quad N(\ell)^\rho \alpha^{-N\frac{1-\rho}{2}} \left(\alpha - \frac{\alpha-1}{\rho}\right)^{-N\frac{\ell}{2}} \\ &\times\exp\left\{NR\frac{E_b}{N_0}\left[-1 + \frac{\beta^2}{\alpha}(1-\rho) + \rho\left(1-\frac{\ell}{N}\right)\frac{\left(\beta + \frac{1-\beta}{\rho}\right)^2}{\alpha - \frac{\alpha-1}{\rho}}\right]\right\} \end{aligned} \tag{4.69}$$

for the AWGN channel case, where $\beta = (1-\ell/N)/[(1/\alpha) - (\ell/N)(1-\rho)]$, $0 < \rho \leq 1$, and $0 < \alpha \leq 1/(1-\rho)$. α and β are two parameters that have to be optimized to find the lowest value of $P_{ew}^{(\ell)}$.

Consequently, and using the average weight distribution $\overline{N(\ell)}$ of the GLD codes, we have the following improved upper bounds on the ML decoding word- and bit-error probabilities :

$$\overline{P_{ew}} \leq \sum_{\ell=1}^{N} \min_{\substack{0<\rho\leq 1 \\ 0<\alpha\leq 1/(1-\rho)}} \overline{P_{ew}^{(\ell)}} \tag{4.70}$$

$$\overline{P_{eb}} \leq \sum_{\ell=1}^{N} \frac{\ell}{N} \min_{\substack{0<\rho\leq 1 \\ 0<\alpha\leq 1/(1-\rho)}} \overline{P_{ew}^{(\ell)}} \tag{4.71}$$

where $\overline{P_{ew}^{(\ell)}}$ is equal to the expression (4.69) where $N(\ell)$ is replaced by $\overline{N(\ell)}$.

Tangential Sphere Bound

An improved version of the tangential bound of Berlekamp [20] (valid for signals set of equal energy) has been presented by Poltyrev ([81] and [54]) and applied to turbo-codes by Sason and Shamai ([92] and [93]). This *tangential sphere bound* is tighter than the Duman & Saheli bound. As all the codewords

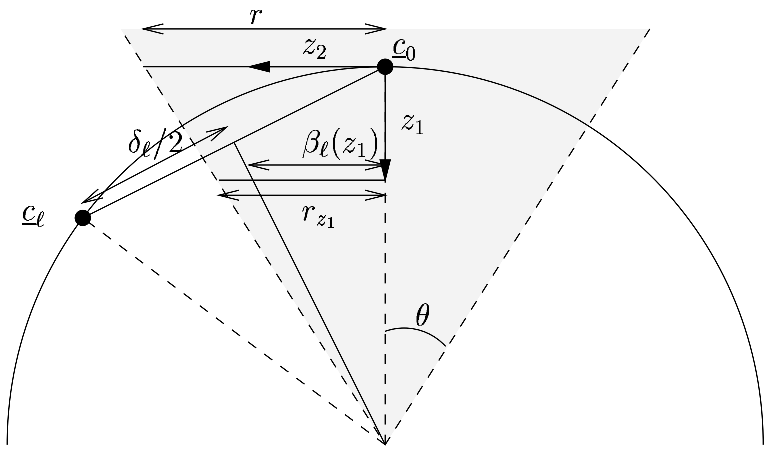


Figure 4.12: Tangential Sphere Bound

have the same energy N, they can be represented as points belonging to the sphere of radius $\sqrt{N}$. The Euclidean distance between two codewords with Hamming distance ℓ is $\delta_\ell = 2\sqrt{\ell N}$. Let $z = \sum_{i=1}^{N} z_i$ be the N-length AWGN vector. The probability of ML decoding error P_{ew} can be written as follows :

$$\begin{aligned} P_{ew} &= \text{Prob}\,(\mathbf{E}|z \in C_\theta)\,\text{Prob}\,(z \in C_\theta) + \text{Prob}\,(\mathbf{E}|z \in C_\theta)\,\text{Prob}\,(z \notin C_\theta) \\ &\leq \min_\theta \{\text{Prob}\,(\mathbf{E}|z \in C_\theta)\,\text{Prob}\,(z \in C_\theta) + \text{Prob}\,(z \notin C_\theta)\} \end{aligned} \tag{4.72}$$

where C_θ is the N-dimensional cone of half-angle θ centered at the origin and pointing to $\underline{c}_0$ (the all-zero transmitted codeword), and $\mathbf{E}$ the ensemble of error events. Let z_1 be the noise component in the origin direction, and z_2 the noise component orthogonal to z_1 and defining together with z_1 the plane containing the origin, $\underline{c}_0$ and $\underline{c}_\ell$, a codeword of weight ℓ (see Figure 4.12).

Let us start by fixing z_1 and defining $y = \sum_{i=2}^N z_i$ and $y_1 = \sum_{i=3}^N z_i$. They have a χ^2 distribution with respectively $N-1$ and $N-2$ degrees of freedom. The probability of a decoding error given z_1 is :

$$\begin{aligned} P_{ew}(z_1) &\leq \min_\theta \left\{ \sum_{\ell:\beta_\ell(z_1)<r_{z_1}} N(\ell)\text{Prob}\left(\beta_\ell(z_1) < z_2 \leq r_{z_1}, y \leq r_{z_1}^2\right) + \text{Prob}\left(y > r_{z_1}^2\right) \right\} \\ &\leq \min_\theta \left\{ \begin{array}{c} \sum_{\ell:\beta_\ell(z_1)<r_{z_1}} N(\ell)\text{Prob}\left(\beta_\ell(z_1) < z_2 \leq r_{z_1}, y_1 \leq r_{z_1}^2 - \beta_\ell(z_1)^2\right) \\ +\text{Prob}\left(y > r_{z_1}^2\right) \end{array} \right\} \end{aligned} \tag{4.73}$$

where all the quantities are shown in Figure 4.12 and defined in [54]. It is shown that θ (or equivalently r) has an optimal value θ_o (resp. r_o) that minimizes $P_{ew}(z_1)$. This value only depends on the weight distribution $N(\ell)$. By summing over all possible values of y, y_1 and z_1 according to their pdf, the final result is :

$$\begin{aligned} P_{ew} &\leq \int_{-\infty}^{\infty} \left\{ \begin{array}{c} \sum_{\ell:\beta_\ell(z_1)<r_{z_1}} N(\ell) \int_{\beta_\ell(z_1)}^{r_{z_1}} p(z_2) \int_0^{r_{z_1}^2-\beta_\ell(z_1)^2} p(y_1)dy_1dz_2 \\ + \int_{r_{z_1}^2}^{\infty} p(y)dy \end{array} \right\} p(z_1)dz_1 \\ &\leq \int_{-\infty}^{\infty} \frac{e^{-\frac{z_1^2}{2\sigma^2}}dz_1}{\sqrt{2\pi\sigma^2}} \left\{ \begin{array}{c} \sum_{\ell:\beta_\ell(z_1)<r_{z_1}} N(\ell)\left[Q\left(\frac{\beta_\ell(z_1)^2}{\sigma}\right) - Q\left(\frac{r_{z_1}}{\sigma}\right)\right]\chi^2_{(N-2,\sigma)}\left(r_{z_1}^2 - \beta_\ell(z_1)^2\right) \\ +1 - \chi^2_{(N-1,\sigma)}\left(r_{z_1}^2\right) \end{array} \right\} \end{aligned} \tag{4.74}$$

where $\chi^2_{(n,\sigma)}$ is the cdf of a random variable following a central χ^2 distribution with n degrees of freedom and standard deviation σ :

$$\chi^2_{(n,\sigma)}(x) = \frac{1}{2^{\frac{n}{2}}\sigma^n\Gamma\left(\frac{n}{2}\right)} \int_0^x t^{\frac{n}{2}-1}e^{-\frac{t}{2\sigma^2}}dt = \frac{1}{\Gamma\left(\frac{n}{2}\right)} \int_0^{x/2\sigma^2} u^{\frac{n}{2}-1}e^{-u}du \tag{4.75}$$

which has the following equivalent when n is large [1] :

$$\chi^2_{(n,\sigma)}(x) \sim 1 - Q\left(\frac{\frac{x}{\sigma^2} - n}{\sqrt{2n}}\right) \tag{4.76}$$

The word (respectively bit) error probability bound on the ensemble of GLD code is obtained by replacing $N(\ell)$ by $\overline{N(\ell)}$ (respectively $\ell/N \times \overline{N(\ell)}$) in (4.74).

The three upper bounds on the bit error probability are plotted in Figure 4.13 for a 2-level GLD code and Hamming constituent code $(15, 11, 3)$. The repetition factor is equal to $L = 28$, leading to an $(N = 420, K = 196)$ code. They are compared with the performance of a GLD code with a fixed pseudo-random interleaver iteratively decoded as explained in Section 4.3.

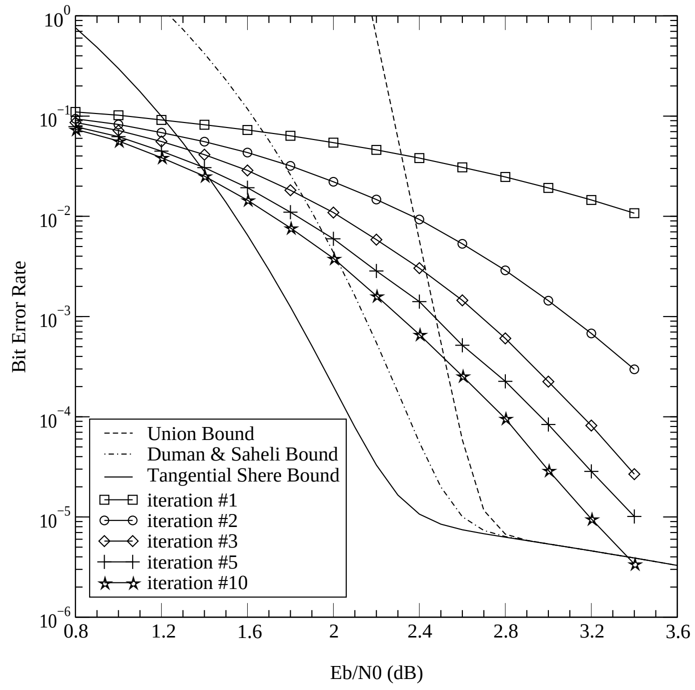


Figure 4.13: Bit Error Probability upper bounds and simulation results for the $J = 2$ GLD code with Hamming $(15, 11, 3)$ constituent code. $N = 420$, $K = 196$.

4.3 Iterative decoding of a GLD code

In this section, we restrict the presentation of the iterative decoding of GLD codes to the case of 2-level codes for three major reasons. First, we proved that GLD codes with only $J = 2$ levels are asymptotically good. Second, two levels is the best choice in terms of rate, as it decreases with J (4.6). Third, the structure, graphical representation and decoding are simpler. However, the decoding scheme presented can be applied to GLD codes with more than 2 levels as briefly discussed later.

The GLD decoding scheme is similar to Gallager's probabilistic decoding. The basic idea is the same : for each coded bit, we compute its probability given its received sample considering that it belongs to the super-code $\mathcal{C}^1$. This probability is fed to the super-code $\mathcal{C}^2$ decoder as an *a priori* information. We then compute the probability of each bit given its received sample and this side information considering it belongs to $\mathcal{C}^2$. This second probability is returned of the first super-code decoder as an *a priori* information and this process is iterated.

Each super-code decoding is performed by means of an MAP decoder[8]. Its complexity is not prohibitive if we exploit the fact that a super-code is composed of $L = N/n$ *independent* constituent codes of small length n. It is performed using L simple soft-input Ssoft-output (SISO) decoders that can furthermore work in parallel on every constituent code. In our implementation, we used forward-backward decoders [3] on the code trellis. Their complexity is still reasonable in our case as we use small length n constituent codes of high rate (see Appendix B for a description of the FB algorithm applied to block codes and an analysis of its complexity). Other SISO decoding algorithms for linear block codes, optimal or suboptimal, such as replication decoding [4] [53], Chase algorithm [31] (see also [84]), modified Battail-Fang [7] and Fossorier-Lin [43] algorithms with soft output, log-map and max-log-map simplified versions of the FB [90], or even SOVA [8] [51] could also have been used.

The super-code decoding generates for each coded bit c_i, $i = 1, \cdots, N$ an *a posteriori* probability APP_i and an extrinsic probability Ext_i. The latter is fed through the appropriate interleaver to the other super-code decoder. Decisions on each coded bits $\hat{c}_i$ can be taken at each decoding step by a threshold decoder on APP_i. One iteration step consists on the successive decoding of the first and the second super-codes, *i.e.* two decoding steps.

Let us denote by $r = (r_1, \cdots, r_N)$ the channel outputs corresponding to the codeword $c = (c_1, \cdots, c_N)$. We first compute the likelihoods $y = (y_1, \cdots, y_N)$ where $y_i = p(r_i | c_i = 0)$. The *a priori* probabilities of the $\mathcal{C}^1$ SISO

[8]A more appropriate term would be *symbol by symbol MAP Probability computer.*

decoder at the first iteration are set to 0.5 for all bits as no side information is available. The l-th iteration of the decoding scheme is presented in Figure 4.14. Remember that π_2 is the interleaver function : $\pi_2(H_1) = H_2$, hence to retrieve the structured version of the second sub-matrix H_2 at the $\mathcal{C}^2$ SISO decoder stage, the coded bits have to be interleaved with π_2^{-1}, as shown in Figure 4.15.

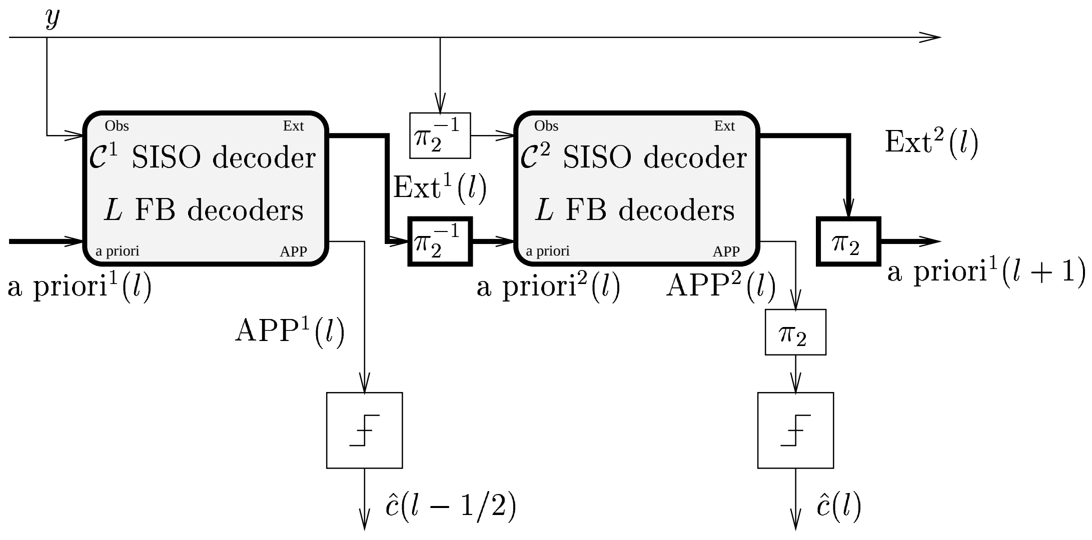
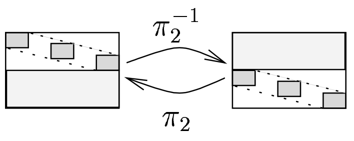


Figure 4.14: The l-th GLD decoding iteration

$$H = \begin{bmatrix} H_1 \\ H_2 = \pi_2(H_1) \end{bmatrix}$$



$$\pi_2^{-1}(H) = \begin{bmatrix} \pi_2^{-1}(H_1) \\ H_1 \end{bmatrix}$$

Figure 4.15: Permutation of the parity-check matrix of the GLD codes with $J = 2$ levels.

4.4 Iterative decoding performance

The decoding algorithm presented in the last section is clearly not optimal, even if the decoding of each super-code is so. We see in Figure 4.13 that there is a loss in a large range of E_b/N_0 between the simulation results for a fixed interleaver GLD code and the upper bound on the ML decoding error of the average ensemble of GLD codes of same length. The questions are : where does this loss come ? From the interleaver choice, or from the decoding ? The influence of the interleaver choice and *good* interleavers suitable for iterative decoding are analyzed in Chapter 5. We focus our attention in this section to the second part of the question. Independently of the interleaver choice, how far is the iterative decoding performance from that of ML ? Is it possible to derive a bound on the iterative decoding error probability ?

To get rid of the interleaver influence, we consider an interleaver of infinite length. Hence, there is no cycles that can disturb the iterative process. Obviously, the technique based on the weight distribution can no longer be applied. The basic idea of this approach is to analyze the propagation of the probabilities issued from the iterative decoding in the graph representing a GLD code. This approach is derived from the original work of Richardson and Urbanke [87] for LDPC codes. We will estimate the probability density function of the outputs of the constituent code SISO decoders in a GLD code, and analyze how these pdfs change with iterations. This work has been jointly achieved with Boutros and Vialle [27] where a universal propagation formula is derived for compound codes.

4.4.1 Probability distribution of the output of the SISO decoder

In this section, we consider the communication over the AWGN channel of a binary stream coded with a constituent linear block code $\mathcal{C}_0$ of length n. We assume that the all-zero codeword is transmitted. The received codewords are decoded with a SISO decoder. Our aim is to characterize the probability density function of the APP at the output of the decoder. This APP is considered as a random variable.

Let us denote by $a(c_j = 1)$ the product of the likelihood and the *a priori* probability of c_j, $j = 1, \cdots, n$: $a(c_j = 1) = p(y|c_j = 1)\pi(c_j = 1)$. We define b_j as the log-ratio of $a(c_j)$:

$$b_j = \ln\left(\frac{a(c_j = 1)}{a(c_j = 0)}\right) \tag{4.77}$$

We have the following basic relation :

$$a(c_j = 1) = e^{b_j} a(c_j = 0) \tag{4.78}$$

The APP of any bit c_i is :

$$\begin{aligned} \mathrm{APP}(c_i = 1) &\propto \sum_{\substack{\underline{c}\in\mathcal{C}_0 \\ \underline{c}_i=1}} \prod_{j=1}^{n} a(c_j = \underline{c}_j) = a(c_i = 1) \sum_{\substack{\underline{c}\in\mathcal{C}_0 \\ \underline{c}_i=1}} \prod_{\substack{j=1 \\ j\neq i}}^{n} a(c_j = \underline{c}_j) && (4.79) \\ &\propto a(c_i = 1)\mathrm{Ext}(c_i = 1) && (4.80) \end{aligned}$$

where $\underline{c}_j$ is the value (0 or 1) of the j-th bit of the codeword $\underline{c}$. LR(Ext)(c_i), the log-ratio of the extrinsic probabilities of c_i, is :

$$\begin{aligned} \mathrm{LR}(\mathrm{Ext})(c_i) &= \ln\left\{\frac{\sum\limits_{\substack{\underline{c}\in\mathcal{C}_0 \\ \underline{c}_i=1}} \prod\limits_{\substack{j=1 \\ j\neq i}}^{n} a(c_j = \underline{c}_j)}{\sum\limits_{\substack{\underline{c}\in\mathcal{C}_0 \\ \underline{c}_i=0}} \prod\limits_{\substack{j=1 \\ j\neq i}}^{n} a(c_j = \underline{c}_j)}\right\} && (4.81) \\ &= \ln\left\{\frac{\sum\limits_{\substack{\underline{c}\in\mathcal{C}_0 \\ \underline{c}_i=1}} \left[\prod\limits_{\substack{j=1,j\neq i \\ \underline{c}_j=1}}^{n} a(c_j = 1) \prod\limits_{\substack{j=1,j\neq i \\ \underline{c}_j=0}}^{n} a(c_j = 0)\right]}{\sum\limits_{\substack{\underline{c}\in\mathcal{C}_0 \\ \underline{c}_i=0}} \left[\prod\limits_{\substack{j=1,j\neq i \\ \underline{c}_j=1}}^{n} a(c_j = 1) \prod\limits_{\substack{j=1,j\neq i \\ \underline{c}_j=0}}^{n} a(c_j = 0)\right]}\right\} && (4.82) \\ &= \ln\left\{\frac{\sum\limits_{\substack{\underline{c}\in\mathcal{C}_0 \\ \underline{c}_i=1}} \exp\left(\sum\limits_{\substack{j=1,j\neq i \\ \underline{c}_j=1}}^{n} b_j\right)}{\sum\limits_{\substack{\underline{c}\in\mathcal{C}_0 \\ \underline{c}_i=0}} \exp\left(\sum\limits_{\substack{j=1,j\neq i \\ \underline{c}_j=1}}^{n} b_j\right)}\right\} && (4.83) \end{aligned}$$

Example :
Let us take the $(7,4,3)$ Hamming code whose codewords are :

c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_0	c_1	c_2	c_3	c_4	c_5	c_6
0	0	0	0	0	0	0	1	0	0	0	1	0	1
0	0	0	1	0	1	1	1	0	0	1	1	1	0
0	0	1	0	1	1	0	1	0	1	0	0	1	1
0	0	1	1	1	0	1	1	0	1	1	0	0	0
0	1	0	0	1	1	1	1	1	0	0	0	1	0
0	1	0	1	1	0	0	1	1	0	1	0	0	1
0	1	1	0	0	0	1	1	1	1	0	1	0	0
0	1	1	1	0	1	0	1	1	1	1	1	1	1

and compute the extrinsic probabilities of c_0 :

$$\mathrm{Ext}(c_0=1) \propto \left\{\begin{array}{ll} & a(c_4=1)a(c_6=1)\cdot a(c_1=0)a(c_2=0)a(c_3=0)a(c_5=0) \\ + & a(c_2=1)a(c_3=1)\cdot a(c_1=0)a(c_4=0)a(c_5=0)a(c_6=0) \\ + & a(c_1=1)a(c_5=1)\cdot a(c_2=0)a(c_3=0)a(c_4=0)a(c_6=0) \\ + & a(c_3=1)a(c_4=1)a(c_5=1)\cdot a(c_1=0)a(c_2=0)a(c_6=0) \\ + & a(c_2=1)a(c_5=1)a(c_6=1)\cdot a(c_1=0)a(c_3=0)a(c_4=0) \\ + & a(c_1=1)a(c_3=1)a(c_6=1)\cdot a(c_2=0)a(c_4=0)a(c_5=0) \\ + & a(c_1=1)a(c_2=1)a(c_4=1)\cdot a(c_3=0)a(c_5=0)a(c_6=0) \\ + & a(c_1=1)a(c_2=1)a(c_3=1)a(c_4=1)a(c_5=1)a(c_6=1) \end{array}\right\}$$

$$\mathrm{Ext}(c_0=0) \propto \left\{\begin{array}{ll} + & a(c_1=0)a(c_2=0)a(c_3=0)a(c_4=0)a(c_5=0)a(c_6=0) \\ + & a(c_3=1)a(c_5=1)a(c_6=1)\cdot a(c_1=0)a(c_2=0)a(c_4=0) \\ + & a(c_2=1)a(c_4=1)a(c_5=1)\cdot a(c_1=0)a(c_3=0)a(c_6=0) \\ + & a(c_1=1)a(c_3=1)a(c_4=1)\cdot a(c_2=0)a(c_5=0)a(c_6=0) \\ + & a(c_1=1)a(c_2=1)a(c_6=1)\cdot a(c_3=0)a(c_4=0)a(c_5=0) \\ + & a(c_2=1)a(c_3=1)a(c_4=1)a(c_6=1)\cdot a(c_1=0)a(c_5=0) \\ + & a(c_1=1)a(c_4=1)a(c_5=1)a(c_6=1)\cdot a(c_2=0)a(c_3=0) \\ + & a(c_1=1)a(c_2=1)a(c_3=1)a(c_5=1)\cdot a(c_4=0)a(c_6=0) \end{array}\right\}$$

$$\mathrm{LR(Ext)}(c_0) = \ln\left\{\frac{\begin{array}{ll} & e^{b_4+b_6}+e^{b_2+b_3}+e^{b_1+b_5} \\ + & e^{b_3+b_4+b_5}+e^{b_2+b_5+b_6} \\ + & e^{b_1+b_3+b_6}+e^{b_1+b_2+b_4} \\ + & e^{b_1+b_2+b_3+b_4+b_5+b_6} \end{array}}{\begin{array}{ll} 1+ & e^{b_3+b_5+b_6}+e^{b_2+b_4+b_5} \\ + & e^{b_1+b_3+b_4}+e^{b_1+b_2+b_6} \\ + & e^{b_2+b_3+b_4+b_6}+e^{b_1+b_4+b_5+b_6} \\ + & e^{b_1+b_2+b_3+b_5} \end{array}}\right\} \qquad (4.84)$$

In fact, it is easy to see that the extrinsic probabilities log-ratio of each bit

c_i is :

$$\mathrm{LR(Ext)}(c_i) = \ln \left\{ \frac{\begin{array}{rl} & e^{b_{i\oplus 4}+b_{i\oplus 6}} + e^{b_{i\oplus 2}+b_{i\oplus 3}} + e^{b_{i\oplus 1}+b_{i\oplus 5}} \\ + & e^{b_{i\oplus 3}+b_{i\oplus 4}+b_{i\oplus 5}} + e^{b_{i\oplus 2}+b_{i\oplus 5}+b_{i\oplus 6}} \\ + & e^{b_{i\oplus 1}+b_{i\oplus 3}+b_{i\oplus 6}} + e^{b_{i\oplus 1}+b_{i\oplus 2}+b_{i\oplus 4}} \\ + & e^{b_{i\oplus 1}+b_{i\oplus 2}+b_{i\oplus 3}+b_{i\oplus 4}+b_{i\oplus 5}+b_{i\oplus 6}} \end{array}}{\begin{array}{rl} 1+ & e^{b_{i\oplus 3}+b_{i\oplus 5}+b_{i\oplus 6}} + e^{b_{i\oplus 2}+b_{i\oplus 4}+b_{i\oplus 5}} \\ + & e^{b_{i\oplus 1}+b_{i\oplus 3}+b_{i\oplus 4}} + e^{b_{i\oplus 1}+b_{i\oplus 2}+b_{i\oplus 6}} \\ + & e^{b_{i\oplus 2}+b_{i\oplus 3}+b_{i\oplus 4}+b_{i\oplus 6}} + e^{b_{i\oplus 1}+b_{i\oplus 4}+b_{i\oplus 5}+b_{i\oplus 6}} \\ + & e^{b_{i\oplus 1}+b_{i\oplus 2}+b_{i\oplus 3}+b_{i\oplus 5}} \end{array}} \right\} \tag{4.85}$$

where $\oplus$ is the addition modulo $n = 7$. $\square$

We define $f(i, \underline{b})$ as $\mathrm{LR(Ext)}(c_i) = f(i, \underline{b})$, that is the function that computes the extrinsic log-ratio of any bit c_i from the log-ratios $\underline{b} = (b_1, \cdots, b_n)$ in (4.83). Finally, we have :

$$\mathrm{LR(APP)}(c_i) = b_i + f(i, \underline{b}) \tag{4.86}$$

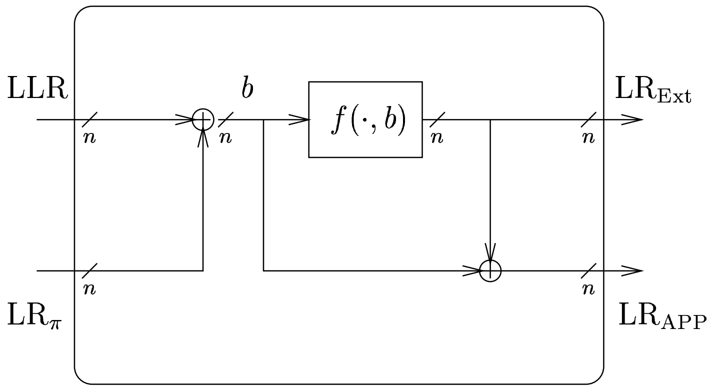


Figure 4.16: SISO decoder using log-ratios

We now consider the inputs of the SISO decoder as random variables with given probability density functions. As the all-zero codeword was transmitted over the AWGN channel, the LLRs of the received samples can be considered as iid random variables following a normal distribution $\mathcal{P}_{\mathrm{LLR}}$:

$$\mathcal{P}_{\mathrm{LLR}} \sim \mathcal{N}\left(-\frac{2}{\sigma^2}, \frac{4}{\sigma^2}\right) \tag{4.87}$$

where σ^2 is the variance of the noise.

Let us assume that the *a priori* probabilities $\pi(c_j = 1)$, $j = 1, \cdots, n$ are iid variables. So are their log-ratio $\mathrm{LR}(\pi)(c_j)$. Their pdf is denoted by $\mathcal{P}_{\mathrm{LR}(\pi)}$. The independence assumption is justified in our case by the fact that *a priori*'s were computed by former SISO decodings involving observations on distinct bits (let us remember that the graph is infinite and without cycles). We will discuss the identical distribution assumption later on.

Since the two inputs of the SISO decoder are iid, so are $a(c_j)$ and b_j, $j = 1, \ldots, n$. The pdf $\mathcal{P}_b$ of b is the convolution product :

$$\mathcal{P}_b = \mathcal{P}_{\mathrm{LLR}} * \mathcal{P}_{\mathrm{LR}(\pi)} \tag{4.88}$$

The analytical computation of $\mathcal{P}_{\mathrm{LR(Ext)}(c_i)}$, the pdf of the extrinsic probabilities given $\mathcal{P}_b$ is unfortunately not simply tractable. Indeed, the terms in the numerator and denominator of (4.83) ((4.85) for the example) that define $f(i, \underline{b})$ are clearly dependent, canceling any chance of simplification. Moreover, the extrinsic probabilities $\mathrm{Ext}(c_i)$ are dependent. That is also true for the APPs. However, the expression (4.83) can be literally compacted if we point out that the terms in the sums over j are independent.

Denoting by "$X^{\otimes k}$" the product of k random variables X that are iid, (4.83) can be rewritten :

$$\mathrm{LR(Ext)}(c_i) = \ln \left\{ \odot \frac{\sum\limits_{\ell=1}^{n} A^1_{i,\ell} \odot \exp{(b)}^{\otimes(\ell-1)}}{\sum\limits_{\ell=0}^{n-1} A^0_{i,\ell} \odot \exp{(b)}^{\otimes\ell}} \right\} \tag{4.89}$$

where $A^1_{i,\ell}$ denotes the number of codewords of $\mathcal{C}_0$ of weight ℓ and having $\underline{c}_i = 1$ and $A^0_{i,\ell}$ the number of codewords of weight ℓ and having $\underline{c}_i = 0$. We introduced the symbol "$\odot$" (which should be read as *Caution !*) meaning that the following terms are not independent, and thus the factorization was written for the sake of compactness.

By looking at (4.85), we see that this expression depends on the realization of $n-1$ random variables b_j of same law, but that the index i has no influence : we can drop the term "$i\oplus$" from the indices of b, and just consider $n-1$ independent random variables $b_1, \cdots, b_{n-1}$ of law $\mathcal{P}_b$. This property is due to the characteristics of the constituent code $\mathcal{C}_0$, and is called *isotropy*. Cyclic codes are proved to be isotropic by Battail [6] [9] thanks to the polynomial description of linear block codes, and Boutros and Vialle

[27]. Consequently, for such codes, the extrinsic probabilities and the APPs (which depend on another random variable b_0, thanks to (4.86)) are *identically distributed* for each coded bit.

In the infinite GLD codes, as at the first iteration the *a priori* probabilities are unknown and set to 0.5, and as we assumed that the *a priori* probabilities were iid, we just proved by recursion that for isotropic codes, the extrinsic probabilities are identically distributed at each decoding iteration step. Even if the extrinsic probabilities at the output of a SISO decoder are dependent, since they become *a priori*'s for distinct SISO decoders at the next iteration step (the graph has no cycle), the *a priori* probabilities are iid at the input of each SISO decoder at any iteration step.

To complete this analysis, we need a practical way to compute $\mathcal{P}_{\text{LR(Ext)}}$ and $\mathcal{P}_{\text{LR(APP)}}$, the common pdf of the outputs of the decoder. In terms of pdf (4.86) leads to :

$$\mathcal{P}_{\text{LR(APP)}} = \mathcal{P}_b * \mathcal{P}_{\text{LR(Ext)}} \tag{4.90}$$

as $f(i, \underline{b})$ involves the $n-1$ random variables b_j independent from b_i. We compute $\mathcal{P}_{\text{LR(Ext)}}$ by a Monte-Carlo method : we choose independently $n-1$ values of b following the law (4.88), and calculate the associated n values of LR(Ext)(c_i) thanks to (4.83) (for large constituent codes, we use directly the SISO decoder). We put these values into histograms that give us accurate estimates of the law of the LR(Ext)(c_i) when we iterate this process. If the code is isotropic, we only need to compute one law, which is common for all bits. This method is depicted in Figure 4.17.

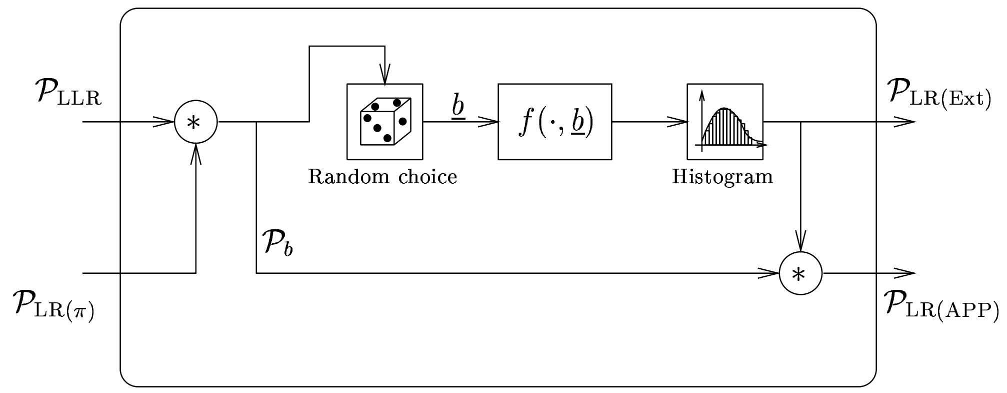

$\mathcal{P}_{\text{LLR}}$
$*$
$\underline{b}$
$f(\cdot, \underline{b})$
Random choice
Histogram
$\mathcal{P}_{\text{LR(Ext)}}$
$\mathcal{P}_b$
$*$
$\mathcal{P}_{\text{LR}(\pi)}$
$\mathcal{P}_{\text{LR(APP)}}$

Figure 4.17: Decoder output pdfs computation

4.4.2 APP propagation and performance of GLD code with infinite length

In the previous section, we exhibit a method to compute the pdf of the APP of a SISO decoder as a function of the pdfs of the inputs (channel observation and *a priori* probabilities) pdfs. To this end, we assumed that the inputs were iid. This is a valid assumption if we consider an infinite GLD code, *i.e.* an infinite graph. We showed in this case that outputs are also iid for isotropic codes. Our aim is to observe how the APP pdf changes with the decoding iteration steps or, similarly, how the APP propagates in the graph, since an iteration step can be seen as information propagation in the graph drawn at Figure 4.18 (we limit our study to GLD codes with $J = 2$ levels). This graph is constructed from the one depicted in Figure 4.3 but with infinite length. We choose one bit c_i and look at its neighborhood (Figure 4.18).

Let us compute the APP of the bit c_i at iteration m, considering it belongs to the *upper* code $\mathcal{C}_0$. The involved *a priori* probabilities of the other bits c_j, $j \neq i$ of $\mathcal{C}_0$ (or equivalently from the bits that are *below* c_i in the graph) are denoted by $\pi^m(c_j)$ and were computed at the previous iteration step $m-1$. The own *a priori* probability of c_i comes from its *lower* code $\mathcal{C}'_0$ SISO decoding at iteration $m-1$ and is denoted by $\pi^m(c_i)$. These *a priori* probabilities are combined with the observations to give a_j (and b_j), $j = 1, \cdots, n$ as defined in the previous section.

We can assign tasks to code and bit vertices in the graph of Figure 4.18 and see the iterative decoding as propagation in this graph. At iteration step m the code vertex $\mathcal{C}_0$ computes the quantity $\mathrm{Ext}^m(c_i)$ given $a^m(c_j)$, $j = 1, \cdots, n$ (or equivalently b_j) thanks to equation (4.83). The bit node c_i computes its own APP given $\mathrm{Ext}^m(c_i)$ coming up from $\mathcal{C}_0$ and its new combined extrinsic information/observation $a^{m+1}(c_i)$ that it passes to $\mathcal{C}'_0$ for the next iteration step.

We continue to assume that the all-zero codeword was transmitted, and start with unknown (0.5) *a priori* probabilities for each bit at the first iteration. Since the graph is infinite (and thus without cycles), if we choose a GLD code with isotropic constituents, we can evaluate at iteration m the pdf of the common APP log-ratio $\mathcal{P}_{\mathrm{LR}(\mathrm{APP}^m)}$ and extrinsic information of all bits using the method described in the previous section. The notion of *isotropy* reveals its meaning here : the APP (and the extrinsic information) follows the same law at a fixed iteration step anywhere in the graph.

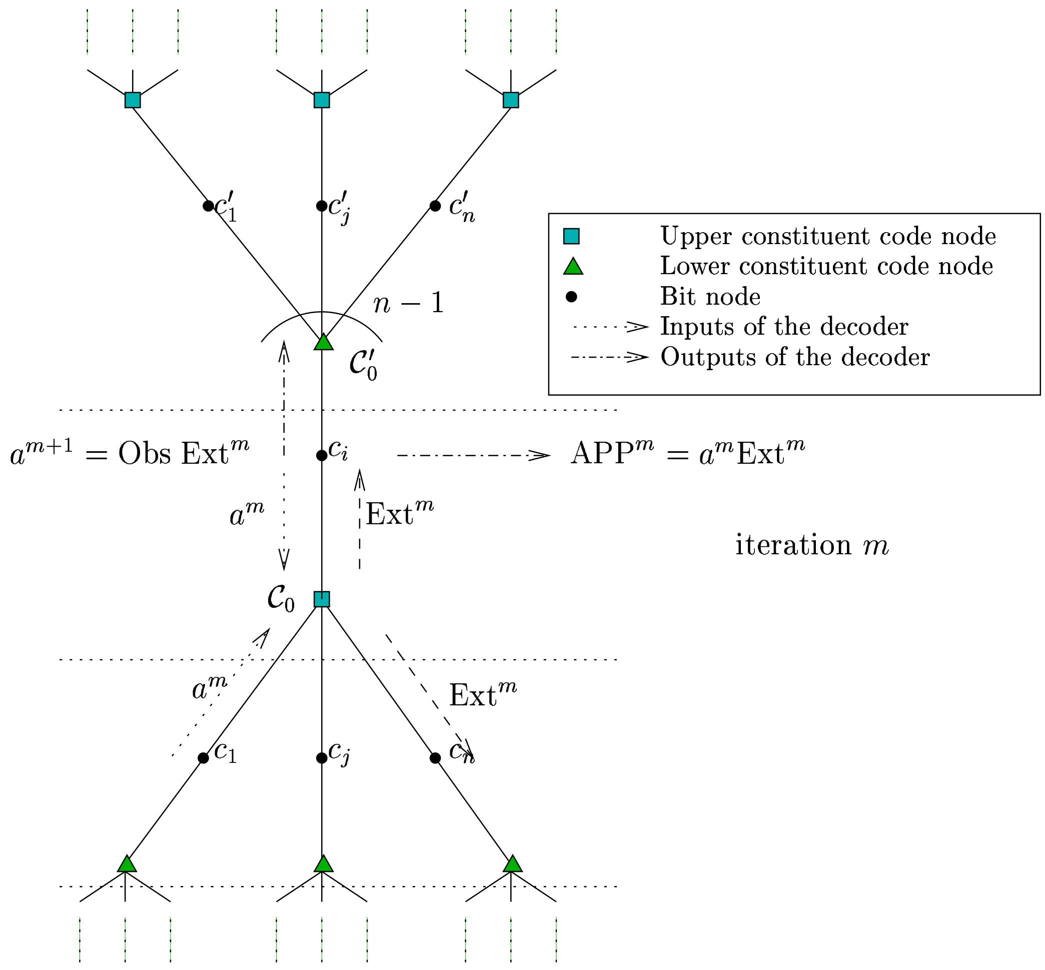


Figure 4.18: Neighborhood of the bit c_i in the infinite graph

The tail of the pdf of the APP log-ratio gives us an estimate of the bit error probability at iteration m :

$$P_{eb}^{\infty}(m) = \int_{z=0}^{\infty} \mathcal{P}_{\mathrm{LR(APP^m)}}(z)dz \tag{4.91}$$

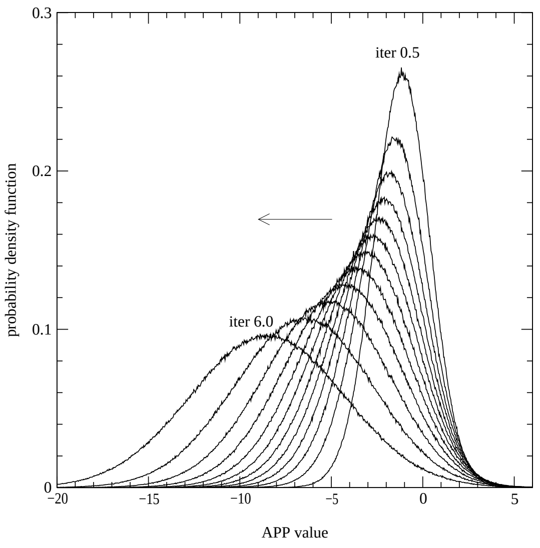


Figure 4.19: APP pdf of the infinite GLD code with $J = 2$ and Hamming $(7, 4, 3)$ constituent code for decoding iteration steps from 0.5 to 6 over the AWGN channel at $E_b/N_0 = 2.0$dB (all-zero codeword transmitted).

Figure 4.19 shows how the APP pdf changes with the iteration steps for a GLD code with $J = 2$ levels and Hamming $(7, 4, 3)$ constituent codes over the AWGN channel at $E_b/N_0 = 2.0$dB. Since the pdf shifts to the left while decoding iteration proceeds, the bit error probability decreases. A bit error probability estimate is also derived thanks to (4.91) and gives a bound on the performance of perfect iterative decoding on an infinite graph. This bound is more useful than the ones involving ML decoding as derived in Section 4.2.5 since it deals with the decoder practically used.

This result is compared with the results of simulating fixed length GLD codes with same parameters and plotted in Figure 4.20. The interleavers chosen for the fixed length GLD code are random without length-2 cycle in the compact graph representation. When the interleaver length increases, the results of simulation approach the performance of perfect iterative APP propagation. Hence, it justifies the semi-analytical method used to compute

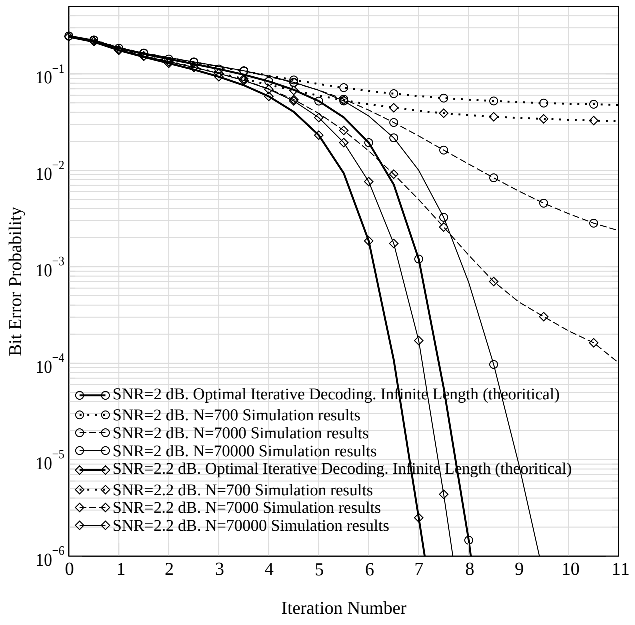


Figure 4.20: Performance of perfect iterative decoding of GLD code with infinite length compared with simulation results for fixed length. The GLD codes have $J = 2$ levels, and are constructed with Hamming $(7, 4, 3)$ constituent codes.

the pdf of the extrinsic information. The influence of the interleaver choice and its length on the iterative decoding is studied in the next Chapter.

The minimal signal-to-noise ratio that achieves zero error probability with perfect iterative decoding can be easily found by decreasing the SNR until the $P_{eb}^{\infty}(m)$ estimate no longer tends to zero for any iteration steps number m. This value for the presented GLD code is $E_b/N_0 = 1.26$dB. The threshold value for a GLD code based on $(15, 11, 3)$ constituent codes is $E_b/N_0 = 0.83$dB.

Furthermore, even if the code is not proved to be isotropic, *e.g.* extended or shortened BCH codes, the presented method is applicable and leads to valid results if the average of the different extrinsic information pdfs is taken as the common *a priori* probability pdf for all bits at the input of the next decoding stage. Generalization to GLD codes with more than $J = 2$ levels is tractable by replacing the *a priori* pdf by the convolution of the $J - 1$ *a priori* pdfs. This method has also been used successfully in other contexts such as parallel turbo codes [27] and iterative multi-user detection analysis [28].

4.5 Simulation results

We conducted comparative simulations of GLD codes in the following context :

- BPSK (or QPSK) modulation,
- AWGN channel,
- Short frame transmission : $K \approx 200$ bits,
- Total Rate : $R \approx 1/2$,
- Low-to-Moderate decoding complexity : typically no more than 64-state trellises, and much less if possible,
- Bit Error Rate : around 10^{-5}.

These requirements could be those of a geostationnary satellite return link suitable for interactivity.

Our first aim is to compare a GLD based solution with a product code solution [84] [85] [86], in terms of performance, complexity (of coding and decoding) and flexibility. We do not focus on suboptimal implementation of the decoding algorithm, nor on optimization of the interleaver choice in this section. We use for each solution a MAP SISO decoder on the constituent codes, and pick, when needed, an interleaver at random.

Competing Codes
The above requirements almost induce the product choice. The constituent code must have a rate of $r \approx 1/\sqrt{2} \approx 0.7$ and a length of $n \approx 20$. In fact, the only two possible choices are :

- shortened versions of the $(31, 26, 3)$ Hamming code. 12 bits shortening in both directions leads to a product code with the following parameters : $N = 361$, $K = 196$, $R = 0.54$. This construction is depicted in Figure 4.21 (the *left* numbers should be read). We denote this code by $\mathcal{C}^{P1}$ and its constituent codes by $\mathcal{C}_0^{P1}$.

- shortened versions of the $(32, 26, 4)$ extended Hamming code. 12 bits shortening in both directions leads to a product code with the following parameters : $N = 400$, $K = 196$, $R = 0.49$. This construction is depicted in Figure 4.21(the *right* numbers should be read). We denote this code by $\mathcal{C}^{P2}$ and its constituent codes by $\mathcal{C}_0^{P2}$.

On the other side, we choose a 2-level GLD code. Indeed, $J = 2$ results in GLD codes with the highest rate, and simplifies the decoder structure : no particular schedule has to be implemented for passing the information from one super-code to the other. Furthermore, GLD codes are proved to be asymptotically good even if $J = 2$. Hence, we want a constituent code with rate $r \approx 3/4$ to fulfill the requirement on the global code rate. Any code of rate 3/4 is eligible to become constituent code of the GLD code. The longer its length n, the shorter the repetition factor $L = N/n$ will be. We choose the shortest one with the desired rate, *i.e.* the Hamming $(15, 11, 3)$ code, denoted by $\mathcal{C}_0^{GLD}$. Its rate is slightly lower than required, but still acceptable. The parameters of the GLD code, denoted by $\mathcal{C}^{GLD}$ are : $N = 420$, $K = 196$,

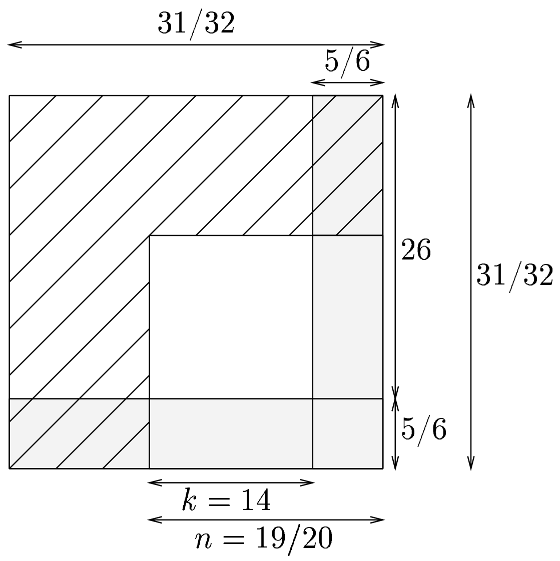


Figure 4.21: (361, 196) and (400, 196) Product Codes with respective constituent code (19, 14, 3) (shortened (31, 26, 3) Hamming code) and (20, 14, 4) (shortened (32, 26, 4) extended Hamming code). The white part represents the information bits and the shaded part the parity bits. The hatched zone represents the bits of respectively (31, 26, 3) and $(32, 26, 4)^2$ product code that have been expurgated.

$L = 28$ and $R = 0.47$. The interleaver is chosen at random[9], with the only condition that no parallel edges link two code vertices in the compact graph[10]. The three compared codes have thus exactly the same dimension. The length of $\mathcal{C}^{GLD}$ is 16% longer than the one of $\mathcal{C}^{P1}$ and 5% longer than the one of $\mathcal{C}^{P2}$.

Compared complexity

In terms of decoding complexity, the advantage of the GLD code is clear. A complete decoding iteration of $\mathcal{C}^{GLD}$ (decoding of the *upper* super-code followed by the decoding of the *lower* super code) involves the use of $2L = 56$ forward-backward algorithms. A complete study of the complexity of the FB for decoding block codes *via* their syndrome trellis is presented in Appendix B. Roughly, the complexity is linearly related to the number of states in the trellis. Since the trellis of $\mathcal{C}_0^{GLD}$ has at most $2^{n-k} = 16$ states in its central

[9]In fact, we picked 10 interleavers at random, and we used the one which leads to the best results

[10]We add this condition to have graphs with the same girth $g = 4$ for the three compared codes. See next Chapter for details

part, the total complexity of a $\mathcal{C}^{GLD}$ decoding iteration can be expressed as $2L2^{n-k} = 896$.

In comparison, a complete decoding iteration of $\mathcal{C}^{P1}$ (horizontal and vertical codes decodings) involves the use of $2n = 38$ FB decoders. Since the trellis of $\mathcal{C}_0^{P1}$ has $2^{n-k} = 32$ states in its central part, the total complexity of a $\mathcal{C}^{P}$ decoding iteration is $2n2^{n-k} = 1216$. It is 1.35 times more complex than the GLD code decoding. The same considerations applied to $\mathcal{C}^{P2}$ lead to a complexity per complete decoding iteration of $2n2^{n-k} = 2560$. It is almost 3 times more complex than the GLD code decoding. However, we have to wait for the simulations and compare the convergence speeds before to draw any conclusion.

Compared minimum Hamming distance
In terms of minimum Hamming distances, those of the two product codes are known exactly : $d_{Hmin}(\mathcal{C}^{P1}) = d_{Hmin}(\mathcal{C}_0^{P1})^2 = 9$ and $d_{Hmin}(\mathcal{C}^{P1}) = d_{Hmin}(\mathcal{C}_0^{P1})^2 = 16$. We do not know the Hamming distance of $\mathcal{C}^{GLD}$. However, we computed the average weight distribution $\overline{N(\ell)}$ over all possible interleaver choices (4.12), and using (4.64) we find an upper bound Δ on the minimum Hamming distance of the average of GLD codes with same parameters as $\mathcal{C}^{GLD}$: $\Delta = 16$. On the other hand, we can use the asymptotical average lower bound δ on minimum Hamming distance of GLD codes (Table 4.5) which is $\delta = 2.6.10^{-2}$. Hence, we have :

$$10.92 = \delta N \lesssim \overline{d_{Hmin}(\mathcal{C}^{GLD})} \lesssim \Delta = 16 \tag{4.92}$$

Consequently, we can say that, using a ML decoder, the average performance of the codes that share the same parameters as $\mathcal{C}^{GLD}$ should be bounded somewhere between the ones of $\mathcal{C}^{P1}$ and $\mathcal{C}^{P2}$. However, the decoding is iterative, and the interleaver of $\mathcal{C}^{GLD}$ is fixed (and chosen with care), hence we can expect some different conclusions.

Simulation results
Simulations were conducted for the three competing codes, within the following common context :

- FB based soft-input soft-output decoding of the constituent codes,
- Up to 12 decoding iteration steps,
- SNR per bit from 0.6 to 3.4 dB, with a 0.2 dB step,

- Monte-Carlo estimation of the BER and FER, with at least 100 erroneous blocks.

The results are presented in the following figures. Figures 4.22 (respectively Figures 4.23) show the performance (in terms of BER and FER) of $\mathcal{C}^{P1}$ (respectively $\mathcal{C}^{P2}$) with respect to the number of decoding iterations. The error rates reach stable values after the third or forth iteration. There is no need to further iterate. As predicted, $\mathcal{C}^{P2}$ is more powerful that $\mathcal{C}^{P1}$ and achieves a BER below 2.10^{-5} at $E_b/N_0 = 3.4$ dB.

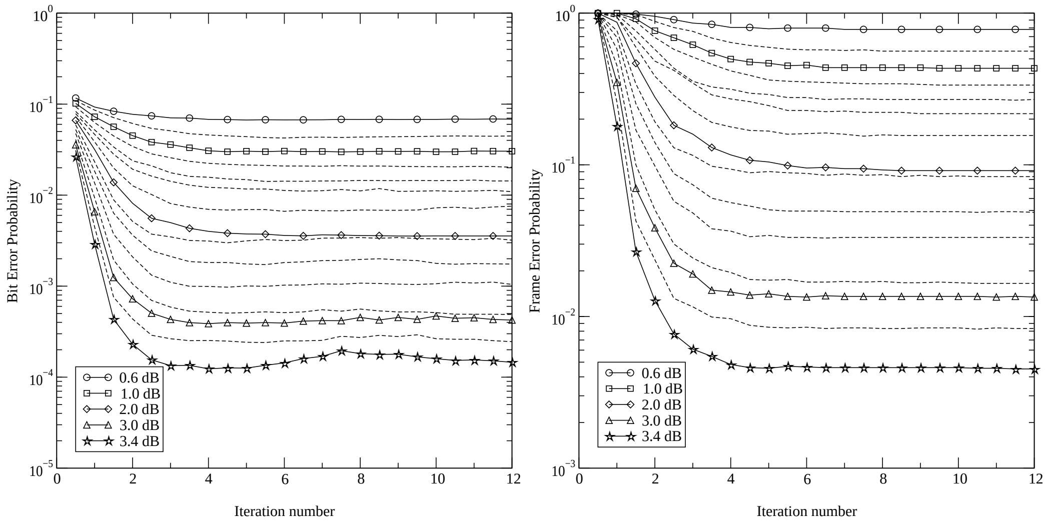


Figure 4.22: Simulation results of the $\mathcal{C}^{P1}$ code : BER (left) and FER (right) *vs.* iteration steps number for Eb/N_0 from 0.6 to 3.4 dB with a 0.2 dB step.

The performance of $\mathcal{C}^{GLD}$ are presented in Figures 4.24. The convergence speed is slightly slower, as the performance still slowly increases even after 8 iteration steps. However, at $E_b/N_0 = 3.4$ dB, the BER is already below 1.10^{-5} after the fifth step. Perfect iterative decoding performance (see Section 4.4.1) are presented in the same figure.

Figures 4.25, 4.26 and 4.27 present the results of $\mathcal{C}^{P1}$, $\mathcal{C}^{P2}$ and $\mathcal{C}^{GLD}$ in a more classical form : error rates *vs.* E_b/N_0. The ML upper bounds are also plotted for $\mathcal{C}^{GLD}$. The fact that the simulation results are quite better than these bounds is explained by the choice of a particular random interleaver better than the average interleaver. Such a degree of freedom (to choose, or construct, an interleaver that lead to a *good* minimum distance and that have good iterative decoding properties) does not exist for product codes : their

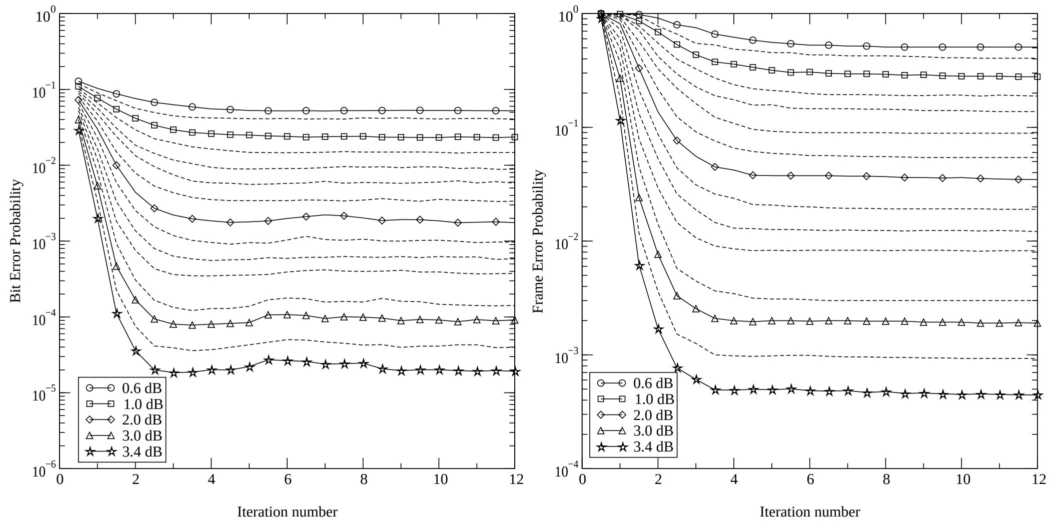


Figure 4.23: Simulation results of the $\mathcal{C}^{P2}$ code : BER (left) and FER (right) *vs.* iteration steps number for Eb/N_0 from 0.6 to 3.4 dB with a 0.2 dB step.

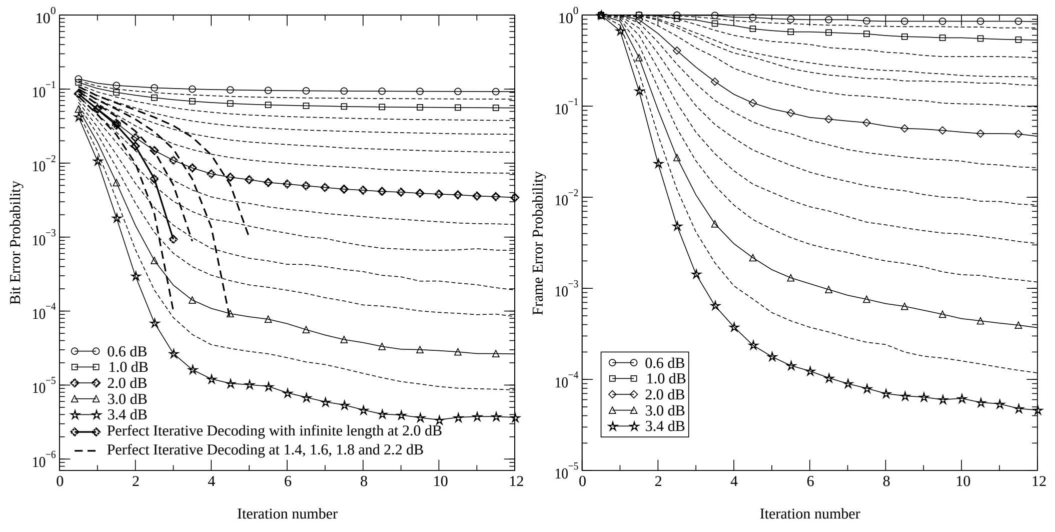


Figure 4.24: Simulation results of the $\mathcal{C}^{GLD}$ code : BER (left) and FER (right) *vs.* iteration step number for Eb/N_0 from 0.6 to 3.4 dB with 0.2 dB step.

graph is completely connected (see Figure 4.4), *i.e.* each constituent code shares one bit with each code of the opposite part. Hence any interleaver in a product code will only perform a permutation over the coded bits (the edges of the graph) but will not influence the iterative decoding performance. Here comes to the light one advantage of the GLD codes over the product codes : the interleaver can be optimized in two directions, namely the highest minimum Hamming distance, and the suitability to iterative decoding. This last subject is examined in the next chapter.

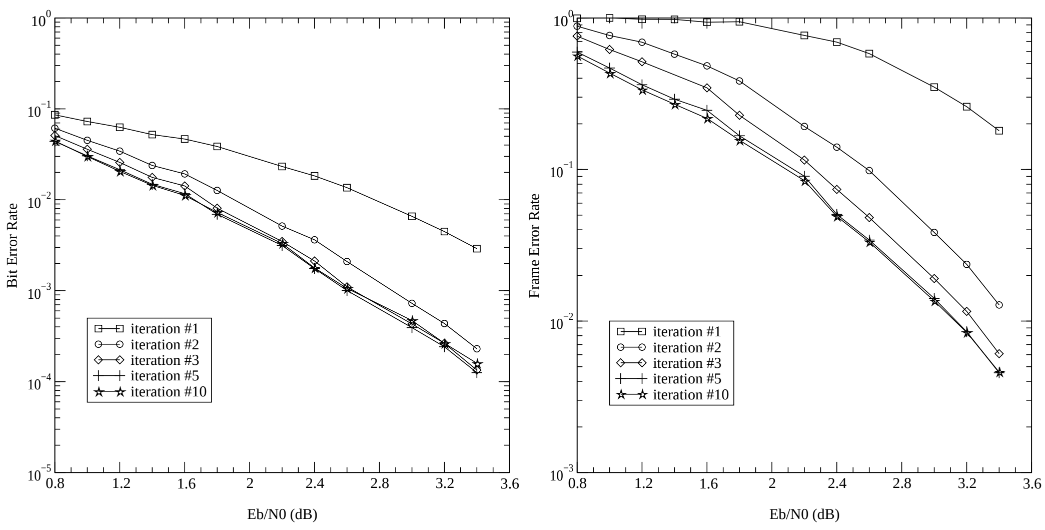


Figure 4.25: Simulation results of the $\mathcal{C}^{P1}$ code : BER (left) and FER (right) *vs.* E_b/N_0 for decoding iteration steps number $1, 2, 3, 5, 10$.

The last Figure (4.28) summarizes the comparison between the three competing codes. At BER and FER of practical interest, the GLD code outshines the two product codes. With the same number of iteration steps (5), it leads to a performance (in BER and FER) that is better by 0.2 dB than that of $\mathcal{C}^{P2}$, which has a decoding complexity three times higher. This gain reaches 0.4 dB if we consider the performance of the GLD code after 10 decoding iterations (decoding of this code is still 1.5 times simpler than the one of $\mathcal{C}^{P2}$).

This simple case study allows us to overline the advantages of GLD codes over block product codes for small length codes. They are less complex to decode and achieve better performance. They have a supplementary freedom degree, namely, the interleaver choice. They have also a higher flexibility : different constituent codes and repetition factors lead to the same length

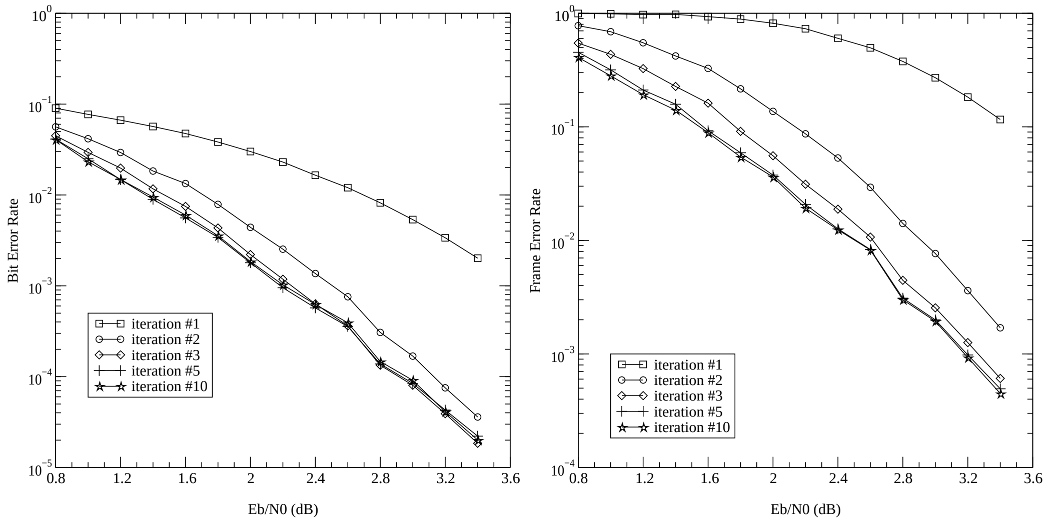


Figure 4.26: Simulation results of the $\mathcal{C}^{P2}$ code : BER (left) and FER (right) *vs.* E_b/N_0 for decoding iteration steps number $1, 2, 3, 5, 10$.

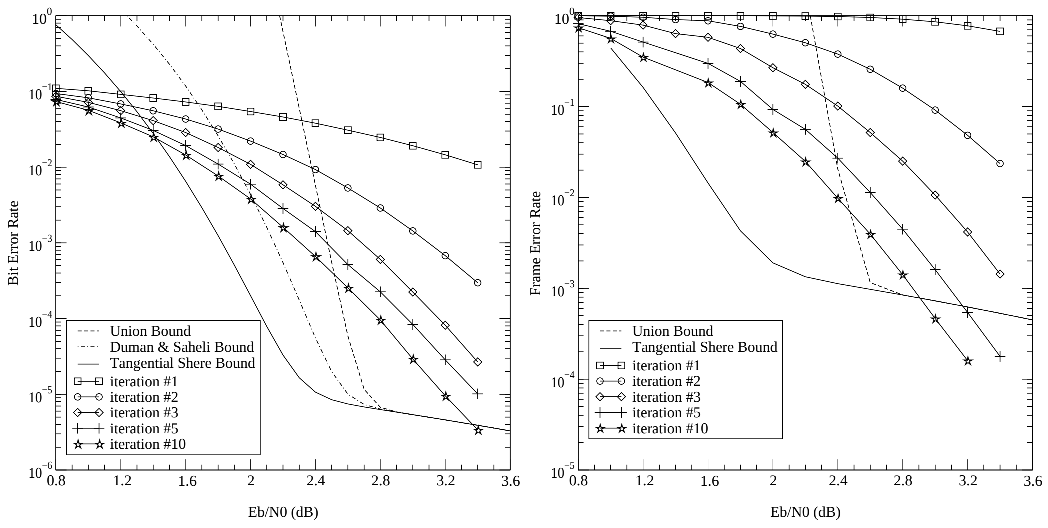


Figure 4.27: Simulation results of the $\mathcal{C}^{GLD}$ code : BER (left) and FER (right) *vs.* E_b/N_0 for decoding iteration steps number $1, 2, 3, 5, 10$.

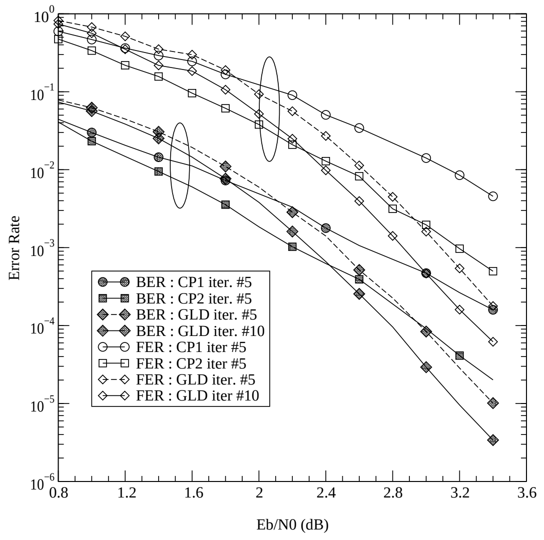


Figure 4.28: Compared performance of $\mathcal{C}^{P1}$, $\mathcal{C}^{P2}$ and $\mathcal{C}^{GLD}$.

and dimension, whereas these parameters almost completely fix the product code. Furthermore iterative decoding works better for GLD code for two main reasons :
– the interleaver acts on all the bits,
– choosing an appropriate interleaver, the number of independent decoding iteration steps is higher.

The major drawback of GLD code arises when the code length becomes too large. We did not find yet any efficient encoding procedure based only on the constituent codes that does not need the construction of the generator matrix of the code. This leads to high memory requirement and numerous binary operations in the encoder part. One track to follow in order to alleviate its task could be to consider the graphical representation of the GLD code.

Chapter 5

GLD interleavers based on graphs*

Nul n'est besoin d'espérer pour entreprendre,
Ni de réussir pour persévérer.
Guillaume d'Orange, prince de Nassau

Our main aim in this chapter is to discuss different methods for designing interleavers for 2-level GLD codes [1] with constituent code length n, leading to a *compact* bipartite biregular graph representation with prescribed girth g. Since choosing an interleaver is equivalent to assigning the edges of the *compact* graph depicted in Figure 4.3, we refer to the girth of an interleaver suitable for GLD codes as the girth of its corresponding graph.

The first section is devoted to general relationships between girth and the parameters of codes based on graphs and known results. The second section presents an algorithmic design of random interleavers that guarantees a girth $g \geq 4$. The third section focuses on two algebraic designs of interleavers with girth $g = 6$. We also describe a method for designing interleavers with girth $g = 8$. Simulation results using these interleavers are presented in the last section, and the performance improvement gained by using higher girth interleavers is discussed.

*Parts of this chapter have been presented at the Philips DSP'99 conference.
[1]In all this Chapter, "GLD code" should be read "2-level GLD code".

5.1 Influence of graph and code parameters on iterative decoding

Let us denote by G a connected n-regular graph with V vertices and N edges. The vertices represent the constituent codes all assumed to be identical to a same code $\mathcal{C}_0$, and the edges the bits. Hence, G represents a compound code $\mathcal{C}$ whose V constituent codes have length n and where each bit belongs to $J = 2$ constituent codes. We do not yet require that the graph be bipartite. This graph is depicted in Figure 5.1.

We denote by g the girth of the graph G, defined as the length of its shortest cycle, by d the diameter of the graph G, defined as the greatest distance between any two vertices, by d_{Hmin} the minimum Hamming distance of the constituent codes and by D_{Hmin} the minimum Hamming distance of $\mathcal{C}$. Let us just recall that every bipartite graph has an even girth (see *e.g.* [24]).

Assuming a general decoding of the code $\mathcal{C}$ involving iterative decodings of the constituent codes and APP propagation (see Section 4.4.2),

- $\lfloor g/2 \rfloor$ corresponds to the *maximum number of independent decoding iteration steps*
- the diameter corresponds to the *minimum number of decoding iteration steps such that all the observations are passed to every bit APP.*

The girth and diameter of any graph are linked by the following basic relation [34] :

$$g \leq 2d + 1 \tag{5.1}$$

Graphs satisfying the equality would lead to codes which could be decoded using a *perfect* iterative decoding : it would not suffer from cycles and dependencies. Indeed, all the observations will transit to all the bits in at most d iteration steps, following a single path in the graph. However, this equality is rarely met[2].

[2]Furthermore, we are interested in bipartite graphs, which have an even girth, and hence can not meet the equality.

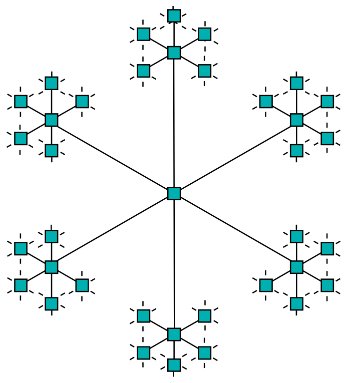


Figure 5.1: Graph G with $n = 6$. The constituent code vertices are represented by squares, and the coded bits lie on the edges.

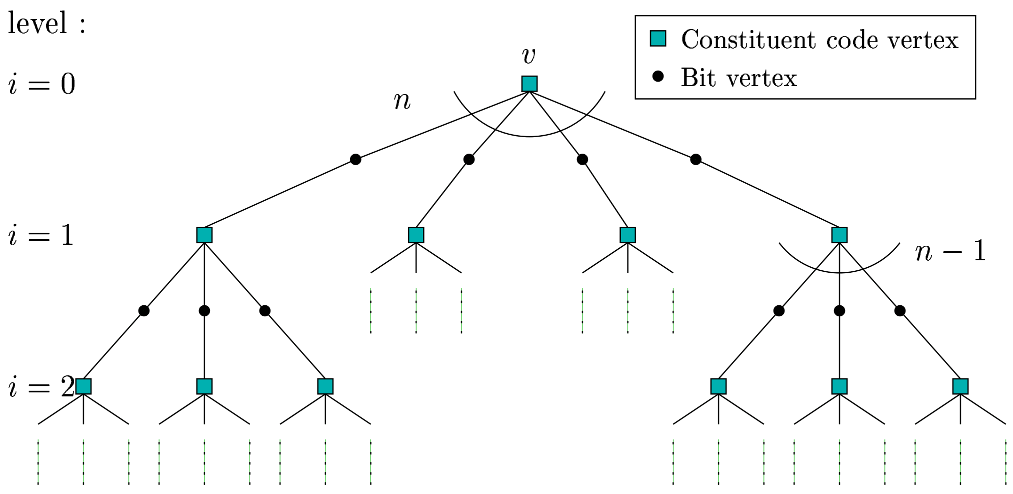


Figure 5.2: Dependency graph of a compound code with graph G, $n = 4$.

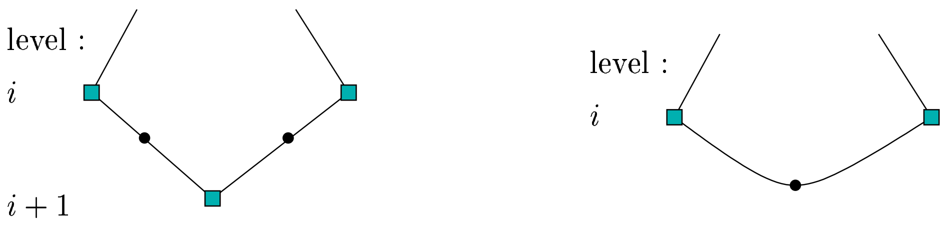


Figure 5.3: Girth configurations. i is the highest level without cycle in the dependency graph. The left figure represents an even girth configuration, and the right figure an odd one.

5.1.1 Girth, code length and minimum distance

Figure 5.2 represents the graph G where we selected an arbitrary vertex v, and where we ordered the other vertices by levels : each vertex at level i is at a distance i from v. This *dependency* graph has no cycles at levels $i < \lfloor g/2 \rfloor$. The number $V(i)$ of vertices per level and $N(i)$, the number of edges (or bits) between level $i-1$ and i are easily counted : $V(0) = 1$ and

$$V(i) = N(i) = n\,(n-1)^{i-1}, \qquad 1 \leq i < \lfloor g/2 \rfloor \tag{5.2}$$

Two cases have to be considered, depending on g being even or odd.

$\underline{g \textbf{ even :}}$ the first cycles in the dependency graph occur at level $i = g/2$ (see Figure 5.3, left). Hence, we have the following inequalities :

$$V_e(g/2) \geq \frac{n\,(n-1)^{g/2-1}}{n} = (n-1)^{g/2-1} \tag{5.3}$$

$$N_e(g/2) \geq n\,(n-1)^{g/2-1} \tag{5.4}$$

and for $g > 4$ the total number of constituent codes and bits in the graph is lower bounded by :

$$V_e \geq \sum_{i=0}^{g/2} V_e(i) \geq 1 + n\frac{(n-1)^{g/2-1} - 1}{n-2} + (n-1)^{g/2-1} \hat{=} V(n,g) \tag{5.5}$$

$$N_e \geq \sum_{i=1}^{g/2} N_e(i) \geq n\frac{(n-1)^{g/2} - 1}{n-2} \tag{5.6}$$

$\underline{g \textbf{ odd :}}$ the first cycles in the dependency graph occur below the level $i = (g-1)/2$, (see Figure 5.3, right). Hence, the general formula (5.2) stands for $V_o\,((g-1)/2)$, and we have :

$$N_o\,((g+1)/2) \geq \frac{n\,(n-1)^{(g-1)/2}}{2} \tag{5.7}$$

We obtain this inequality by considering that all the edges (and consequently the bits) beyond level $i = (g-1)/2$ connect two distinct vertices of this level. Thus, for $g > 3$ we have :

$$V_o \geq \sum_{i=0}^{(g-1)/2} V(i) \geq 1 + n\frac{(n-1)^{(g-1)/2} - 1}{n-2} \hat{=} V(n,g) \tag{5.8}$$

$$N_o \geq \sum_{i=1}^{(g+1)/2} N(i) \geq \quad n \quad \frac{n\,(n-1)^{(g-1)/2}-2}{2\,(n-2)} \tag{5.9}$$

These results have first been published by Tutte [100]. He proved that if an n-regular graph of girth g matches the equality $V = V(n, g)$, its diameter d verifies (5.1) with equality if g is even, and it is bipartite and have a diameter[3] $d = g/2$ is g is even. The next section presents some graphs reaching or approaching these bounds.

The dependency graph (Figure 5.2) can also be used to analyze the link between the girth and the minimum Hamming distance of the compound code. Let us assume that two codewords of $\mathcal{C}$ differ in a bit b represented by an edge connected to a particular constituent code vertex v, and let us construct the dependency graph whose root vertex is v. The two codewords must differ in at least d_{Hmin} edges (or bits) at the first level in the dependency graph. The minimum distance of the constituent codes at level 1 ensures us that descending to the next level, the codewords differ along at least $d_{Hmin}(d_{Hmin}-1)$ edges among the $n(n-1)$ edges, and so on. Hence, the method used to enumerate the edges (leading to (5.6) and(5.9)) is applicable to enumerate the different edges between two codewords by replacing n with d_{Hmin}. Lower bounds on the minimum Hamming distance of $\mathcal{C}$ can be derived :

$$D_{Hmin} \geq \qquad d_{Hmin}\frac{(d_{Hmin}-1)^{g/2}-1}{d_{Hmin}-2} \qquad g \text{ even} \tag{5.10}$$

$$D_{Hmin} \geq \; d_{Hmin} \quad \frac{d_{Hmin}(d_{Hmin}-1)^{(g-1)/2}-2}{2(d_{Hmin}-2)} \qquad g \text{ odd} \tag{5.11}$$

These bounds have been expressed in a more general case, when the bit vertices are connected to more than two constituent code vertices (*i.e.* when the code has more than $J = 2$ levels) by Tanner [97]. They are relatively weak as compared with the ones derived using the weight enumerator function of the constituent codes (Figure 4.11). Indeed, N and D_{Hmin} are both expressed as exponential functions of the girth, and hence do not provide any asymptotical information on the normalized Hamming distance.

Codes based on graphs (Tanner graphs, see Figure 4.2) have been investigated by several authors. Recently, Sipser and Spielman [95] used expander graphs and their algebraic properties [2] to explicitly construct a class of com-

[3]this is the best distance value a graph with even girth can achieve.

pound codes that are asymptotically good, and provided simple sequential and parallel hard decoding algorithms.

From the above considerations, compound codes based on graphs that would achieve the best performance with our iterative soft-input soft-output decoding algorithm have to be constructed from a graph of degree n which

1. has a large girth, so as to provide both a large minimum Hamming distance and a large number of independent decoding iteration steps,
2. has a small diameter, to ensure quick iterative decoding,
3. has a small number of vertices, thus yielding a short code,
4. is bipartite, to easily schedule the iterative process.

These constraints are dependent. A graph verifying the third condition (meaning that[4] $V = V(n, g)$) will also verify the first and second, and also the fourth if g is even. Such a graph is called a (n, g)-graph.

However, (n, g)-graphs are few and when an (n, g)-graph does not exist, the n-regular graph of girth g with the fewest possible number of vertices is called a (n, g)-cage. A lot of work to find the (n, g)-graphs and (n, g)-cages have been carried through in the 60's and 70's, as it was of great interest for the telephone networks. A brief survey of these graphs is presented in the next section.

Furthermore, our approach to maximize the girth may not be the valid one. It may be more clever to look at the whole cycles distribution rather than only at the shortest cycle (The same argument applied to the distance distribution is used by Battail [10] to illuminate the astonishing performance of turbo codes). The cycle distribution of our constructed interleavers is computed in Section 5.5.

[4]this equality is called the *Moore bound* by some authors for all g, even though Moore graphs have an odd girth

5.1.2 A brief survey of known (n, g)-graphs and (n, g)-cages

For $\underline{g = 3}$ and $\underline{g = 4}$, there exist $(n, 3)$- and $(n, 4)$-graphs for any degree n. The $(n, 3)$-graph is the complete graph with $V(n, 3) = n+1$ vertices, and the $(n, 4)$-graph is the complete bipartite graph with $V(n, 4) = 2n$ vertices (also called the generalized digon). The latter case corresponds to the conventional product code. Figure 5.4 shows simple examples of these cases.

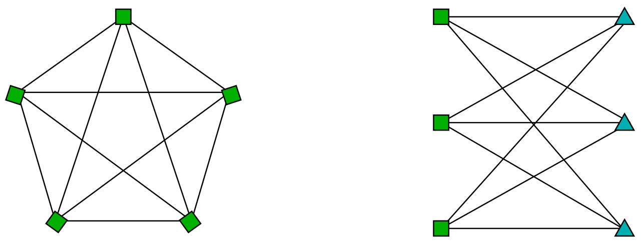


Figure 5.4: Classical $(4, 3)$-graph (left) and (3,4)-graph (right).

The $\underline{g = 5}$ case has been first studied by Hoffman and Singleton [56]. They proved that $(n, 5)$-graphs exist only for $n = 2, 3, 7$ and possibly $n = 57$. The existence of the latter is enigmatic and not yet proved, to our knowledge. The $(2, 5)$ graph is the David star, the $(3, 5)$-graph is the Petersen graph and the $(7, 5)$-graph is called the Hoffman-Singleton graph.

The $\underline{g = 6, 8, 12}$ (n, g)-graphs are known to exist only when $n - 1$ is a prime power [19]. They are respectively generalized triangles, quadrangles and hexagons of order $n - 1$. $(n, 6)$-graphs are the point/line graphs of projective planes, whose constructions are explained in Section 5.3. The $g = 8$ case is described in Section 5.4.

There are no other (n, g)-graphs, as stated by Biggs [24]. The knowledge of (n, g)-cages is also limited. For small values of degrees and girths, results are known and accurate (because the graphs are as small as to enable exhaustive search). For example the $(7, 6)$-cage has 90 vertices, as $V(7, 6) = 86$. It is not a bipartite graph. In comparison, Section 5.3 presents the construction of a bipartite graph of degree $n = 7$ and girth $g = 6$ that has 96 vertices. Cages hunters like Royle [91] maintain state-of-the-art tables of known cages.

5.2 Random interleavers with girth $g \geq 4$

The first attempt to introduce graph-theoretical considerations in the construction of random interleavers suitable for GLD codes uses the same approach as Gallager ([47] Appendix C). He presented a construction procedure that guarantees no *closed path* of prescribed length (*i.e.* the equivalent of cycles) in the parity check matrix of LDPC codes.

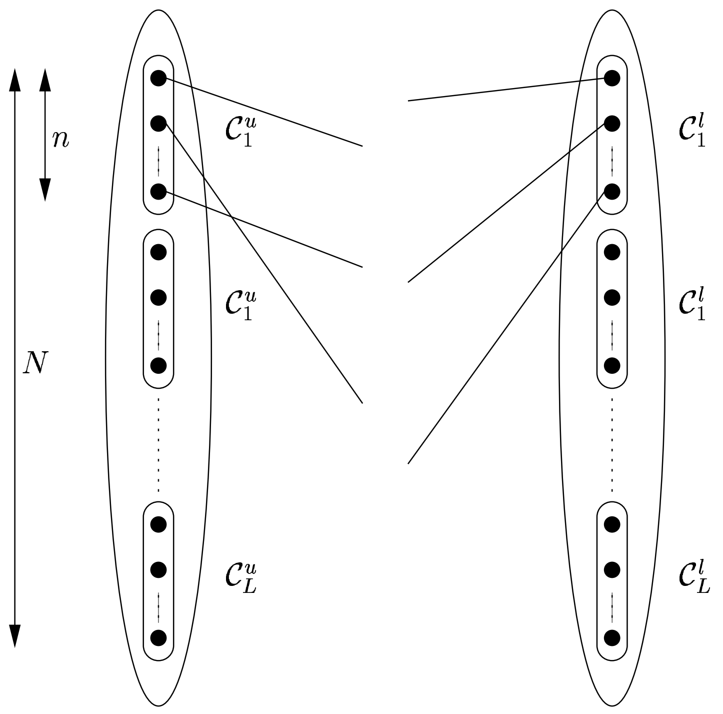


Figure 5.5: n-clumped random matching of size N

We first construct a random matching over N vertices. Any pseudo-random design method of classical interleaver can be used for this purpose, especially those developed for turbo codes. The N vertices in each part of the random matching are grouped in L clumps of n bits.

For each clump C_i^u, $i = 1, \cdots, L$ in the left part, we look at its n edges. We denote by $\mathcal{I}(C_i^u)$ the set of right vertices connected to C_i^u. If two (or more) elements of $\mathcal{I}(C_i^u)$ belong to the same right clump C_j^l, we permute the edges connected to these elements with others belonging to $\mathcal{I}(C_{i'}^u)$, $i' > i$, such that $\mathcal{I}(C_i^u)$ no longer includes two vertices belonging to the same right clump.

If (especially for the last clump C_l^u) we can not find such a permutation, we pick another random matching and try again.

In the third step, we *compact* the random matching, that is, we consider each clump as a single vertex. This gives birth to a random regular bipartite graph of degree n that has no cycles of length two. Its girth is thus greater than, or equal to, 4.

A special case of this construction is the *Generalized Row-Column Interleaver.* Let n be the length of the constituent code $\mathcal{C}_0$ and $N = Ln$ the length of the compound code $\mathcal{C}$. We construct a matching over N vertices indexed (in the two parts) from $i = 0$ to $i = N - 1$ thanks to the incidence relation :

$$\mathcal{I}(i) = (i \times n) \text{ modulo } N + \lfloor \frac{i \times n}{N} \rfloor \tag{5.12}$$

This means that the vertex i in the left part is connected to the vertex $\mathcal{I}(i)$ in the right part.

This matching is compacted to a bipartite regular graph of degree n that has a girth of 4. If $N = n^2$, the graph is the completely connected bipartite graph, that is the graphical representation of a block product code with the trivial row-column interleaver. It is the $(n, 4)$-graph with $2N$ vertices presented in the previous section.

The matching and the bipartite graph of a generalized row-column interleaver with parameters $N = 12$ and $n = 3$ are depicted in Figure 5.6.

5.3 Interleavers with girth 6

Our aim in this section is to build bipartite biregular graphs with a girth $g = 6$. An obvious construction would be to extend the edges permutations of a random matching previously described to eliminate cycles of length 4. However, this is a complex task from the algorithmic point of view, and an inelegant and brute force approach. We choose instead to investigate graphs with strong algebraic structure.

The result on $(n, 6)$-graphs stated in Section 5.1.1 is the basis of our first

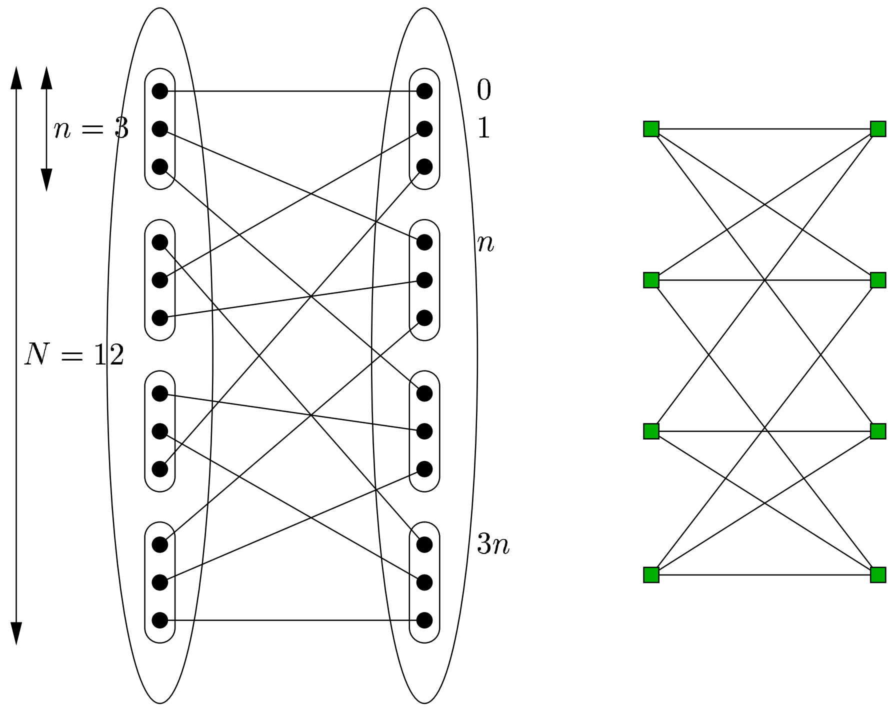


Figure 5.6: Matching and bipartite graph of the generalized row-column interleaver with $N = 12$ and $n = 3$.

construction. However, $(n, 6)$-graphs exist only if $n - 1$ is a prime power. Our second algebraic construction leads to graphs of degree n where n is a prime power.

5.3.1 Interleavers from projective geometry $\mathrm{PG}(2, q)$

The use of the point/line incidence graph of the projective plane $\mathrm{PG}(2, q)$ (which is the $(q+1, 6)$-graph) as the underlying structure of compound codes was first introduced by Tanner [97].

A projective geometry consists of a finite set $\mathcal{P}$ of *points* together with a finite set $\mathcal{L}$ of subsets of $\mathcal{P}$ called *lines*. We denote by p a point of $\mathcal{P}$ and by l a line of $\mathcal{L}$. We say that p lies on l and l passes through p if $p \in l$. Some axioms have to be verified :

1. there is a single line passing through any two distinct points
2. every line contains at least three points
3. if two distinct lines l and m have a common point p and letting $q, r \in l$ and $s, t \in m$ then the lines defined by (q, t) and (r, s) also have a common point
4. for any point p, there is at least two lines not containing p
5. for any line l, there are at least two points not lying on l.

The interested reader can refer to Dembowski [33] for further details and properties of the projective geometries.

The projective geometry $\mathrm{PG}(m, q)$ is obtained from a finite field $\mathrm{GF}(q)$. The points of $\mathcal{P}$ are taken to be the equivalence classes of the nonzero $m+1$-tuples $(a_0, a_1, \cdots, a_m)$ of $\mathrm{GF}(q)^{m+1}$ for the equivalence relation of proportionality $\sim$:

$$\begin{aligned} & (a_0, a_1, \cdots, a_m) \sim (b_0, b_1, \cdots, b_m) \\ \Longleftrightarrow \quad & \exists \lambda \in \mathrm{GF}(q)^* : (a_0, a_1, \cdots, a_m) = (\lambda b_0, \lambda b_1, \cdots, \lambda b_m) \end{aligned} \tag{5.13}$$

The five axioms hold for this construction. The dimension of $\mathrm{PG}(m, q)$ is clearly m. Since there are q^{m+1} nonzero $(m+1)$-tuples and since any equivalence class has $q-1$ elements, the number of points in $\mathrm{PG}(m, q)$ is :

$$|\mathcal{P}| = \frac{q^{m+1}}{q-1} \tag{5.14}$$

The line $l = (pq)$ through two distinct points $p = (a_0, a_1, \cdots, a_m)$ and $q = (b_0, b_1, \cdots, b_m)$ consists of the points :

$$(\lambda a_0 + \mu b_0, \lambda a_1 + \mu b_1, \cdots, \lambda a_m + \mu b_m) \tag{5.15}$$

where λ and μ are elements of $\mathrm{GF}(q)$ which are not both zero. Since there are $q^2 - 1$ possible choices for (λ, μ) and since any point of l appears $q-1$ times in (5.15), the number of points per line is $q+1$.

Without entering into the details, it can be shown that, by introducing the definition of *subspaces* of $\mathrm{PG}(m, q)$, the lines are subspaces of dimension 1, and are $\mathrm{PG}(1, q)$. Hence, the number of $PG(1, q)$ contained in a $PG(m, q)$

is the number of lines in the geometry and combinatorial considerations show it is equal to :

$$|\mathcal{L}| = \frac{(q^{m+1}-1)\,(q^m-1)}{(q^2-1)\,(q-1)} \tag{5.16}$$

We deduce from (5.14)and (5.16) that the number of lines on which a point lies is $(q^m-1)/(q-1)$.

The point/line incidence graph of $\mathrm{PG}(m,q)$ is defined as the bipartite graph whose first part contains the points and the second one contains the lines of the geometry. An edge connects a point vertex to a line vertex if the point lies on the line. The degree of the point vertices is hence $(q^m-1)/(q-1)$ and the degree of the line vertices is $q+1$. This graph is depicted in Figure 5.7.

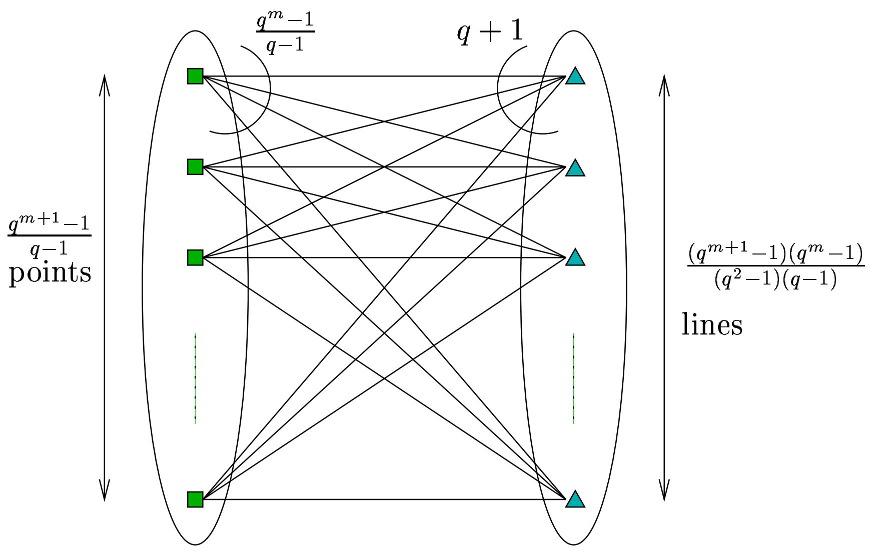

$\frac{q^m-1}{q-1}$
$q+1$
$\frac{q^{m+1}-1}{q-1}$ points
$\frac{(q^{m+1}-1)(q^m-1)}{(q^2-1)(q-1)}$ lines

Figure 5.7: $\mathrm{PG}(m,q)$ bipartite graph representation

This graph does not present any cycle of length 2 by construction, and axiom 1 guarantee that there is no cycle of length 4 (if such a cycle would exit, two distinct points would lie on two distinct lines).

However, this graph has not the form of the *compact* graph of a GLD code, since it is not biregular, and the numbers of vertices in the two parts are not equal.

If we take $m=3$, $\mathrm{PG}(3,q)$ is called a generalized triangle and has the following properties : the number of lines is equal to the number of points :

$$|\mathcal{P}| = |\mathcal{L}| = \frac{q^3 - 1}{q - 1} = q^2 + q + 1 \tag{5.17}$$

and the number of points per line is equal to the number of lines on which lies each point, that is $q+1$. As previously stated, the point/line incidence graph is the $(q+1, 6)$-graph. It is depicted for $m = 2$ in Figure 5.8.

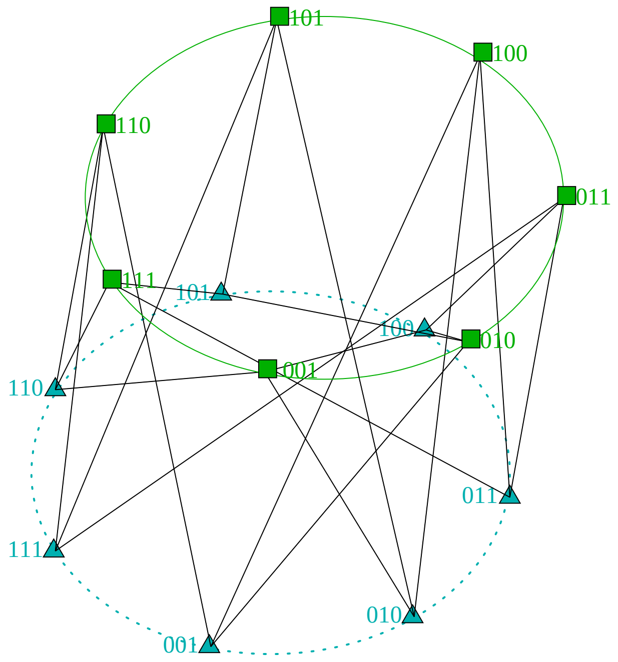


Figure 5.8: PG(2, 2) graphical representation. Square vertices on the upper circle represent the points of $\mathcal{P}$, triangle vertices on the lower circle represent the lines of $\mathcal{L}$. An edge between a point vertex and a line vertex means that the point lies on the line.

The point vertices are labelled by their coordinates. As any line subspace PG$(1, q)$ is now a hyperplane of PG$(2, q)$, it consists of those points $(a_0, a_1, \cdots, a_m)$ which satisfy the following linear equation :

$$\lambda_0 a_0 + \lambda_1 a_1 + \lambda_2 a_2 = 0 \tag{5.18}$$

where the λ's are elements of GF(q). Hence, the points and the line subspaces are dual, and lines can be labelled by their coordinates $(\lambda_0, \lambda_1, \lambda_2)$. This allows us to efficiently construct these geometries, even for large values of q.

Hence, from any projective geometry PG$(2, q)$, and any constituent code of length $n = q + 1$, it is possible to construct the GLD code of total length $N = (q^2 + q + 1)(q + 1) = n^3 - n^2 + n$, whose *compact* graph has a girth of 6. This code is optimal as regards its code length.

5.3.2 Interleavers from Cayley graphs

The graphs presented in the previous section exist only if their degree is a prime power plus one. We present a second construction of biregular bipartite interleaver where the degree is a prime power.

Schellwat [94] presented Cayley graphs built from dihedral groups. His aim was to construct graphs of degree n where each vertex has a large number of distinct neighbors. The last property is closely related to the expansion factor c of the graph. The situation is unusual : most random graphs of *fixed* degree n are good expanders, but testing whether a particular one has a good expansion factor is a complex task. However, Alon [2] highlighted the link between the expansion factor of a graph and its spectrum[5], more precisely the value of its second highest eigenvalue μ. Roughly speaking, the smaller μ, the higher c. However, when the length N goes to infinity, the lowest value μ of any family of n-regular graphs is lower bounded by a function of n . Some graph families (called *Ramanujan* graphs, constructed in [75] and [71]) exhibit the remarkable property that μ is upper bounded. Moreover, these graphs have also a girth that is lower bounded by :

$$g(N) \geq (4/3 + o(1)) \log_{n-1}(N) \tag{5.19}$$

which is astonishingly close to the asymptotic upper bound version of (5.5) and (5.8) :

$$g(N) \leq (2 + o(1)) \log_{n-1}(N) \tag{5.20}$$

Relaxing the fixed degree condition, Schellwatt constructed a graph family with better spectrum properties than Ramanujan graphs. However, he did not investigate the girth of these graphs. As we will show, they have a girth of 6.

Since many readers of the digital communications community (including the author...) are not too familiar with the graph theory and discrete

[5]the spectrum of the graph is the set of eigenvalues of its adjacency matrix.

mathematics, we do not use nor dihedral groups neither Cayley graphs properties to explain this construction. Indeed, it can be presented only with considerations on Galois fields.

We construct bipartite biregular graphs of degree $n = q$ with q^2-1 vertices in each part, where q is a prime power. It is particularly interesting with $n = 7, 11, 13, 16, \ldots$ because the first presented construction is not applicable, and the "smallest" graphs are either not bipartite or unknown.

Let $q = p^m$, where p is prime, and construct GF(q) as GF$(q) =$ GF$(p)[x]/p(x)$ where $p(x)$ is a prime polynomial of degree m over GF(p). Let $Q = q^2$, and construct GF$(Q) =$ GF$(q)[x]/q(x)$ thanks to $q(x)$, a primitive polynomial of degree 2 over GF(q). $q(x)$ can be written as $q(x) = x^2 + r.x + s$, with $(r, s) \in$ GF$(q)^2$. Let α be a prime element of GF(Q). α is then a root of $q(x)$.

We define $\mathcal{S}$ as :

$$\mathcal{S} = \{\alpha + a_i, a_i \in \mathrm{GF}(q), i \in [1, \cdots, q]\} \tag{5.21}$$

We have $|\mathcal{S}| = q$. We also define the mapping f as :

$$f = \left\{ \begin{array}{rcl} \mathrm{GF}(Q) \setminus \{0\} & \longrightarrow & (\mathrm{GF}(Q) \setminus \{0\})^q \\ x & \longmapsto & \mathcal{S}x \end{array} \right. \tag{5.22}$$

f is the mapping that links any non-zero element of GF(Q) to the set of its product with each element of $\mathcal{S}$. If we label each of the $Q - 1$ nodes of the two parts of the graph by an element x of GF$(Q) \setminus \{0\}$, $f(x)$ defines the set of lower nodes connected to the upper node x. Thus, f defines a bipartite graph of degree q on the upper part (because the elements of $\mathcal{S}x$ are all distinct).

Let us prove that all the nodes in the lower part are also connected to q distinct nodes in the upper part. Let $y \in$ GF$(Q) \setminus \{0\}$ represent the label of a lower node. Like any element of GF(Q), y can be projected on the $(1, \alpha)$ basis :

$$y = a_y.\alpha + b_y \tag{5.23}$$

Let us show that for each $a_i \in$ GF(q), there is a single $x_i \in$ GF$(Q) \setminus \{0\}$ such that : $y = (\alpha + a_i).x_i$. As x_i can be expressed as $x_i = a_{x_i}.\alpha + b_{x_i}$, it leads to the system :

$$\left\{ \begin{array}{rcrcl} (a_i - r).a_{x_i} & + & b_{x_i} & = & a_y \\ -s.a_{x_i} & + & a_i.b_{x_i} & = & b_y \end{array} \right. \tag{5.24}$$

whose determinant is $a_i^2 - r.a_i + s = q(-a_i)$. But $-a_i$ belongs to GF(q) and thus can not be a root of $q(x)$. So (5.24) has a single solution which gives x_i as a function of a_i.$\square$

Let us now prove that such a graph has no cycle of length 4. Such a cycle would correspond to Figure 5.9.

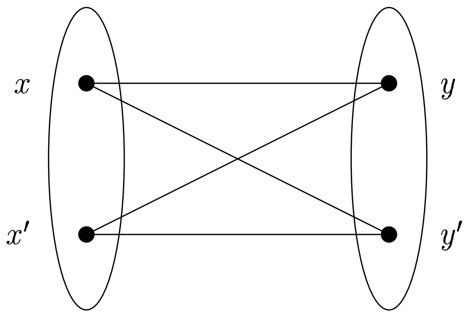


Figure 5.9: Cycle of length 4

There would exist a_1, a_2, b_1, b_2, belonging to GF(q) and all distinct, such that :

$$\begin{array}{c} y{=}(\alpha + a_1).x \\ y{=}(\alpha + b_1).x' \end{array} \quad \text{and} \quad \begin{array}{c} y'{=}(\alpha + a_2).x \\ y'{=}(\alpha + b_2).x' \end{array}$$

This would lead to $(a_1 + b_2 - a_2 - b_1).\alpha + a_1.b_2 - b_1.a_2 = 0$. But α can not satisfy this equation of degree 1 over GF(q). Hence, we would have :

$$\begin{cases} a_1.b_2{=}b_1.a_2 \\ a_1 + b_2{=}b_1 + a_2 \end{cases} \tag{5.25}$$

Two cases may occur. First, if one element among (a_1, a_2, b_1, b_2) is zero, (5.25) can not be verified. Else, we would have :

$$\frac{b_1.a_2}{b_2} + b_2 = b_1 + a_2 \quad \Longrightarrow \quad b_1.\left(\frac{a_2}{b_2} - 1\right) = a_2 - b_2$$

that is $b_1 = b_2$ which is also impossible. Hence, the graph has no cycle of length 4, and its girth is at least 6. $\square$

Using this graph construction, and any constituent code of length $n = q$, we build a GLD code of total length $N = (q^2 - 1)q = n^3 - n$, and girth 6. For $n = 7$, the graph has $N_{codes}(7, 6) = 96$ nodes. In comparison, the $(7, 6)$-cage is not bipartite and has 90 nodes. Recently, Guinand and Lodge [50] presented Tanner graphs also constructed with bipartite biregular graphs. They use connected components of graphs based on $D(m,q)$ [40] (which come from the point/line incidence graph of structures which are in some sense analogues of generalized m-gons) where they remove some lines and points. With the same parameters ($g = 6$, $n = 7$), it leads to a bipartite graph with 98 vertices. Our construction is hence slightly better.

5.4 Interleavers with girth 8

The $(n, 8)$-graphs are also bipartite biregular, when $n-1$ is a prime power. These generalized quadrangles can be used to build GLD codes. We briefly describe here a method to construct *conventional generalized quadrangles* [99] [32] [33].

The first step consists in building the projective geometry $\mathrm{PG}(4,q)$. This task is more difficult than building $\mathrm{PG}(2,q)$ since the point- and line subspaces are no longer dual. (5.18) no longer holds. First, we have to enumerate all the $(q^5-1)/(q-1)$ points, and then to explicitly enumerate each line as a set of points related to (5.15).

As depicted in Figure 5.7, the point/line incidence graph of $\mathrm{PG}(4,q)$ is not biregular. We consider the nonsingular quadric :

$$Q(4,q) \quad : \quad X_0^2 + X_1X_2 + X_3X_4 = 0 \tag{5.26}$$

where X_i is the i-th coordinate of $\mathrm{PG}(4,q)$. The following result holds : the points of the projective geometry whose coordinates satisfy (5.26) together with the lines formed with these points lead to a point/line incidence graph which :

- is bipartite (by definition),
- has $(q^2+1)(q+1)$ points,
- has $(q^2+1)(q+1)$ lines,
- is biregular of degree $q+1$,
- has a girth of $g = 8$.

Hence, this graph, used with constituent codes of length $n = q + 1$ leads to a GLD code

- with $2(q^2 + 1)(q + 1) = 2n(n^2 - 2n + 2) = V(n, 8)$ constituent codes,
- of length $N = n^4 - 2n^3$.

However, since GLD codes are attractive for small lengths, this construction seems less useful : even with the shortest Hamming code ($n = 7$), it leads to a GLD code of length $N = 1715$ which is already quite large.

5.5 Simulation results

To illustrate the algebraic graph constructions we presented, and to validate the benefits we can expect from them, we proceed to two comparisons between GLD codes of same length and constituent codes. We also present an algorithmic method to estimate the number of cycles in a GLD graph, and try to link the simulation results with this cycle analysis.

5.5.1 Performance results

The first comparison involves the $(7, 4, 3)$ Hamming code as constituent code $\mathcal{C}_0$. We build the GLD code based on the interleaver derived from the Cayley graph of parameters $p = 7$, $m = 1$, $q = 7$, $Q = 49$. Hence, the GLD code has a total length $N = 336$ and dimension $K = 51$. It is compared with two GLD codes of same length N and constituent code $\mathcal{C}_0$, but built with a generalized row-column interleaver (girth 4) and a pseudo-random interleaver that guarantees a girth of 4.

The second comparison uses the $(12, 8, 3)$ shortened Hamming code $\mathcal{C}_0'$ as constituent code. The GLD code is constructed thanks to PG$(2, 11)$ such that $n = q+1$. Its total length is $N' = 1596$ and its dimension is $K' = 549$. It is also compared with two GLD codes of girth 4 with respectively row-column and pseudo-random interleavers.

We simulated these codes with a BPSK modulation over an AWGN channel, using the same protocol as in Section 4.5. Figures 5.10 and 5.11 present the BER *vs.* the decoding iterations number for these codes.

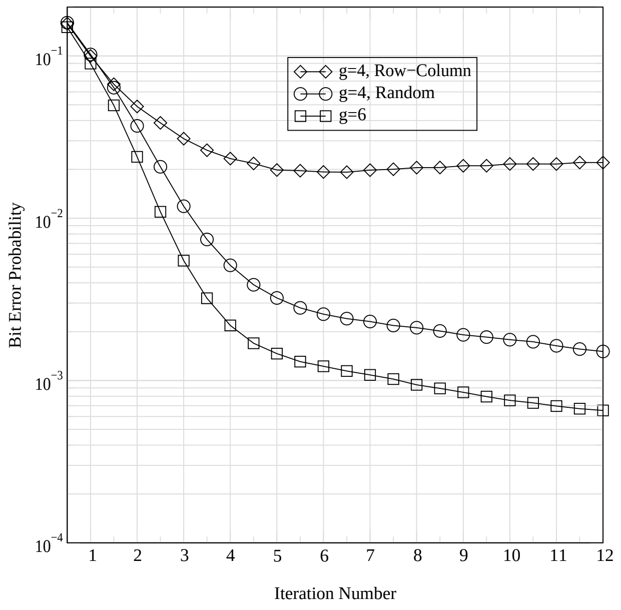


Figure 5.10: GLD codes based on $\mathcal{C}_0 : (n = 7, k = 4, d_{min} = 3)$ for $E_b/N_0 = 3.8dB$

We observe that the row-column interleaver exhibits a very poor performance in both cases, even when compared with another interleaver of same girth. Interleavers with a girth $g = 6$ perform better than conventional pseudo-random interleavers and lead to a gain in terms of iterative convergence between 3 and 6 iteration steps for $\mathcal{C}'_0$ at practical E_b/N_0 values.

5.5.2 Estimation of the cycles distribution

Our aim in this section is to enumerate the cycles[6] that are present in any interleaver suitable for GLD codes, and more specifically to estimate their length distribution. To achieve this task, we construct the dependency graphs for all *bit vertices* of the graphical representation (see Figure 4.2) of a GLD code.

[6]we want to count the *simple* cycles : each vertex appears *at most* once in a cycle.

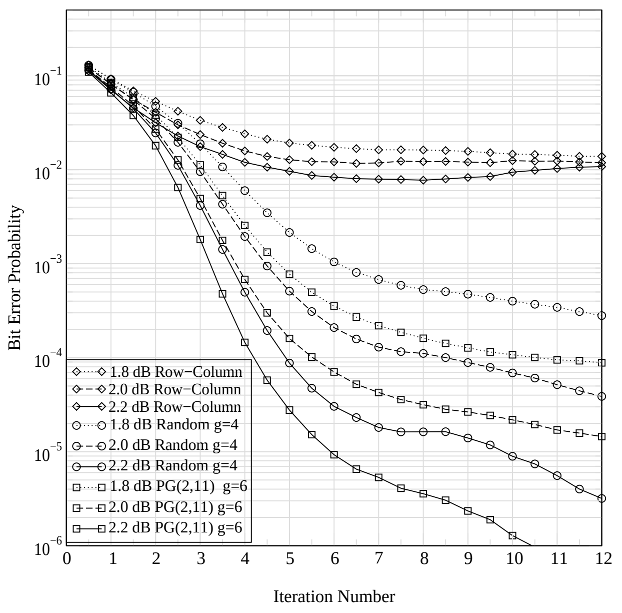


Figure 5.11: GLD codes based on $\mathcal{C}'_0 : (n = 12, k = 8, d_{min} = 3)$ for different E_b/N_0 values

The dependency graph of a bit vertex c_i is constructed similarly to the dependency graph of a code vertex (Figure 5.2), but starting from the vertex c_i at the top level. Each vertex (bits and codes) is plotted at most once in the graph.

For *all* of the N dependency graphs, we count the cycles that include the top-level bit node and their length. To do this, we look at each bit vertex that is a “leaf” (*i.e.* no constituent code vertex is connected under it). Let j be the level where it appears. If there are two *completely* distinct paths that connect this “leaf” bit vertex to the top-level bit vertex, we count a cycle of length $l = 2j$.

After this step, we have counted k times each cycle that involves k bit vertices (*i.e.* of length k). Even if this process is theoretically simple, it becomes tricky to implement. Let us take two simple examples that highlight the difficulties. The two codes have $N = 9$ bits and 6 constituent codes of length 3. The first one is built with a row-column interleaver and the second one with a random interleaver. They are presented in Figure 5.12.

All the dependency graphs of the first code are similar, and one of them (for the bit vertex 1) is depicted in Figure 5.13. It has 4 cycles of length 4,

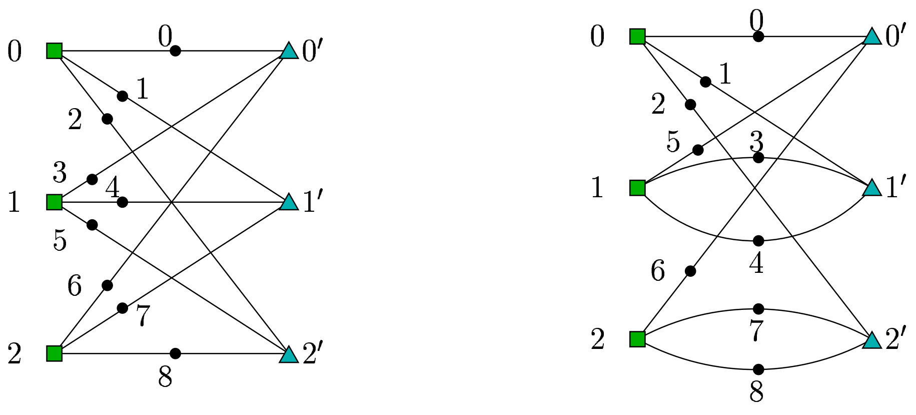


Figure 5.12: $n = 3$, $N = 9$ row-column (left) and random (right) interleavers.

namely $(1, 0, 3, 4, 1)$, $(1, 2, 5, 4, 1)$, $(1, 0, 6, 7, 1)$ and $(1, 2, 8, 7, 1)$. Since each of them includes the top-level bit vertex, they are all counted. Hence, the total number of counted cycles has the simple distribution : $9 \times 4 = 36$ cycles of length 4. Since each cycle is counted 4 times (its length), the total number of cycles in the graph is : $36/4 = 9$ cycles of length 4.

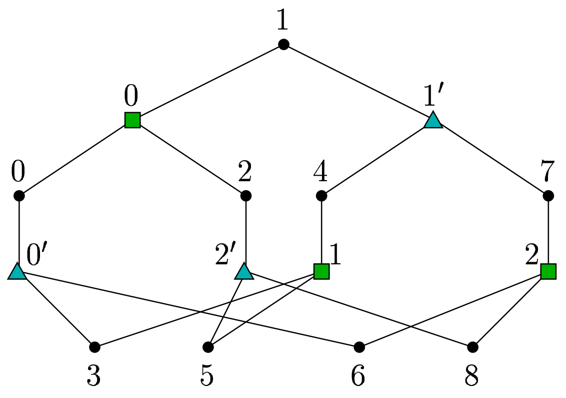


Figure 5.13: Dependency graph of the bit node 1 for the $n = 3$, $N = 9$ row-column GLD code.

One of the dependency graphs (for the bit vertex 0) of the second code is similar to the one depicted in Figure 5.13, but two other types of graphs also exist. They are presented in Figure 5.14. In the dependency graph of the bit vertex 1, cycles involving the constituent code 2 have not to be taken into account, since there are no leaves at the last level. On the other side, there are two cycles that include the bit vertex 5 : $(1, 0, 5, 3, 1)$ and $(1, 0, 5, 4, 1)$.

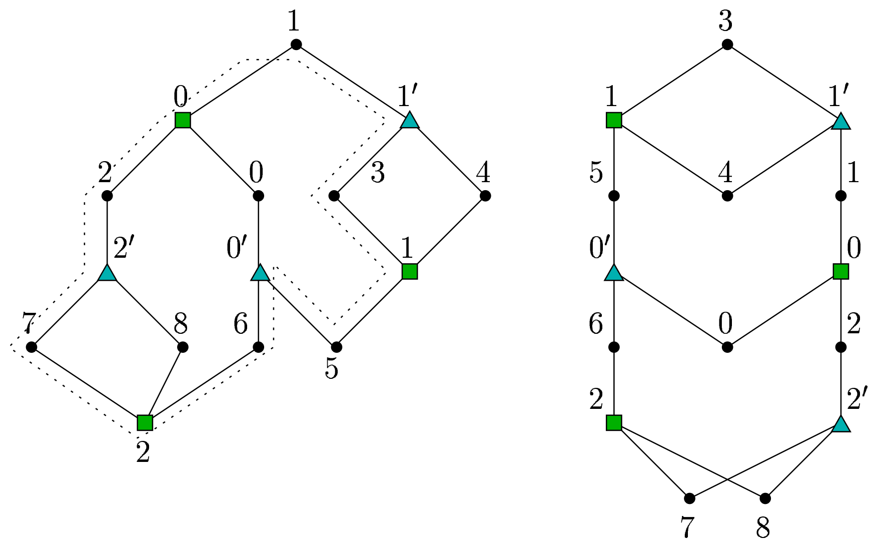


Figure 5.14: Dependency graphs of the bit vertices 1 (left) and 3 (right) for the $n = 3$, $N = 9$ Random GLD code.

These are the only cycles to be counted for the dependency graph of the bit vertex 1. Hence, when a cycle *forks* one or several times, it has to be properly counted.

The dependency graph of the bit vertex 3 presents one cycle of length 2 : $(3, 4, 3)$, one cycle of length 4 : $(3, 5, 0, 1, 3)$ and two cycles of length 6 : $(3, 5, 6, 7, 2, 1, 3)$ and $(3, 5, 6, 8, 2, 1, 3)$.

We can summarize the cycle count for the second code as in Table 5.1 (please, do not read the bracketed numbers by now).

Here the second difficulty appears : the counted number of cycles of length 6 is not a multiple of 6. Hence, we forgot counting some cycles. By looking carefully to the dependency graph of the bit vertex 1 again, we see that we did not take into account the "hidden" cycles $(1, 3, 5, 6, 7, 2, 1)$, $(1, 3, 5, 6, 8, 2, 1)$, $(1, 4, 5, 6, 7, 2, 1)$ and $(1, 4, 5, 6, 8, 2, 1)$. One of them is represented by dotted line in Figure 5.14. Taking them into account (they are bracketed in the last table), we find the correct following results : the graph has $4/2 = 2$ cycles of length 2, $16/4 = 4$ cycles of length 4 and $24/6 = 4$ cycles of length 6.

The "hidden" cycles are not only difficult to see, it is also difficult to find them algorithmically. Hence, we decide not to count them. We obtain a

top-level bit of the DG	type of the DG	length 2 cycle number	length 4 cycle number	length 6 cycle number
0	0	0	4	0
1	1	0	2	0 [4]
2	1	0	2	0 [4]
3	3	1	1	2
4	3	1	1	2
5	1	0	2	0 [4]
6	1	0	2	0 [4]
7	3	1	1	2
8	3	1	1	2
counted cycles		4	16	8 [24]
number of cycles		2	4	1.33 [4]

Table 5.1: Cycle count for the $n = 3$, $N = 9$ GLD code with random interleaver.

lower bound on the number of cycles in the graph as follows : let $n_{max}(l)$ be the maximum number of cycles of length l counted over all the dependency graphs. Let $s(l)$ be the sum of the cycles of length l counted in each dependency graph. The number $n(l)$ of cycles of length l verifies :

$$n(l) \leq \max\left\{n_{max}(l), \left\lceil\frac{s(l)}{l}\right\rceil\right\} \tag{5.27}$$

We used this method to compare the cycle distribution of the GLD codes based on the $(12, 8, 3)$ shortened Hamming code presented in Section 5.5.1. The results are plotted in Figure 5.15.

As already known, the PG$(2, 11)$ interleaver has only cycles of length 6. The random interleaver has cycles of length from 4 to 8. The row-column interleaver behavior is completely different. It has much more cycles, and they extend to much higher length values.

We can conclude from these results that the row-column interleaver is not a good choice : we have to iterate much more ($26/2 = 13$ steps) so as to make the whole information available everywhere in the graph, and this information is highly correlated since the number of cycles is large.

These results also proved that an algebraic interleaver of girth 6 leads to

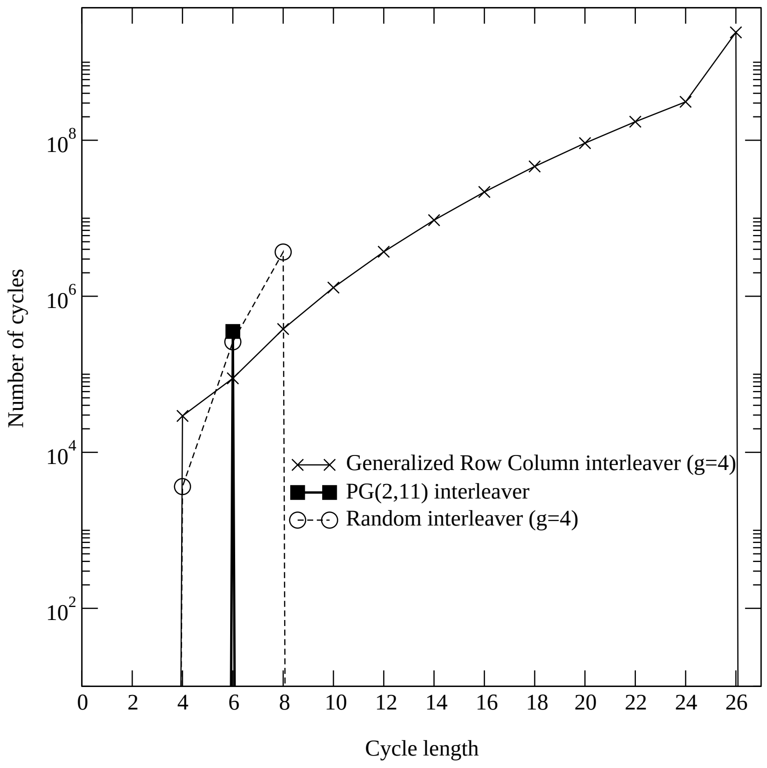


Figure 5.15: Cycle distributions of GLD codes based on the $(12, 8, 3)$ shortened Hamming code.

better performance than a random interleaver of girth 4. Indeed, it has less cycles, and they occur later. Since, for this code length, the PG$(2, 11)$ is the only interleaver with a girth of 6 (apart from an edge permutation), there is no randomly chosen interleaver with girth 6. Hence, in this case, random choice of the interleaver is not the best solution and a strong structured construction leads to better results.

Conclusions and Perspectives

Conclusions

In this thesis, we built new tools to understand and evaluate iterative SISO decoding of different compound codes (Turbo, LDPC and GLD codes). Dependency graphs are useful to conceptually understand the iterative decoding and to derive new design criteria for interleavers specifically adapted to this decoding. They illuminate the influence of cycles, and predict that bits involved in short length cycles are less protected.

APP propagation is a fruitful way to estimate the performance of compound codes under iterative decoding. Even if it assumes a cycle free graph, and so an infinite interleaver, it gives complementary information with respect to the classical maximum likelihood decoding performance.

Combining the results of Gallager on LDPC codes and those of Tanner on graph codes, we built the GLD class of compound codes and described an effective iterative decoding scheme. We proved that these codes are asymptotically good with only two levels. In this case, the new *a priori* information arriving at the input of any supercode decoder only contains the interleaved extrinsic information of the other supercode decoder. In the case of LDPC codes, the new *a priori* information is the product of $j-1$ extrinsic information. Therefore, the GLD decoding schedule is efficient and simple, and parallel decoding implementation can reduce the latency. GLD codes are both a generalization of, and an attractive alternative to product codes, thanks to their higher flexibility in term of code length as shown in the comparative simulations that we conducted.

We finally constructed new interleavers for GLD codes that present good properties in terms of iterative decoding. They are based on algebraic graphs and perform better (their iterative decoding speed is higher) than random interleavers for small code length.

Perspectives

Further investigations need to be carried out. A best characterization of the window size used in the dependency graphs of turbo codes would be of great interest.

Another point that has not been studied is the encoding algorithm of GLD codes. We currently systematize the parity check matrix to produce the associated generator matrix. Even if the systematization is done offline once and that the encoding complexity is not too high for small block length, this method is a drawback of GLD codes compared to other compound codes for medium length. An efficient encoding process taking into account the graphical representation of the GLD may be found, to avoid the explicit construction of the generator matrix. This task could be easier for GLD codes based on deterministic algebraic interleavers. Some research has been already done in this direction for LDPC codes by Spielman [96] and Luby *et. al.* [70].

The promising approach of deterministic algebraic interleavers for small length may be extended to other compound codes, such as turbo codes. In the latter case, the exploitation of the graph structure is not straightforward since it is less natural than the graphical representation of GLD codes.

Appendix A

First cycle distribution of dependency graphs constructed from row-column interleavers

Among the parameters influencing the performance of turbo coding schemes, the choice of the interleaver of fixed length N is important. We theoretically show that independently of its size, the row-column deterministic interleaver achieves always poor performance. The criterion involved in this study is related to the construction of the dependency graphs of the turbo codes (see Chapter 2), more precisely to the height (number of levels) where the first collisions occur.

We show that *all the row column interleavers involved in a turbo coding scheme lead to dependency graphs of same first cycle level, that is 3.*

A.1 Row-column interleaver definition

The row-column interleavers are characterized by two parameters (k, n), respectively their number of rows and columns. Their total length is $N = kn$. We assume $k > 1$ and $n > 1$, to rule the trivial identity interleaver out. The input data is written row-by-row into a $k \times n$ matrix, and then copied column-by-column as the output.

There is an kn-periodic arithmetic relationship between the data input- and output order. Let d_i be the input of the interleaver at time i ($i \in [0, kn-1]$). If we don't take into account the delay, this data outputs at time $j(i)$, such that

$$i \to j(i) = (i \bmod n) \times k + \left\lfloor \frac{i}{n} \right\rfloor . \tag{A.1}$$

The corresponding de-interleaver is the (n, k) interleaver. Hence, we have the following formula

$$j \to i(j) = (j \bmod k) \times n + \left\lfloor \frac{j}{k} \right\rfloor . \tag{A.2}$$

We easily check that $i(j(i)) = i$.

0	1	$\to \cdots \to$	$n-1$
n	$n+1$	$\to \cdots \to$	$2n-1$
$\vdots$			$\vdots$
$(k-1)n$		$\to \cdots \to$	$kn-1$

input order of the data in the interleaver

$0'$	k'	$\cdots$	$((n-1)k)'$
$1'$	$(k+1)'$		
$\downarrow$ $\vdots$ $\downarrow$	$\downarrow$ $\vdots$ $\downarrow$		$\downarrow$ $\vdots$ $\downarrow$
$(k-1)'$	$(2k-1)'$	$\cdots$	$(kn-1)'$

output order of the data from the interleaver

A.2 Graphical properties of row-column interleavers

Any interleaver can be graphically seen as a deterministic matching between two chains whose vertices represent respectively the input- and output data of the interleaver.

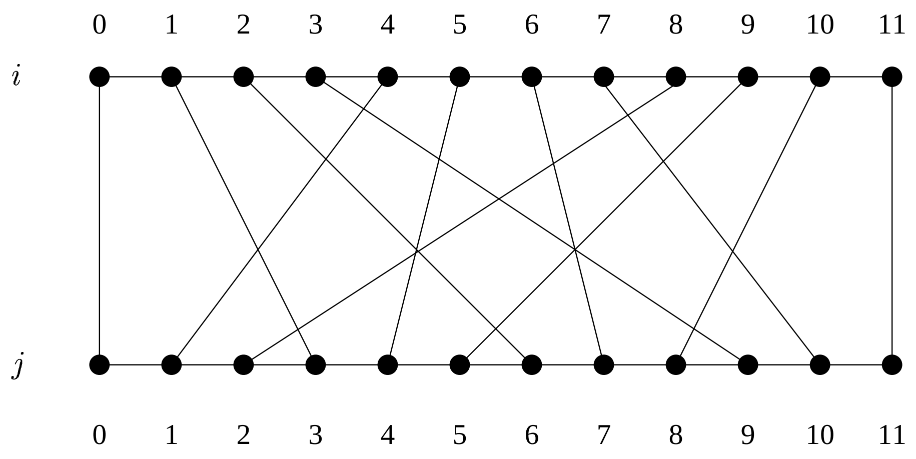


Figure A.1: Graph of the (3, 4) interleaver.

An example is given at Figure (A.1), where we can easily check the relationships (A.1) and (A.2).

We discuss two important results on row-column interleaver graphs.

Theorem A.2.1 *Each vertex in the input chain has at least one neighbor such that the corresponding vertices in the output chain are at a distance of k.*

Proof A.2.1 Let i and $i+1$ be the indices of two adjacent vertices in the input chain ($\forall i \in [0, k \times n - 2]$). We distinguish two cases.

- 1° case : $i \neq u \times n - 1$, $\forall u \in [1, k-1]$, hence we have

$$\left.\begin{array}{c} \left\lfloor \frac{i}{n} \right\rfloor = \left\lfloor \frac{i+1}{n} \right\rfloor \\ (i + 1 \bmod n) = (i \bmod n) + 1 \end{array}\right\} \Rightarrow |j(i+1) - j(i)| = k$$

- 2° case : $i = u \times n - 1$, $\forall u \in [1, k-1]$. Then the preceding equality is no longer true. Although the exact value of $|j(i+1) - j(i)|$ could be calculated, we don't need it.

Each vertex of index β has two neighbors of indices α and γ, except for the ends (case follows). One at most of the two couples of adjacent vertices (α, β)

and (β, γ) can correspond to the 2° case, otherwise we would have $n = 1$, which is impossible. Hence, taking the (one of the) couple(s) corresponding to the 1° case, the result is straightforward.

Let's now look at the ends. Both for $(0, 1)$ and for $(kn-2, kn-1)$, there is no $u \in [1, k-1]$ such that these couples can be written as $(un-1, un)$. Hence, the ends correspond to the 1° case, and the claimed result holds for any couple of adjacent vertices. □

Theorem A.2.2 *Each couple of vertices at a distance k in the output chain has a corresponding couple of vertices in the input chain which are adjacent.*

Proof A.2.2 Let j and $j+k$ ($\forall j \in [0, k(n-1)+1]$) be the indices of two vertices at a distance of k in the output chain. Due to (A.2) we have

$$\begin{aligned} i(j+k) &= (j + k \bmod k) \times n + \left\lfloor \frac{j+k}{k} \right\rfloor \\ &= (j \bmod k) \times n + \left\lfloor \frac{j}{k} \right\rfloor + 1 \end{aligned}$$

and thus

$$|i\,(j+k) - i\,(j)| = 1$$

□

Let's consider any (k, n) row-column interleaver, and the dependency graph of any of its vertices of the input chain, say that of index α. Due to Theorem (2.1) this vertex has at least one neighbor of index β such that their corresponding vertices thru the interleaver (of indexes γ and δ), i.e., in the output chain are at a distance of k. Their level is 2 in the dependency graph.

Let's examine their respective neighbors (of index $\gamma-1$ and $\delta-1$, or $\gamma+1$ and $\delta+1$; at the ends, only one couple exists). They too are at a distance of k. Due to Theorem (2.2), their corresponding nodes thru the interleaver are adjacent, at level 3 in the dependency graph. Thus, there is a collision at this level, as shown in Figure (A.2).

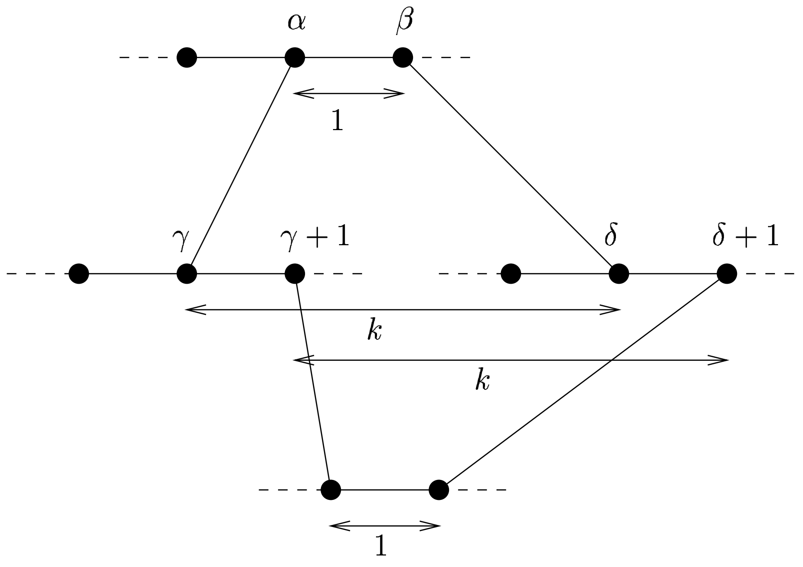


Figure A.2: First collision in the dependency Graph at level 3

We have shown that *all the row-column interleavers involved in a turbo coding scheme lead to dependency graphs where the height of first collision is the same, namely 3.*

Appendix B

The forward-backward algorithm*

The forward-backward (FB) algorithm [3] computes the *a posteriori* probabilities associated with the states and transitions of a Markov source whose outputs are observed through a memoryless channel. Hence, it is not *stricto sensu* a decoding algorithm : it does not give decisions on the states or transitions of the Markov source.

We used it in our work as a soft-input soft-output (SISO) decoder for convolutional and linear block codes.

The Viterbi algorithm [42] used on convolutional codes is a maximum likelihood (ML) decoder that minimizes the *word* error probability. Its soft-output version (SOVA) [8] [51] gives decisions together with reliability information on the bits, without any guaranteed optimality. The FB algorithm is optimal in the sense of ML : the decisions that are taken from the *a posteriori* probabilities it gives minimize the *symbol* error probability, not the *word* error probability. Hence, the decoded symbols can correspond to a path that is not in the trellis.

The FB algorithm is first presented on a general Markov source, and its applications to convolutional and linear block codes are explained in a second step. Complexity issues are finally discussed in the case of SISO decoding of linear block codes.

*This appendix is the English translation of a part of a lesson I taught to ENST students.

B.1 Markov source Model

B.2 Markov source model

Let us recall some properties of a finite-state discrete Markov source. It is characterized a each instant t by its state $S_t = m$, $m \in \{0, \cdots, M-1\}$ and its output X_t. M is called the number of states and X is an element of $\mathcal{X}$, its alphabet. The cardinality of $\mathcal{X}$ is finite.

The transitions between states are characterized by their probabilities :

$$\text{Prob}_t \left(S_t = m | S_{t-1} = m_1, S_{t-2} = m_2, \cdots, S_{t-\nu} = m_\nu\right). \tag{B.1}$$

where ν is the memory of the Markov source. These probabilities may be time-variant.

The outputs are also characterized by their probabilities :

$$\text{Prob}_t \left(X_t = X | S_t = m, S_{t-1} = m_1, S_{t-2} = m_2, \cdots, S_{t-\nu} = m_\nu\right). \tag{B.2}$$

where $X \in \mathcal{X}$. Thus, these probabilities only depend on the current state and the past ν state transitions.

We limit our study to Markov sources with memory $\nu = 1$, which involves no restriction of generality for a discrete source. In this case, the transitions can be characterized by a matrix P_t, called the state transition matrix, of size $M \times M$, where :

$$P_t^{i,j} = p_t\left(i|j\right) = \text{Prob}_t\left(S_t = i | S_{t-1} = j\right). \tag{B.3}$$

Similarly, the source outputs are characterized by the probabilities :

$$\text{Prob}_t \left(X_t = X | S_t = m, S_{t-1} = m'\right) \tag{B.4}$$

B.3 Discrete memoryless channel model

A discrete memoryless channel (DMC) has the following property : Let X_1^t denote a stream of t symbols belonging to $\mathcal{X}$, the channel input alphabet, and let Y_1^t be the stream of the corresponding output symbols (called the observations), belonging to $\mathcal{Y}$, the channel output alphabet. Knowing the transition probabilities $R_t(y|x)$ of the channel, for $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, we can write :

$$\text{Prob}\left(Y_1^t|X_1^t\right) = \prod_{j=1}^{t} R_t\left(Y_j|X_j\right) \tag{B.5}$$

where $Y_1^T = \{Y_1, \cdots, Y_T\}$.

B.4 The Forward-Backward Algorithm

B.5 Forward-backward algorithm

Let us consider an M-state Markov source of memory 1. Without loss of generality, we consider it as time-invariant *i.e.* :

$$\text{Prob}_t\left(S_t = m|S_{t-1} = m'\right) = p\left(S_t = m|S_{t-1} = m'\right) \tag{B.6}$$
$$\text{Prob}_t\left(X_t = X|S_t = m, S_{t-1} = m'\right) = q\left(X_t = X|S_t = m, S_{t-1} = m'\right) \tag{B.7}$$

at any instant t.

The output symbols X_t of this Markov source are the inputs of a DMC channel, and their corresponding outputs are Y_t. Let us assume that at times $t = 0$ and $t = T$, the state of the source is known, namely : $S_0 = 0$ and $S_T = 0$.

The aim of the FB algorithm is to compute the following *a posteriori* probabilities (APP) :

$$\text{Prob}\left(S_t = m|Y_1^T\right) \tag{B.8}$$
$$\text{Prob}\left(S_t = m; S_{t-1} = m'|Y_1^T\right) \tag{B.9}$$

at any time t. These are the state and state transition probabilities, respectively, given all the available observations.

Let us define :

$$\lambda_t\left(m\right) = \text{Prob}\left(S_t = m; Y_1^T\right) \tag{B.10}$$

and

$$\sigma_t(m, m') = \text{Prob}\left(S_t = m; S_{t-1} = m'; Y_1^T\right) \tag{B.11}$$

We have $\lambda_t(m) \propto \text{Prob}\left(S_t = m | Y_1^T\right)$ and $\sigma_t(m, m') \propto \text{Prob}\left(S_t = m; S_{t-1} = m' | Y_1^T\right)$. Hence, we will calculate λ_t and σ_t instead of the quantities (B.8) and (B.9).

B.5.1 Definition of the quantities α, β and γ

Let us define :

$$\alpha_t(m) = \text{Prob}\left(S_t = m; Y_1^t\right), \tag{B.12}$$

$$\beta_t(m) = \text{Prob}\left(Y_{t+1}^T | S_t = m\right), \tag{B.13}$$

and :

$$\gamma_t(m, m') = \text{Prob}\left(S_t = m; Y_t | S_{t-1} = m'\right) \tag{B.14}$$

We will derive expressions of λ_t and σ_t as functions of α, β and γ. We have :

$$\begin{aligned}\lambda_t(m) &= \text{Prob}\left(S_t = m; Y_1^t\right) \text{Prob}\left(Y_{t+1}^T | S_t = m; Y_1^t\right) \\ &= \text{Prob}\left(S_t = m; Y_1^t\right) \text{Prob}\left(Y_{t+1}^T | S_t = m\right) \\ &= \alpha_t(m)\, \beta_t(m) \end{aligned} \tag{B.15}$$

since the “future” outputs of a Markov source do not depend on the previous outputs. As the outputs of the channel depend directly on the outputs of the source, this property is also true for them.

Using the same property, we have:

$$\begin{aligned}\sigma_t(m, m') &= \text{Prob}\left(S_t = m; S_{t-1} = m'; Y_1^t\right) \text{Prob}\left(Y_{t+1}^T | S_t = m; S_{t-1} = m'; Y_1^t\right) \\ &= \text{Prob}\left(S_t = m; S_{t-1} = m'; Y_1^t\right) \text{Prob}\left(Y_{t+1}^T | S_t = m\right) \\ &= \text{Prob}\left(S_{t-1} = m'; Y_1^{t-1}\right) \text{Prob}\left(S_t = m; Y_t | S_{t-1} = m'; Y_1^{t-1}\right) \times \\ &\quad\ \text{Prob}\left(Y_{t+1}^T | S_t = m\right) \\ &= \text{Prob}\left(S_{t-1} = m'; Y_1^{t-1}\right) \text{Prob}\left(S_t = m; Y_t | S_{t-1} = m'\right) \times \\ &\quad\ \text{Prob}\left(Y_{t+1}^T | S_t = m\right) \\ &= \alpha_{t-1}(m')\, \gamma_t(m, m')\, \beta_t(m) \end{aligned} \tag{B.16}$$

We can give a "visual" interpretation of this last equation : α represents the joint probability of the previous observations and the starting state of the transition, β represents the probability of the future observation given the ending state of the transition, and γ represents the transition metric.

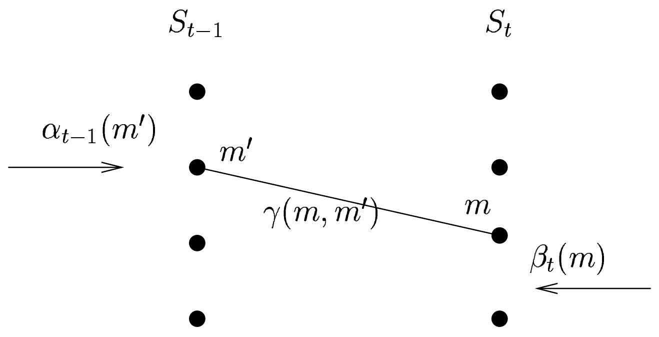


Figure B.1: Interpretation of the expression of σ

We now have to compute the quantities α, β and γ.

B.5.2 Recursive computation of α

$$\begin{aligned}
\alpha_t(m) &= \text{Prob}\left(S_t = m; Y_1^t\right) \\
&= \sum_{m'=0}^{M-1} \text{Prob}\left(S_t = m; S_{t-1} = m'; Y_1^t\right) \\
&= \sum_{m'=0}^{M-1} \text{Prob}\left(S_{t-1} = m'; Y_1^{t-1}\right) \text{Prob}\left(S_t = m; Y_t | S_{t-1} = m'; Y_1^{t-1}\right) \\
&= \sum_{m'=0}^{M-1} \text{Prob}\left(S_{t-1} = m'; Y_1^{t-1}\right) \text{Prob}\left(S_t = m; Y_t | S_{t-1} = m'\right) \\
&= \sum_{m'=0}^{M-1} \alpha_{t-1}(m') \gamma_t(m,m') \qquad \text{(B.17)}
\end{aligned}$$

Hence, the αs can be computed recursively if the γs are known. This recursion is the "forward" part of the algorithm.

B.5.3 Recursive computation of β

$$\begin{aligned}
\beta_t(m) &= \text{Prob}\left(Y_{t+1}^T | S_t = m\right) \\
&= \sum_{m'=0}^{M-1} \text{Prob}\left(S_{t+1} = m'; Y_{t+1}^T | S_t = m\right) \\
&= \sum_{m'=0}^{M-1} \text{Prob}\left(S_{t+1} = m'; Y_{t+1} | S_t = m\right) \text{Prob}\left(Y_{t+2}^T | S_t = m; S_{t+1} = m'; Y_{t+1}\right) \\
&= \sum_{m'=0}^{M-1} \text{Prob}\left(S_{t+1} = m'; Y_{t+1} | S_t = m\right) \text{Prob}\left(Y_{t+2}^T | S_{t+1} = m'\right) \\
&= \sum_{m'=0}^{M-1} \beta_{t+1}(m') \, \gamma_{t+1}(m', m) \qquad \text{(B.18)}
\end{aligned}$$

As the αs, the βs are computed recursively, but in the opposite direction. This is the "backward" part of the algorithm.

B.5.4 Expression of γ

In the "forward" and "backward" parts, it is necessary to know the values of γ to achieve the recursions. This computation does not strictly belong to the algorithm, since γ is the probability of an observation given a transition. The channel and the source characteristics are involved. Assuming a discrete memoryless channel, we have :

$$\begin{aligned}
\gamma_t(m, m') &= \text{Prob}\left(S_t = m; Y_t | S_{t-1} = m'\right) \\
&= \text{Prob}\left(S_t = m | S_{t-1} = m'\right) \text{Prob}\left(Y_t | S_t = m; S_{t-1} = m'\right) \\
&= \sum_{X \in \mathcal{X}} \text{Prob}\left(S_t = m | S_{t-1} = m'\right) \text{Prob}\left(X_t = X; Y_t | S_t = m; S_{t-1} = m'\right) \\
&= \sum_{X \in \mathcal{X}} \text{Prob}\left(S_t = m | S_{t-1} = m'\right) \text{Prob}\left(X_t = X | S_t = m; S_{t-1} = m'\right) \times \\
&\qquad \text{Prob}\left(Y_t | X_t = X; S_t = m; S_{t-1} = m'\right) \\
&= \sum_{X \in \mathcal{X}} \text{Prob}\left(S_t = m | S_{t-1} = m'\right) \text{Prob}\left(X_t = X | S_t = m; S_{t-1} = m'\right) \times \\
&\qquad \text{Prob}\left(Y_t | X_t\right) \\
&= \sum_{X \in \mathcal{X}} p(m | m') \, q(X | m, m') \, R(Y_t | X_t) \qquad \text{(B.19)}
\end{aligned}$$

where p, q and R are defined by (B.6), (B.7) and (B.5), respectively.

We can now compute all the variables of the FB algorithm, using (B.19), (B.17), (B.18), (B.15) and (B.16).

B.5.5 Initialization

The two recursive relations (B.17) and (B.18) need an initialization of the first (respectively, the last) term. We use the assumption on the states of the Markov source at the initial and final instants. We have $S_0 = 0$ and $S_T = 0$. This implies :

$$\begin{cases} \alpha_0\left(0\right) = 1, \\ \alpha_0\left(m\right) = 0, \quad \forall m \in \{1, \ldots, M-1\} \end{cases} \tag{B.20}$$

and :

$$\begin{cases} \beta_T\left(0\right) = 1, \\ \beta_T\left(m\right) = 0, \quad \forall m \in \{1, \ldots, M-1\} \end{cases} \tag{B.21}$$

B.5.6 Summary of the FB algorithm

The forward-backward algorithm needs the *a priori* knowledge of the Markov source :

- $p\left(m|m'\right)$, the transition probability,
- $q\left(X|m, m'\right)$, the output probability.

It also needs the transition probabilities of the channel :

- $R\left(Y|X\right)$

The observations at the channel output Y_1^T are inputs to the algorithm. Its aim is to compute the *a posteriori* probabilities :

- of the states $\text{Prob}\left(S_t = m|Y_1^T\right)$,
- of the state transitions $\text{Prob}\left(S_t = m; S_{t-1} = m'|Y_1^T\right)$.

It proceeds in several steps :

1. **initialization :** the αs and βs are initialized using (B.20) and (B.21).
2. **computation of the γs :** they are computed using (B.19).
3. ***forward* recursion :** the αs are computed using (B.17). If the observations arrive in a sequential order, the last two steps don't induce any latency.
4. ***backward* recursion :** the βs are computed using (B.18). This computation can only begin when the last observation is available.
5. **computation of the *a posteriori* probabilities :** (B.15) and (B.16) are used.

In a practical implementation of this algorithm, normalizing all probabilities is necessary. The block diagram of the forward-backward algorithm is presented in Figure B.2

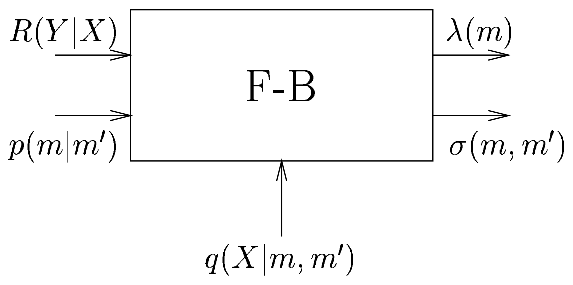


Figure B.2: Forward-backward algorithm block diagram

B.6 Application of the FB algorithm to the SISO decoding of convolutional codes

A binary convolutional encoder of rate k/n can be seen as a Markov source of memory 1 and $2^{k(L-1)}$ states, where L is the constraint length of the code.

Furthermore, the outputs of an encoder are *deterministic* functions of the transitions. Hence, the output probabilities

$$q_t\left(X|m,m'\right) = \text{Prob}_t\left(X_t = X|S_t = m, S_{t-1} = m'\right) \qquad \text{(B.22)}$$

are Kronecker δ-functions of X, whose value is 1 if X is the output associated with the state transition (m',m), else 0. X_t is a binary vector of length n, corresponding to the output of the encoder at time t.

We use the forward-backward algorithm only to compute the *a posteriori* state transition probabilities $\sigma_t(m,m')$. Indeed, these transitions are related to the encoder input. Hence, it is possible to get the *a posteriori* probabilities on the encoder input, *i.e.* the information bits.

Here again, we restrict our study to time-invariant convolutional codes. Let us denote by $U_t = \left\{u_t^1, \cdots, u_t^k\right\}$ the inputs of the encoder at time t and $X_t = \{x_t^1, \cdots, x_t^n\}$ its outputs at the same time. The decoder has access to $Y_t = \{y_t^1, \cdots, y_t^n\}$,$t = 1, \cdots, T$, which are the outputs of the DMC associated with X_t.

Our aim is to compute the *a posteriori* probabilities on the encoder input :

$$\text{APP}\left(u_t^i\right) = \text{Prob}\left(u_t^i|Y_1^T\right) \quad \forall i \in \{1,\cdots,k\}, \quad \forall t \in \{1,\cdots,T\} \qquad \text{(B.23)}$$

But the encoder operation results in a *deterministic* relation between the inputs and the state transitions. Let $\mathcal{T}_u^i$ be the set of state transitions for which the i-th bit is u and let T be one of them. We have :

$$\text{APP}\left(u_t^i = u\right) = \sum_{T\in\mathcal{T}_u^i} \sigma_t\left(T\right) = \sum_{m',m/u_t^i=u} \sigma_t\left(m,m'\right) \qquad \text{(B.24)}$$

Hence, SISO decoding of a convolutional code can be performed using the FB algorithm, followed by a summation over the correct *a posteriori* state transition probabilities to find the APP of the bits.

Using (B.16) and (B.19), we have :

$$\begin{aligned}\text{APP}\left(u_t^i = u\right) &= \sum_{m',m/u^i=u} \alpha_{t-1}\left(m'\right) \sum_{X\in\mathcal{X}} p\left(m|m'\right) q\left(X|m,m'\right) R\left(Y_t|X_t\right) \beta_t\left(m\right) \\ &= \sum_{m',m/u^i=u} p_t\left(m|m'\right) \alpha_{t-1}\left(m'\right) R\left(Y_t|X\left(m',m\right)\right) \beta_t\left(m\right) \qquad \text{(B.25)}\end{aligned}$$

The last equality is found considering the fact that $q\left(X|m,m'\right)$ is a Kronecker δ-function of X : either X corresponds to the only output of the code associated with the (m',m) transition, and thus $q\left(X|m,m'\right) = 1$, either $q\left(X|m,m'\right) = 0$. Hence, we denote by $X(m',m)$ the outputs of the encoder associated with the (m',m) transition. The simplified version of the γ formula follows :

$$\gamma_t\left(m,m'\right) = p_t\left(m|m'\right) R\left(Y_t|X\left(m',m\right)\right) \tag{B.26}$$

The term $p_t\left(m|m'\right)$ corresponds to an *a priori* knowledge that the decoder can have on the transitions.

B.6.1 Simple SISO decoder for convolutional codes based on the forward-backward algorithm

In the case where a single convolutional code is decoded, there is no *a priori* information on the transitions if the encoder input is iid. Then, $p_t\left(m|m'\right)$ is a constant, initialized to 2^{-k} for all the permitted transitions (m',m). The block diagram of such a SISO decoder is presented in Figure B.3.

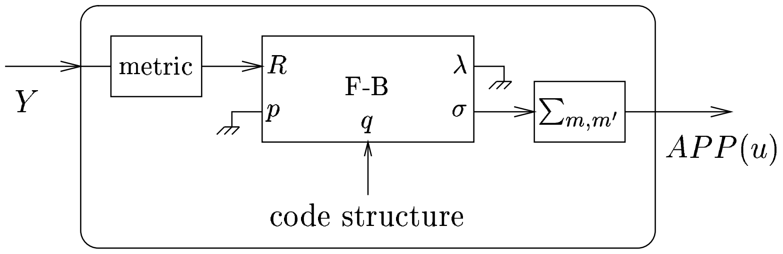


Figure B.3: SISO decoder for an isolated convolutional code

B.6.2 SISO decoder for convolutional codes based on the forward-backward algorithm suitable for iterative decoding

In the context of iterative SISO decoding, *a priori* probabilities on each information bits $\pi\left(u_t^i\right)$ are known. If we assume that these probabilities are *independent*, the *a priori* probabilities of the transitions are related to the *a priori* probabilities of the information bits according to :

$$p_t\left(m|m'\right) = \prod_{i=1}^{k} \pi\left(u_t^i = u^i(m',m)\right) \tag{B.27}$$

where $u^i(m',m)$ is the value of the i-th bit of the (m',m) transition.

The APP of the information bits are computed just as in the previous case, that is, using (B.25). We now have to extract the extrinsic probability of each bit u_t^i from its APP. The extrinsic probability is defined as the novelty in the APP that comes from the decoder. The APP can be rewritten :

$$\begin{aligned}
\text{APP}\left(u_t^i = u\right) &= \sum_{m',m/u^i=u} \alpha_{t-1}\left(m'\right) \left\{ \begin{array}{c} \prod\limits_{i'=1}^{k} \pi\left(u_t^{i'} = u^{i'}(m',m)\right) \\ \times \prod\limits_{j=1}^{n} R\left(y_t^j|y^j\left(m',m\right)\right) \end{array} \right\} \beta_t\left(m\right) \\
&= \pi\left(u_t^i = u\right) \sum_{m',m/u^i=u} \alpha_{t-1}\left(m'\right) \left\{ \begin{array}{c} \prod\limits_{\substack{i'=1 \\ i'\neq i}}^{k} \pi\left(u_t^{i'} = u^{i'}(m',m)\right) \\ \times \prod\limits_{j=1}^{n} R\left(y_t^j|y^j\left(m',m\right)\right) \end{array} \right\} \beta_t\left(m\right)
\end{aligned}$$

The last equality holds because the *a priori* term $\pi\left(u_t^i = u\right)$ is common to all the products in the sum.

Two cases have to be distinguished whether the convolutional code is systematic or not.

Non systematic convolutional code

Since the information bits are not directly present as decoder outputs, and thus are not observable, the only available information concerning the bit u_t^i at the decoder input is its *a priori* information. Hence, its extrinsic probability is :

$$\begin{aligned}
\text{Ext}\left(u_t^i = u\right) &= \sum_{m',m/u^i=u} \alpha_{t-1}\left(m'\right) \left\{ \begin{array}{c} \prod\limits_{\substack{i'=1 \\ i'\neq i}}^{k} \pi\left(u_t^{i'} = u^{i'}(m',m)\right) \\ \times \prod\limits_{j=1}^{n} R\left(y_t^j|y^j\left(m',m\right)\right) \end{array} \right\} \beta_t\left(m\right) \\
&= \sum_{m',m/u^i=u} \alpha_{t-1}\left(m'\right) \widetilde{\gamma_t^i}(m,m')\beta_t\left(m\right) = \sum_{m',m/u^i=u} \widetilde{\sigma_t^i}\left(m,m'\right)
\end{aligned}$$

where $\widetilde{\gamma_t^i}(m,m')$ is just $\gamma_t(m,m')$ where the term depending on the bit u_t^i has been taken off. Hence, the extrinsic information can be seen as the output of a modified FB algorithm, where steps 1 to 4 described in Section B.5.6 are unchanged, and where two different metrics $\gamma_t(m,m')$ and $\widetilde{\gamma_t^i}(m,m')$ are used at step 5 to compute respectively the *a posteriori* state transition probabilities $\sigma_t(m,m')$ and the extrinsic state transition probabilities $\widetilde{\sigma_t^i}(m,m')$ for all information bits lying on the transitions.

Systematic convolutional code

The output of the convolutional encoder at any time t is :

$$X_t = \left\{u_t^1, \cdots, u_t^k, x_t^1, \cdots, x_t^{n-k}\right\} \tag{B.28}$$

where the u's are information bits and the x's parity bits. Thus, we have :

$$R\left(Y_t|X_t\right) = \prod_{i'=1}^{k} R\left(y_t^{i'}|u_t^{i'}\right) \prod_{j=1}^{n-k} R\left(y_t^{k+j}|x_t^j\right) \tag{B.29}$$

Defining :

$$\widetilde{\gamma_t^i}(m,m') = \prod_{\substack{i'=1 \\ i'\neq i}}^{k} \pi\left(u_t^{i'} = u^{i'}(m',m)\right) \prod_{\substack{i'=1 \\ i'\neq i}}^{k} R\left(y_t^{i'}|u^{i'}(m',m)\right) \prod_{j=1}^{n-k} R\left(y_t^{k+j}|x^j(m',m)\right) \tag{B.30}$$

and :

$$\widetilde{\sigma_t^i}(m,m') = \alpha_{t-1}\left(m'\right)\widetilde{\gamma_t^i}(m,m')\beta_t\left(m\right) \tag{B.31}$$

$\widetilde{\gamma_t^i}(m,m')$ is the metric of the transition (m',m) where informations concerning the bit u_t^i available at the decoder input (namely its *a priori* probability and its observation) have been taken off. Hence, its extrinsic probability is :

$$\text{Ext}\left(u_t^i = u\right) = \sum_{m',m/u^i=u} \widetilde{\sigma_t^i}\left(m,m'\right) \tag{B.32}$$

The same modified FB algorithm with two metrics as stated for the non systematic convolutional code is thus used to compute the APP and extrinsic probabilities of the information bits.

With both types of SISO decoder, it is also possible to compute these probabilities for the coded bits, by summing the corresponding state transition probabilities. This is specially useful in the iterative (turbo) decoding of serial concatenated convolutional codes where the outer code is often non systematic.

B.6.3 Initial and final states initialization

Another point to sketch out is that the FB algorithm needs that the starting and ending states of the encoder are given. Knowing the initial state of the encoder is not a problem since the all-zero state is, in general, forced as the encoder initial state. However, the decoder has no special information available regarding its final state. Three solutions can be used :

- initialization of β to $1/M$: no information concerning the final state is provided to the decoder : Hence, all the final states have same probability.
- addition of tail bits at the end of the information bit stream, in such a way that the final state of the encoder is forced to zero. This solution has been widely used in the turbo codes context, even if it slightly decreases the rate of the code, considered as a block code. In this case, the initialization of the βs follows (B.21).
- "tail-biting" solution : particular convolutional codes can be designed such that the encoder assumes again the initial state when the encoding is completed. Therefore, the decoding trellis can be seen as circular, and an FB algorithm with unknown initial and final states is used on a length greater than T. This method has also been applied to parallel turbo codes [23].

B.7 Application of the FB algorithm to the SISO decoding of a linear block codes

The forward-backward algorithm can be used as a SISO decoder for any code that has a trellis representation. So are the linear block codes. The trellis representation of linear block codes is a wide topic (see [101]). Trellis design procedures for both the syndrome and coset trellises together with construction of codes with optimum bit position reordering leading to smaller trellises are studied in [101], [59] and [66]. Immediate construction based on the polynomial description of linear block codes is presented in [6], [9], [4] and [5].

The simpler trellis representation of a linear block code $\mathcal{C}_0$ is its syndrome trellis, which is easily constructed from its parity-check matrix H_0. We present this construction for the $(7,4,3)$ Hamming code whose matrix H_0 is :

$$H_0 = \left[\begin{array}{cccccccc} & c_0 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 \\ s_0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ s_1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ s_2 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right] \qquad \text{(B.33)}$$

Any codeword $\underline{c} = \{c_0, \cdots, c_6\}$ satisfies :

$$\underline{s} = \{s_0, s_1, s_2\} = \sum_{i=0}^{6} c_i \underline{h}_i^t = \{0,0,0\} \qquad \text{(B.34)}$$

where $\underline{h}_i$ is the i-th row of H_0. The syndrome trellis is based on the partial syndromes, that is :

$$\underline{s}^j = \left\{s_0^j, s_1^j, s_2^j\right\} = \sum_{i=0}^{j-1} c_i \underline{h}_i^t \qquad j = 1, \cdots, 7 \qquad \text{(B.35)}$$

and $\underline{s}^0 = \{0,0,0\}$ by definition. For any codeword, we obviously have $\underline{s}^7 = \underline{s} = \{0,0,0\}$.

The syndrome trellis is defined as follows :
– its 2^{n-k} states are labelled with the partial syndromes values
– its length is n and each transition represents a coded bit value
– hence, the number of transitions per node is at most 2
– the nodes represent the values that the partial syndromes can take for all codewords
– since $\underline{s}^0 = 0$ and $\underline{s}^n = \underline{s} = 0$ for all codewords, the trellis starts and ends at the zero state
– a path in the trellis represents a codeword. Hence, there is 2^k paths
– horizontal transitions represent coded bits with value 0
– oblique transitions represent coded bits with value 1
The syndrome trellis of the $(7,4,3)$ Hamming code is represented in Figure B.4.

This trellis representation is not always the smallest one (in terms of number of states and total number of nodes) especially for codes of rate lower than 1/2. Reordering the bits, and using other equivalent parity-check matrix can lead to smaller trellises. However for the constituent codes used in GLD codes, we use this trellis, as the constituent code rate r has to be greater than 1/2 by construction.

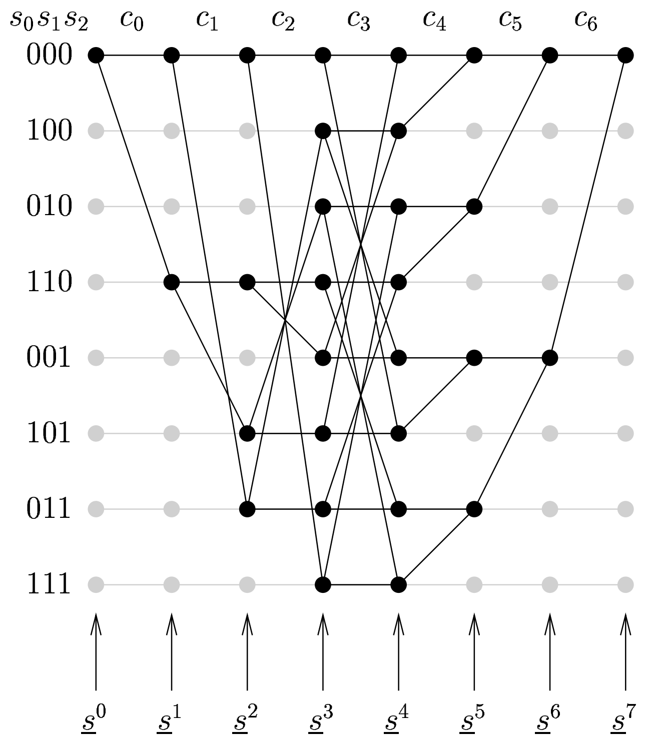


Figure B.4: Syndrome trellis of the $(7, 4, 3)$ Hamming code.

The use of the FB algorithm on this trellis as a SISO decoder of $\mathcal{C}_0$ is straightforward and simple :

- the metric $\gamma_t(m, m')$, $t = 0, \cdots, n-1$ of any transition (m', m) is just the product of the *a priori* probability $\pi(c_t)$ of the coded bit c_t lying on the transition with its likelihood $R(y_t|c_t)$.

- the APP of c_t is computed as for convolutional codes : it is the sum of all the $\sigma_t(m, m')$ for which the transition (m', m) corresponds to the bit value :
$$\mathrm{APP}(c_t) = \sum_{m',m/c_t} \sigma_t(m, m') \tag{B.36}$$

- since the informations related to c_t that are known at the decoder input are its *a priori* probability and its observation, *i.e.* $\gamma_t(m, m')$ where (m', m) is the transition corresponding to the bit, the extrinsic probability of c_t is :

$$\mathrm{Ext}(c_t) = \sum_{m',m/c_t} \alpha(m')\beta(m) \tag{B.37}$$

B.8 Complexity analysis of the SISO decoder of linear block codes based on the FB algorithm

In this section, we give an estimate of the complexity of the FB algorithm used as a SISO decoder for linear block codes in terms of memory usage, floating point additions and product.

The general model of the syndrome trellis used to evaluate the complexity is the following. It is divided in three parts : the *opening* part, the *middle* part and the *closing* part. The opening and closing part are symmetrical. The length of the opening and closing parts is $n - k$ and the length of the middle part is $2k - n$. The number of nodes at time t in the opening part is 2^t. The number of nodes in the middle part is 2^{n-k}. The number of right transitions per node is 2 in the opening and middle part. The number of left transitions per node is 2 in the middle and closing part. This model overestimates the number of nodes since optimization of the trellis is possible, but is valid and useful to perform a general analysis of complexity. It is presented in Figure B.5.

We perform the analysis assuming that a straight implementation of the FB algorithm is used: it deals with probabilities (not LR values). We don't estimate the cost of the normalization, which corresponds to compute both $APP(c_t = 1)$ and $APP(c_t = 0)$ since it can be avoided by using the LR implementation.

computation of γs

Each transition metric $\gamma(m', m)$ is the product of the observation of the corresponding bit with its *a priori* probability. The observations have to be computed for every algorithm, and depend on the channel. Hence, we don't

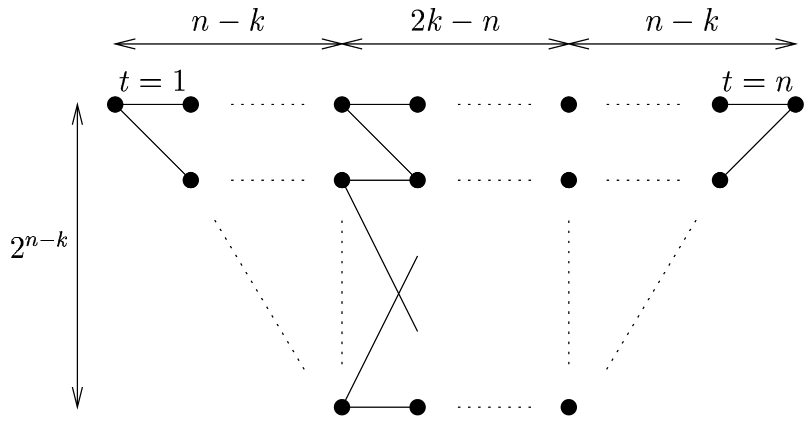


Figure B.5: General trellis model for a (n,k) linear block code.

take it into account. Thus, computing each γ needs 1 product. There are only 2 values of γ ($\gamma(c_t = 1)$ and $\gamma(c_t = 0)$) at each time $t = 1, \cdots, n$ that have to be computed since the metrics share them.

γ **cost : $2n$ products**.

computation of the αs

The three parts of the trellis have to be distinguished in the computation of the αs. The formula used is (B.17).

- *opening part* : with our assumption on the trellis structure each state node is connected to only one previous state node. Hence, there is only one term (and no summation) in (B.17). The total number of products needed for the computation of all the αs in the opening part is $2^{n-k+1} - 3$. The correcting negative term is due to side effects at the beginning of the trellis : the first α is initialized to 1, and thus the following ones are just γs.
- *middle part* : Following (B.17), 2 products and 1 summation are needed for each α. As there is $2^{n-k} \times (2k-n)$ nodes, the computation of this part involves $2^{n-k+1} \times (2k-n)$ products and $2^{n-k} \times (2k-n)$ summations.
- *closing part* : the number of operation per α is the same as in the

middle part, but the total number of nodes (and of αs) in this part is $2^{n-k} - 1$.

α **cost :** $2^{n-k+1} \times (2k - n + 2) - 5$ **products and** $2^{n-k} \times (2k - n + 1) - 2$ **summations**.

computation of the βs

Due to symmetry, the computation load for computing the βs is the same as for the αs.

computation of the extrinsic probabilities

We just compute $\mathrm{Ext}(c_t = 1)$ for $t = 1, \cdots, n$. Following (B.37) one product has to be computed per term in the summation. Hence, we need to evaluate the number of transitions corresponding to $c_t = 1$ for the three parts of the trellis.

- *opening part* :there are $2^{n-k} - 1$ transitions corresponding to $c_t = 1$ in this part, thus $2^{n-k} - 1$ products and $2^{n-k} - n + k - 1$ summations are needed.

- *middle part* : there are $2k - n$ extrinsic values to compute, and each involves 2^{n-k} transitions. Hence it corresponds to $2^{n-k} \times (2k - n)$ products and $(2^{n-k} - 1) \times (2k - n)$ summations.

- *closing part* : the computation load is the same as for the opening part due to symmetry.

extrinsic probabilities cost : $2^{n-k} \times (2k - n + 2) - 2$ **products and** $2^{n-k} \times (2k - n + 2) - n - 2$ **summations**.

computation of the APPs

Using (B.36) the computation of an APP value involves the same number of summations as for the extrinsic probability and twice the number of products.

summary

The previous results are summarized in Table B.1. In a first approximation, and as intuitively predictable, the complexity is proportional to the maximum number of states of the trellis, 2^{n-k}.

	Products number	Summations number
γs computation	$2n$	0
αs computation	$2^{n-k+1}(2k-n+2)-5$	$2^{n-k}(2k-n+1)-2$
βs computation	$2^{n-k+1}(2k-n+2)-5$	$2^{n-k}(2k-n+1)-2$
Extrinsic computation	$2^{n-k}(2k-n+2)-2$	$2^{n-k}(2k-n+2)-n-2$
APP computation	$2^{n-k+1}(2k-n+2)-4$	$2^{n-k}(2k-n+2)-n-2$
Intermediate Iteration	$5\times 2^{n-k}(2k-n+2)$ $+2n-12$	$2^{n-k}(6k-3n+4)$ $-n-6$
Final Iteration	$6\times 2^{n-k}(2k-n+2)$ $+2n-14$	$2^{n-k}(6k-3n+4)$ $-n-6$

Table B.1: Complexity of the FB algorithm as a SISO decoder for a linear block code (n,k)

Appendix C

Publications

Iterative decoding convergence analysis :
– J. Boutros and O. Pothier : *Convergence analysis of turbo decoding,* Proceedings of the 5-th Canadian Workshop on Information Theory (CWIT'97), Toronto, June '3-6, 1997, pp. 25-28
– J. Boutros and O. Pothier : *Convergence analysis of turbo decoding,* Submitted to IEEE Transactions on Information Theory.

GLD codes :
– O. Pothier, L. Brunel and J. Boutros : *A Low Complexity FEC Scheme Based on the Intersection of Interleaved block Codes,* Proceedings of the Vehicular Technology Conference (VTC'99), Vancouver, June 1999.
– J. Boutros, O. Pothier and G. Zémor : *Generalized Low Density (Tanner) Codes,* Proceedings of the International Conference on Communication (ICC'99), Houston, July 1999.
– J. Boutros, O. Pothier and S. Vialle : *An asymptotical good family of Generalized Low Density codes,* in preparation for IEEE Transactions on Information Theory.

Algebraic interleavers construction :
– O. Pothier : *Algebraic Interleavers for Codes on Graphs,* Second Philips conference on Digital Signal Processing, Veldoven, November 1999.

Bibliography

[1] M. Abramowitz & I. Stegun : Handbook of mathematical functions, Dover Publications Inc, New York, ninth printing, 1972, p. 257.

[2] N. Alon : *Eigenvalues and expanders,* Combinatorica 6, 2, pp. 83-96, 1986

[3] L. R. Bahl, J. Cocke, F. Jelinek, J. Raviv : *Optimal decoding of linear codes for minimizing symbol error rate,* IEEE Trans. on Information Theory, vol. 20, pp. 284–287, March 1974.

[4] G. Battail, M.C. Decouvelaere : *Décodage par répliques,* Annales des Télécommunications, vol. 31 , no. 11-12, pp. 387-404.

[5] G. Battail, M.C. Decouvelaere, P. Godlewski : *Replication decoding,* IEEE Trans. on Information Theory, vol. IT-25, pp. 332-345, May 1979.

[6] G. Battail : *Description polynomiale des codes en blocs linéaires*, Annales des Télécommunications, vol. 38 , no. 1-2, pp. 3-15.

[7] G. Battail, J. Fang : *Décodage pondéré optimal des codes linéaires en blocs. II.– Analyse et résultats de simulation,* Annales des Télécommunications, vol. 41, no. 11-12, nov.-dec. 1986, pp. 1-25.

[8] G. Battail : *Pondération des symboles décodés par l'algorithme de Viterbi,* Annales des Télécommunications, vol. 42, no. 1-2, Janvier-Fevrier. 1987, pp. 1-8.

[9] G. Battail : *Polynomial description of linear block codes and its application to soft-input, soft-output decoding*, Annales des Télécommunications, vol. 54, no. 3-4, mars-avril 1999, pp. 148-165.

[10] G. Battail : *A Conceptual Framework for Understanding Turbo codes,* International Symposium on Turbo Codes, pp. 55-62, Brest, September 1997.

[11] G. Battail : Théorie de l'information, Masson, Paris, 1997

[12] C. M. Bender & S A. Orszag : Advanced Mathematical Methods for Scientists and Engineers, McGraw-Hill Book Company, New York, 1978, pp. 218ff.

[13] S. Benedetto, E. Biglieri, V. Castellani : Digital Transmission Theory, Prentice-Hall, Englewood Cliffs, New Jersey, 1987.

[14] S. Benedetto, G. Montorsi: *Unveiling turbo-codes: some results on parallel concatenated coding schemes,* IEEE Trans. on Information Theory, vol. 42, no. 2, pp. 409-429, March 1996.

[15] S. Benedetto, G. Montorsi: *Design of parallel concatenated convolutional codes,* IEEE Trans. on Communication, vol. 44, no. 5, pp. 591-600, May 1996.

[16] S. Benedetto, G. Montorsi, D. Divsalar, F. Pollara : *Serial concatenation of interleaved codes: Performance analysis, design and iterative decoding,* TDA Progress Report 42-126, JPL, April-June 1996.

[17] S. Benedetto, G. Montorsi : *Serial concatenation of block and convolutional codes,* Electronic Letters, Vol. 32, pp. 887-888, 1996.

[18] S. Benedetto, G. Montorsi : *Iterative Decoding of serially concatenation convolutional codes,* Electronic Letters, Vol. 32, pp. 1186-1188, 1996.

[19] C. T. Benson : *Minimal Regular graphs of Girths eight and twelve,* Canad. Journal Math., vol. 18, pp. 1091-1094, 1966.

[20] E .R. Berlekamp : *The technology of error-correcting codes,* Proc. IEEE, vol. 68, no. 8, May 1980, pp. 564-593.

[21] E. R. Berlekamp : Algebraic Coding Theory, Aegean Park Press, Revised 1984 Edition.

[22] C. Berrou, A. Glavieux, P. Thitimajshima : *Near Shannon limit error-correcting coding and decoding : turbo-codes,* Proceedings of ICC'93, Genève, pp. 1064-1070, May 1993.

[23] C. Berrou, C. Douillard, M. Jézéquel : *Multiple parallel concatenation of circular recursive systematic convolutional (CRSC) codes,* Annales des Télécommunications, vol. 54, no. 3-4, mars-avril 1999, pp. 166-172.

[24] N. Biggs :Algebraic Graph Theory, Cambridge Tracts in Mathematics, Cambrigre University Press, 1974.

[25] J. Boutros, O. Pothier : *Convergence analysis of turbo decoding,* Canadian Workshop on Information Theory, Toronto, June 1997.

[26] J. Boutros, O. Pothier, G. Zémor : *Generalized Low Density (Tanner) Codes* Proceedings of ICC'99, Vancouver, June 6-10, 1999.

[27] J. Boutros, S. Vialle : *Performance Limits of Concatenated Codes with Iterative Decoding,* submitted to ISIT'2000, Sorrento, Italy.

[28] L. Brunel : Algorithmes de décodage de canal pour l'accès multiple à étalement de spectre. Ph. D. Dissertation, ENST, Nov. 99.

[29] C. Brutel, J. Boutros : *Serial Concatenation of Interleaved Convolutional Codes and M-ary Continuous Phase Modulations*, Annales des Télécommunications, vol. 5, no. 3-4, mars-avril 1999, pp. 235-242.

[30] C. Cardinal, D. Haccoun, F. Gagnon, and N. Batani : *Turbo Decoding Using Convolutional Self Doubly Orthogonal Codes,* Proceeedings of ICC'99, Vancouver, June 6-10, 1999.

[31] D. Chase : *A class of algorithms for decoding block codes with channel measurements information*, IEEE Trans. on Information Theory, Vol. IT 18, Jan. 1972, pp. 170-182.

[32] F. De Clerck, J. A. Thas and H. Van Maldeghem : *Generalized Polygons and Semipartial Goemetries,* Euler Institute for Discrete Mathematics and its Applications (EIDMA) minicourse, Eindhoven 22-26 April 1996. Available online at `http://cage.rug.ac.be/~fdc/eidma.ps`

[33] P. Dembowski : Finite Geometries, Springer Verlag, New York, 1968.

[34] E. Diestel : Graph Theory, Graduate Texts in Mathematics, Springer-Verlag, 1997.

[35] D. Divsalar, S. Dolinar, R.J. McEliece, F. Pollara : *Transfer function bounds on the performance of turbo codes,* TDA Progress Report 42-122, JPL, April-June 1995.

[36] S. Dolinar and D. Divsalar : *Weight Distributions for Turbo Codes Using Random and Nonrandom Permutations,* TDQ Progress Report 42-122, JPL, August 1995. Available online at `http://tmo.jpl.nasa.org/tmo/progress_report/42-122/title.htm`

[37] T. M. Duman, and M. Salehi : *New Performance Bounds for Turbo Codes,* IEEE Trans. on Communications, vol. 46, No. 6, June 1998, pp. 717-723.

[38] P. Elias : *Error-free coding,* IRE Trans. on Information Theory, pp. 29-37, 1954.

[39] K. Engdahl and K. S. Zigangirov : *On Low-Density Parity-Check Convolutional Codes*, Proceedings of WCCC'99, Paris, pp. 379-392, January 1999.

[40] Z. Füredi, F. Lazebnik, À. Seress, V. A. Ustimenko and A. J. Woldar : *Graphs of Prescribed Girth and Bi-degree,* Journal of Combinatorial Theory, Series B 64, pp. 228-239, 1995.

[41] G. D. Forney : *Concatenated Codes,* Cambridge, MIT press, 1966.

[42] G. D. Forney: *The Viterbi algorithm,* Proceedings of the IEEE, vol. 61, no. 3, pp. 268-278, March 1973.

[43] M. P. C. Fossorier, S. Lin :*Soft-Input Soft-Output Decoding of Linear Block Codes Based on Ordered Statistics*, Proceedings of GLOBCOM'98, pp. 2828-2833.

[44] B. J. Frey and F. R. Kschischang : *Probability propagation and iterative decoding*, Allerton Conference, October 1996.

[45] B. J. Frey : *Pseudo-Random Codes Based on Bayesian Networks,* Proceedings of the 5th Canadian Workshop on Information Theory (CWIT), pp. 15-18, Toronto, june 3-6, 1997.

[46] R. G. Gallager : *Low-Density Parity-Check Codes,* IRE Trans. on Information Theory, January 1962, pp. 21-28.

[47] R. G. Gallager : Low-density parity-check codes, MIT Press, 1963. Also available online at `http://justice.mit.edu/people/pubs/ldpc.ps`

[48] R. G. Gallager: Information Theory and Reliable Communication, John Wiley & Sons, 1968.

[49] A. Glavieux, C. Laot, and J. Labat : *Turbo equalization over a frequency selective channel,* International Symposium on Turbo Codes, pp. 96-102, Brest, September 1997.

[50] P. S. Guinand and J. Lodge : *Tanner Type Codes Arrising from Large Girth Graphs,* Proceedings of the 5th Canadian Workshop on Information Theory, pp. 5-7, Toronto, June 1997.

[51] J. Hagenauer, P. Hoeher: *A Viterbi algorithm with soft-decision outputs and its applications,* Proceedings of GLOBECOM'89, Dallas, Texas, pp. 47.1.1–47.1.7, November 1989.

[52] J. Hagenauer, E. Offer, L. Papke: *Iterative decoding of binary block and convolutional codes,* IEEE Trans. on Information Theory, vol. 42, no. 2, March 1996, pp. 429-445.

[53] C. R. P. Hartmann and L. D. Rudolph : *An Optimum Symbol-by-Symbol Decoding Rule for Linear Codes,* IEEE Trans. on Information Theory, vol. IT-22, no. 5, Sept. 1976, pp. 514-517.

[54] H. Herzberg and G. Poltyrev :*The Error Probability of M-ary PSK Block Coded Modulation Schemes,* IEEE Trans. on Communications, vol. 44, no. 4, April 1996, pp. 427-433.

[55] P. Hoeher : *New iterative ("turbo") decoding algorithms,* International Symposium on Turbo Codes, pp. 63-70, Brest, September 1997.

[56] A. J. Hoffman and R. R. Singleton : *On Moore Graphs with Diameter 2 and 3,* IBM Journal Res. Develop., vol. 4, pp. 497-504, November 1960.

[57] J. Hokfelt and T. Maseng : *On the convergence rate of iterative decoding,* Proceedings of GLOBECOM'98, Sydney, Australia, November 1998. Also available online at `http://www.tde.lth.se/jht/pdfs/ct04p1.pdf`

[58] J. Hokfelt, Ove Edfors, and T. Maseng : *Turbo codes : correlated extrinsic information and its inpact on iterative decoding convergence,* Proceedings of VTC'99, Houston, 18-21 May, 1999.

[59] B. Honary, G. Markarian : Trellis Decoding of Block Codes : A Practical Approach, Kluwer International Series in Engineering and Computer Science, Secs 391, June 1997.

[60] N. Kahale and R. Urbanke : *On the Minimum Distance of Parallel and Serially Concatenated Codes,* submitted to Trans. on Information Theory, 1999.

[61] F. R. Kschischang and B. Frey :*Iterative Decoding of compound Codes by Probability in Graphical Models,* IEEE Journal on Selected Area in Communications (JSAC), Vol. 16, No. 2, February 1998, pp. 219-230.

[62] L. H. Charles Lee: Convolutional coding, fundamentals and applications, Artech House, 1997.

[63] M. Lentmaier and K. S. Zigangirov : *On Generalized Low-Density Parity-Check Codes Based on Hamming Component Codes,* IEEE Communication Letters, Vol. 3, No. 8, August 1999, p. 248ff.

[64] Y. Li, B. Vucetic, Y. Sato: *Optimum Soft-Output Detection for Channels with Intersymbol Interference,* IEEE Trans. on Information Theory, vol. 41, no. 3, May 1995.

[65] S. Lin, D. J. Costello : *Error Control Coding: Fundamentals and applications,* Englewood Cliffs, NJ, Prentice-Hall, 1983.

[66] S. Lin (Editor) : Trellises and Trellis-Based Decoding Algorithms for Linear Block Codes, Kluwer International Series in Engineering and Computer Science, Secs 443, March 1998.

[67] J. H. Lodge, P. Hoeher, and J. Hagenauer : *The decoding of multidimensional codes using separable MAP "filters",* Proceedings of the 16th Biennial Symp. Comm., Kingston, Canada, May 1992, pp. 343-346.

[68] J. H. Lodge, R. Young, P. Hoeher, and J. Hagenauer : *Separable MAP "filters"for the decoding of product and concatenated codes,* Proceedings of ICC'93, Genève, pp. 1064-1070, May 1993, pp. 1740-1745.

[69] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, D .A. Spielman : *Improved Low-Density Parity-Check Codes Using Irregular Graphs and Belief Propagation,* Proceedings of the International Symposium on Information Theory (ISIT'98), p. 117, August 16-21, 1998. Also available online at `http://www.icsi.berkeley.edu/ luby/PAPERS/belerr.ps`

[70] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, D .A. Spielman : *Analysis of Low Density Codes and Improved Designs Using Irregular Graphs,* Proceedings of the 30th

ACM STOC, May 23-26 1998. Also available online at `http://www.icsi.berkeley.edu/ luby/PAPERS/anaerr.ps`

[71] A. Lubotsky, R. Philips, and P. Sarnack : *Ramanujan Graphs,* COMBINATORICA, vol. 8(3), pp. 261-277, 1988.

[72] D. J. C. MacKay, S. T. Wilson, M. C. Davey *Comparison of Constructions of Irregular Gallager Codes,* To appear in IEEE Trans. on Communications, Submitted July 1998.

[73] D. J. C MacKay : *Good Error-Correcting Codes based on Very Sparse Matrices,* IEEE Trans. on Information Theory, 1999. Also available online at `http://wol.ra.phy.cam.ac.uk/mackay/mncN.ps.gz`.

[74] F.J. MacWilliams, N.J.A. Sloane: The theory of error-correcting codes, North-Holland, Englewood Cliffs, eighth impression, 1993.

[75] G. A. Margulis :*Explicit group-theoritical constructions of combinatorial schemes and their application to design of expanders and concentrators,* Problemy Peredachi Informatsii, vol. 24(1), pp. 51-60, 1988.

[76] R. J. McEliece, D. J. C. MacKay, and J.-F. Cheng :*Turbo decoding as an instance of Pearl's "belief propagation" algorithm,* IEEE Journal on Selected Areas in Communications (JSAC) Vol. 16, No. 2, pp.140-152, February 1998

[77] K. R. Narayanan and G. L. Stüber :*A serial Concatenation Approach to Iterative Demodulation and Decoding,* IEEE Trans. on Communications, Vol. 47, No. 7, pp. 956-961, July 1999.

[78] J. Pearl :*Fusion, propagation, and structuring in belief networks,* Artif. Intell., Vol. 29, pp. 241-288, 1986.

[79] W. W. Peterson and E. J. Weldon : Error-Correcting Codes, M.I.T. Press, Second edition, 1972.

[80] L.C. Perez, J. Seghers, D.J. Costello : *A distance spectrum interpretation of turbo codes,* IEEE Trans. on Information Theory, vol. 42, no. 6, pp. 1698-1709, November 1996.

[81] G. Poltyrev : *Bounds on the Decoding Error Probability of Binary Linear Codes Via their Spectra,* IEEE Trans. on Information Theory, vol. 40, no. 4, July 1994, pp. 1284-1292.

[82] O. Pothier, L. Brunel, J. Boutros : *A low complexity FEC scheme based on the intersection of interleaved block codes,* Proceedings of VTC'99, Houston, 18-21 May, 1999.

[83] J.G. Proakis: *Digital Communications,* New York, McGraw Hill, 3rd edition, 1995.

[84] R. Pyndiah, A. Glavieux, A. Picart, and S. Jacq : *Near Optimum product codes,* Proceedings of GLOBECOM'94, San Francisco, pp. 339-343,Nov. 1994.

[85] R. Pyndiah : *Iterative Decoding of Product Codes : Block Turbo Codes,* International Symposium on Turbo Codes, pp. 71-79, Brest, September 1997.

[86] R. Pyndiah : *Near optimum decoding of product codes : block turbo codes,* IEEE Transactions on Communications, vol. 46, no. 8, August 1998.

[87] T. Richardson, R. Urbanke : *The capacity of low-density parity check codes by probability propagation in graphical models,* Bell Labs report, submitted to IEEE Trans. on Information Theory. Available online at `http://http://lcavwww.epfl.ch/ ruediger/papers/capldpcc.ps`

[88] T. Richardson, A. Shokrollahi and R. Urbanke : *Design of Provably Good Low-Density Parity Check Codes,* submitted to IEEE Trans. on Information Theory. Available online at texttthttp://cm.bell-labs.com/cm/ms/who/tjr/papers/degree.ps

[89] P. Robertson, E. Villebrun, and P. Hoeher : *A comparison of optimal and sub-obtimal MAP decoding algorithms operating in the log-domain,* Proceedings of ICC'95, Seattle, Washington, June 1995, pp. 1009-1013.

[90] P. Robertson, P. Hoeher, and E. Villebrun : *Optimal and Sub-Optimal a Posteriori Algorithms Suitable for Turbo Decoding,* European Trans. on Telecommunications, vol. 8, no. 2, March-April 1997, pp. 119-125.

[91] G. Royle : Cages of higher valency. Available online at `http://www.cs.uwa.edu.au/ gordon/cages/allcages.html`

[92] I. Sason and S. Shamai (Shitz) : *Bounds in the error probability of ML decoding fo block and turbo-block codes,* Annales des Télécommunications, vol. 5, no. 3-4, mars-avril 1999, pp. 183-200.

[93] I. Sason and S. Shamai (Shitz) : *Improved Upper Bounds on the ML Decoding Error probability of Parallel and Serial Concatenated Turbo Codes via their Ensemble Distance Spectrum*, submitted to IEEE Trans. on Information Theory, march 1999.

[94] H. Schellwat : *Highly Expanding Graphs Obtained from Dihedral Groups,* DIMACS Series in Discrete Mathematics and Theoritical Computer Science, vol. 10, pp. 117-123, 1993.

[95] M. Sipser and D. Spielman : *Expander Codes,* IEEE Trans. on Information Theory, Vol. 42, no 6, pp. 1710-1722, November 1996.

[96] D. Spielman : *Linear-time encodable and decodable error-correcting codes,*, IEEE Trans. on Information Theory, Vol. 42, November 1996.

[97] R. M. Tanner : *A recursive approach to low complexity codes,* IEEE Trans. on Information Theory, Vol. IT-27, September 1981.

[98] S. Ten Brink : *Convergence of iterative decoding,* Electronic Letters, Vol. 35, No. 10, May 1999, pp. 806-808.

[99] J. A. Thas and G. Lunardon : *Finite Generalized Quadrangles,* Intensive Course on Galois Geometry and Generalized Polygons, 15-25 April 1998, University of Ghent. Available online at `http://cage.rug.ac.be/ fdc/intensivecourse/fgq.ps`.

[100] W. T. Tutte : Connectivity in graphs, University Press, Toronto, 1966.

[101] A. Vardy : Trellis Structure of Codes, in Handbookof Coding Theory, Ed. V. Pless, W. Cary Huffman and R. Brualdi, Elsevier Science Publishers, Amsterdam.

[102] S. Vialle, J. Boutros : *A Gallager-Tanner construction based on convolutional codes,* Proceedings of WCC'99, pp. 393-404, Paris, January 1999.

[103] A. J. Viterbi, J. K. Omura : Principles of digital communications and coding, McGraw-Hill, 1979.

[104] A. J. Viterbi, A. M. Viterbi, J. Nicolas, N. T. Sindhushayan : *Prespectives on interleaved concatenated codes with iterative soft-output decoding,* International Symposium on Turbo Codes, Brest, September 1997.

[105] N. Wiberg, H. A. Loeliger and R. Kötter : *Codes and iterative decoding on general graphs*, European Trans. on Telecom., Vol. 6, Sept/Oct 1995.

[106] N. Wiberg : Codes and Decoding on General Graphs, Ph.D. Thesis, Linköpings University, 1996.