HAL
open science

Optimisation conjointe de codes LDPC et de leurs architectures de décodage et mise en œuvre sur FPGA

Jean-Baptiste Doré

To cite this version:

Jean-Baptiste Doré. Optimisation conjointe de codes LDPC et de leurs architectures de décodage et mise en œuvre sur FPGA. Traitement du signal et de l'image [eess.SP]. INSA de Rennes, 2007. Français. NNT : . tel-00191155v2

HAL Id: tel-00191155

https://theses.hal.science/tel-00191155v2

Submitted on 12 May 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Thèse

présentée devant

l'INSA DE RENNES

pour obtenir le grade de

Docteur de l'INSA de Rennes

Spécialité : *Électronique*

par

Jean-Baptiste Doré

Équipe d'accueil : Laboratoire Broadband Wireless Access, France Telecom

École doctorale : Matisse

Composante universitaire : INSA de Rennes

Optimisation conjointe de codes LDPC et de leurs architectures de décodage et mise en oeuvre sur FPGA

Soutenue le 26 Octobre 2007 devant la commission d'Examen

Composition du jury

Rapporteurs

Emmanuel Boutillon	Professeur à l'université de Bretagne Sud
Marc Fossorier	Professeur à l'université de Hawaï, USA

Examinateurs

Ramesh Pyndiah	Professeur à l'ENST Bretagne (Directeur de thèse)
Claude Berrou	Professeur à l'ENST Bretagne (Président du jury)
Thierry Lestable	Docteur Ingénieur à Samsung Electronics Research Institute, UK
Pierre Pénard	Ingénieur à France Telecom (co-encadrant de thèse)
Marie-Hélène Hamon	Ingénieur à France Telecom (co-encadrant de thèse)
Maryline Hélard	Professeure à l'INSA de Rennes

Remerciements

Je remercie en tout premier lieu Marie-Hélène Hamon et Pierre Pénard pour m'avoir accompagné dans ce travail. Ces travaux sont issus d'un travail d'équipe. Je remercie également Ramesh Pyndiah et Frédéric Guilloud pour avoir encadré ce travail.

J'exprime ma gratitude envers Claude Berrou qui m'a fait l'honneur de présider le jury. Je remercie également Marc Fossorier et Emmanuel Boutillon d'avoir rapporté d'une manière détaillée et constructive ce travail. Merci également à Thierry Lestable pour avoir examiné ce travail. Enfin je tiens à remercier Maryline Hélard pour son aide précieuse dans l'organisation de la soutenance et pour avoir accepté d'y participer.

Je remercie tous ceux qui ont contribué à cette thèse par leurs conseils et remarques. En particulier je tiens à remercier l'ensemble des personnes de l'équipe IRI et BCM. J'aurais une pensée particulière à l'équipe des Hardeux qui ont largement contribué à l'intégration du travail sur la plateforme Valentinno. Merci également aux autres membres du laboratoire BWA avec qui j'ai pu passer de très bon moments.

Je n'oublie pas mes collègues et amis Valérian et Xénofon qui m'ont accompagné et conseillé dans mes débuts. J'ai également une pensée pour mes collègues de bureau du dernier semestre, Benoît et Laurent. Enfin, je tiens à remercier l'ensemble des thésards, CDDs, stagiaires et post-docs d'aujourd'hui et d'hier avec qui j'ai pu passer des moments agréables : Pierre-Jean, Iryna, Stéphane, Audrey, Rachid, Alexandre, Chrislin, Anis, Lin, Mamdou, Evangelos et Dominique. Je tiens également à remercier tout particulièrement Benoît et Vincent qui ont largement participé à l'étude et au développement de la partie Hardware à travers leur collaboration.

Enfin, je remercie Amélie, mes parents, mes frères et l'ensemble de mes amis pour m'avoir encouragé durant ces trois années pas toujours faciles...

Résumé

La découverte dans les années 90 des Turbo-codes et, plus généralement du principe itératif appliqué au traitement du signal, a révolutionné la manière d'appréhender un système de communications numériques. Cette avancée notable a permis la redécouverte des codes correcteurs d'erreurs inventés par R. Gallager en 1963, appelés codes Low Density Parity Check (LDPC). L'intégration des techniques de codage dites avancées, telles que les Turbo-codes et les codes LDPC, se généralise dans les standards de communications. Dans ce contexte, l'objectif de cette thèse est d'étudier de nouvelles structures de codage de type LDPC associées à des architectures de décodeurs alliant performances et flexibilité.

Dans un premier temps, une large présentation des codes LDPC est proposée incluant les notations et les outils algorithmiques indispensables à la compréhension. Cette introduction des codes LDPC souligne l'intérêt qu'il existe à concevoir conjointement le système de codage/décodage et les architectures matérielles. Dans cette optique, une famille de codes LDPC particulièrement intéressante est décrite. En particulier nous proposons des règles de construction de codes pour en contraindre le spectre des distances de Hamming. Ces contraintes sont intégrées dans la définition d'un nouvel algorithme de définition de codes travaillant sur une représentation compressée du code par un graphe.

Les propriétés structurelles du code sont ensuite exploitées pour définir l'algorithme de décodage. Cet algorithme, caractérisé par le fait qu'il considère une partie du code comme un code convolutif, converge plus rapidement que les algorithmes habituellement rencontrés tout en permettant une grande flexibilité en termes de rendements de codage. Différentes architectures de décodeurs sont alors décrites et discutées. Des contraintes sur les codes sont ensuite exposées pour exploiter pleinement les propriétés des architectures.

Dans un dernier temps, une des architectures proposées est évaluée par l'intégration d'un décodeur sur un composant programmable. Dans différents contextes, des mesures de performances et de complexité montrent l'intérêt de l'architecture proposée.

Mot-clés : Codes LDPC, Codage de canal, Décodage itératif, Architectures de décodage, Implémentation FPGA.

Abstract

The introduction of Turbo-codes in the early 90's and, more generally the iterative principle, has deeply modified the methods for the design of communication systems. This breakthrough has also resurrected the Low Density Parity Check (LDPC) codes invented by R. Gallager in 1963. Advanced channel coding techniques such as Turbo-codes and LDPC, are now increasingly considered for introduction into communication systems and standards. This evolution towards industrialization motivates the definition of new flexible and efficient decoding architecture for LDPC codes. In this thesis, we focus our research on the iterative decoding of LDPC codes and their hardware implementation.

We first introduce basic concepts and notations for LDPC codes, which are necessary for a good comprehension. This introduction underlines the interest of jointly designing codes, decoding algorithm and architecture. From this perspective, a family of LDPC codes is described. We define some design rules to constrain the distance spectrum of the code. These constraints are introduced into a new algorithm for the design of the code working on a compact representation of the code graph.

A new decoding algorithm is also defined, taking advantage of the intrinsic properties of the code structure. Convergence of the decoding algorithm is increased compared to classical decoding algorithm for LDPC codes. Performance and flexibility of this algorithm is discussed. Different architectures are then described and studied. Some constraints on the codes are derived to target an architecture.

The last part of the thesis illustrates the implementation of one of the architectures discussed into a field-programmable gate array (FPGA). Performance and complexity measures are presented for various contexts, showing the interest of the concept for all these cases.

Keywords : LDPC codes, Channel coding, Iterative decoder, Decoder architecture, FPGA implementation.

Table des matières

Table des matières **ix**

Introduction **xiii**

1 Introduction aux techniques de codage de canal avancées **1**
1.1 Le codage de canal 1
1.1.1 Concepts de base 1
1.1.2 Paramétrisation d'un code 2
1.1.3 Propriétés des codes linéaires 4
1.1.4 Mesure des performances d'un code 5
1.1.5 Exemple de codes linéaires 6
1.2 Principe turbo et Turbo-codes 7
1.2.1 Principe Turbo 7
1.2.2 Turbo-codes 8
1.3 Codes LDPC 10
1.3.1 Bref historique 10
1.3.2 Définitions et notations 10
1.3.3 Décodage des codes LDPC 13
1.3.4 Encodage des codes LDPC 24
1.3.5 Construction des codes LDPC 27
1.4 Réalisation de décodeurs LDPC 30
1.4.1 Méthodologie de conception 30
1.4.2 Éléments de réalisation 32
1.5 Conclusion 35

2 Codes LDPC structurés **37**
2.1 Étude d'une structure particulière 37
2.1.1 Motivations 37
2.1.2 Définition d'une structure 39
2.2 Analyse de la structure étudiée 45
2.2.1 Propriétés de distance 45
2.2.2 Analyse des cycles dans le graphe du code 53
2.2.3 Construction des codes : analyse expérimentale 62
2.3 Étude d'un algorithme de décodage adapté à la structure étudiée . . . 69

2.3.1 Algorithme Layered BP 69
2.3.2 Turbo BP 73
2.4 Conclusion 81

3 Étude théorique d'architectures pour les codes LDPC structurés 83
3.1 Motivations 83
3.2 Étude d'architectures associées à l'algorithme Layered BP 84
3.2.1 Architecture générique de décodage 84
3.2.2 Vers une architecture contrainte 87
3.3 Étude d'architectures associées à l'algorithme Turbo Layered BP . . . 98
3.3.1 Architecture de décodage 98
3.3.2 Mise en oeuvre de la généricité 107
3.4 Étude d'une architecture hybride 115
3.4.1 Structure des codes 115
3.4.2 Étude d'architecture de décodage 118
3.4.3 Synthèse 119
3.5 Conclusion 120

4 Éléments de réalisation d'un décodeur LDPC 121
4.1 Préambule 121
4.2 Étude détaillée de la réalisation du décodeur 122
4.2.1 Algorithme de décodage 122
4.2.2 Processeur de calculs 124
4.2.3 Quantifications 130
4.2.4 Gestion de la mémoire 142
4.2.5 Conclusion 143
4.3 Résultats expérimentaux 144
4.3.1 Mise en oeuvre de la chaîne de test 144
4.3.2 Considérations technologiques 145
4.3.3 Exemple d'application 154
4.4 Conclusion 162

Conclusions et Perspectives 163

Annexes 167

A Interprétation graphique des règles de construction pour le séquencement Turbo Layered BP 169

B Matrices équivalentes de structures non strictement bi-diagonales 173

C Fichier VHDL de configuration du décodeur 177

D Liste des contributions 179

Table des figures **181**

Liste des tableaux **189**

Bibliographie **191**

Introduction

Contexte

Shannon a montré en 1948 qu'il existe une limite au débit d'information transmis en présence de bruit, appelée la capacité du canal, mais n'a pas explicité les moyens de l'approcher. Même si le caractère asymptotique de cette limite ne laisse aucun espoir de l'atteindre, la communauté de la théorie de l'information a recherché comment s'en approcher au plus près. Après plus de 40 ans de recherche, où "des demi-vérités ont fini par devenir des dogmes"[1], Claude Berrou, Alain Glavieux et Punya Thitimajshima ont montré comment réussir à s'approcher de cette fameuse limite avec une complexité raisonnable [2]. Ils améliorent les meilleures performances obtenues à l'époque de presque 3 décibels (dB) par l'application d'un concept analogue à celui de la contre-réaction bien connu des électroniciens. Ce principe, baptisé "Principe Turbo"par analogie aux moteurs, consiste à ré-injecter à l'entrée du système une partie de l'information de sortie. Comme le souligne Gérard Battail dans [1], le gain obtenu est comparable à celui d'un athlète qui aurait amélioré le record du 100 mètres en le courant en moins de 5 secondes[(1)] ! Cette avancée a donc retenti comme un coup de tonnerre dans la communauté scientifique. La manière de concevoir et d'analyser un système a par la suite été révisée. Cette science si exacte qu'est la théorie de l'information a connu une de ses avancés majeures par des observations issues de simulations expérimentales.

Cette révolution a ouvert de nombreuses voies de recherche dans le domaine du codage correcteur d'erreurs et plus globalement dans les systèmes de communications numériques. Cette avancée a eu pour conséquence une re-découverte des codes de Gallager, qui dès 1963 ont été associés au principe de décodage par échange d'information [3]. Les travaux de Gallager sur ces codes, appelés aussi codes LDPC (Low Density Parity Check), n'avaient pas suscité à cette période d'engouement. Une raison communément admise pour expliquer cet oubli, est la difficulté pour l'époque de concevoir des circuits performants permettant de traiter les algorithmes décrits. En 1995, encouragé par le contexte qui a suivi la découverte du principe turbo, MacKay redécouvre les codes de Gallager. Par la suite de nombreux travaux se sont intéressés à ces deux familles de codes, Turbo-codes et codes LDPC, et plus généralement à

[(1)] Le Jamaïcain Asafa Powell a battu le record du monde du 100 mètres messieurs avec un chrono de 9"74 secondes, à la rencontre d'athlétisme de Rieti (Italie) en septembre 2007.

l'application du principe itératif dans un système de communication numérique.

Ces systèmes sont alors considérés dans la définition de nouveaux standards de communications. En 1999, dans le domaine des radiocommunications mobiles de 3ème génération, la norme européenne UMTS (Universal Mobile Telecommunications System) ainsi que son homologue américain CDMA2000 (Code Division Multiple Access) ont retenu les Turbo-codes pour les services de données à haut débit. Les codes LDPC ont fait leur apparition dans le monde industriel en 2003 par leur introduction dans le domaine de la diffusion vidéo numérique par satellite de 2ème génération, DVB-S2 (Digital Video Broadcasting - Satellite - Second Generation). La compétition entre ces deux familles de codes correcteurs d'erreurs motive la définition d'architectures toujours plus performantes. Dans ce contexte, l'étude d'architectures de décodeurs LDPC efficaces est un réel challenge. L'objet de cette thèse est donc d'étudier de nouvelles architectures de décodage pour les codes de type LDPC avec les contraintes qu'imposent les nouveaux standards de radio-communication en terme de débit et de complexité.

Organisation du document

Ce mémoire est composé de quatre chapitres. Le détail de chacun des chapitres est décrit ci-dessous.

Le **chapitre 1** est consacré à une brève introduction des concepts liés au codage de canal et aux techniques avancées, plus particulièrement les Turbo-codes et les codes LDPC. Après avoir introduit les notations et les enjeux du codage de canal, le principe turbo et les Turbo-codes sont présentés. Les codes LDPC sont ensuite introduits. Une synthèse des différents algorithmes de décodage est également proposée. L'importance de l'ordonnancement de décodage est mise en évidence. Puis, différentes méthodes permettant l'encodage des codes LDPC sont décrites. Cette analyse montre qu'une construction bien pensée du code permet un encodage simple et, souligne l'importance de la problématique de construction des codes. Un lien entre les structures de certaines familles de codes LDPC avec d'autres familles de codes telles que les codes convolutifs est mis en évidence. Enfin, les méthodologies de conception d'architectures de décodeurs sont discutées. Des premières conclusions sur la stratégie de conception de codes sont définies.

Le **chapitre 2** a pour objectif de présenter et d'argumenter nos choix sur la structure de codage retenue. Dans un premier temps, nous montrons pourquoi les codes LDPC structurés sont intéressants et offrent de bonnes performances. Dans une deuxième partie, la famille de codes étudiée est décrite précisément. Cette famille de codes LDPC, dont la structure est dérivée des codes Repeat Accumulate, est analysée en détail. En particulier, une analyse des poids de certains mots de code est proposée. Cette étude permet la description de règles de construction pour éviter la définition de mauvais codes. Dans le même esprit, la description d'un outil de détection de cycles

dans le graphe du code est proposée. A partir de cette analyse, un algorithme de construction est décrit. Les différents paramètres de cet algorithme sont par la suite décrits à travers l'exploitation de résultats expérimentaux. La dernière partie de ce chapitre s'attache à illustrer comment un algorithme/séquencement de décodage peut être conçu quand des informations *a priori* sur la structure de la matrice de contrôle de parité définissant le code sont disponibles. En particulier, un nouvel algorithme dénommé *Turbo Layered BP* est introduit. Cette stratégie de décodage utilise une partie de la structure de la matrice de contrôle de parité à son avantage.

Dans le **chapitre 3**, des études détaillées d'architectures sont présentées pour les algorithmes *Layered BP* et *Turbo Layered BP*. Pour toutes les architectures explorées, le débit théorique et les contraintes sur la description du code sont précisés. Les différentes stratégies étudiées sont des architectures séries ou parallèles avec ou sans contraintes de maximisation de l'activité des processeurs. Ce chapitre décrit également comment la problématique de flexibilité du décodeur a été adressée, notamment dans le but de définir un décodeur capable de traiter une large gamme de tailles et de rendements de codage. Une solution basée sur les propriétés de la matrice de contrôle de parité associée à un mot de code poinçonné est proposée. La mise en oeuvre de cette technique très peu complexe est également illustrée. Dans une dernière partie, quelques concepts et résultats sur des architectures hybrides sont présentées.

Le **chapitre 4** est, quant à lui, principalement consacré à la mise en oeuvre des résultats présentés dans les chapitres précédents. L'algorithme de décodage et les simplifications utilisées sont dans un premier temps décrits. La problématique de quantification des données d'entrées et des chemins de données internes au décodeur est ensuite abordée. Des méthodes de réduction de la dynamique des chemins internes du décodeur sont alors proposées. Sur la base de ces résultats, un exemple d'intégration du décodeur sur un composant programmable de type FPGA (Field Programmable Gate Array) est exposé. Une analyse de la complexité est réalisée ainsi que des mesures de performances dans différents contextes. La dernière partie de ce chapitre s'attache à comparer le décodeur réalisé avec un décodeur turbo duo-binaire 8 états. La complexité des décodeurs et les performances des codes sont alors discutés.

Finalement, la conclusion générale de ce travail synthétise les différentes idées présentées dans ce document. Les perspectives de travaux futurs sont également décrites.

Chapitre 1

Introduction aux techniques de codage de canal avancées

Résumé

Ce premier chapitre introduit les techniques avancées de codage de canal. Tout d'abord la notion de codage de canal et quelques notations seront introduites. Dans une deuxième partie le principe turbo, et plus particulièrement les Turbo-codes seront brièvement décrits. La troisième partie de ce chapitre présentera les codes Low Density Parity Check (LDPC). Les méthodes de construction, d'optimisation, d'encodage et de décodage seront discutées. Enfin, dans une dernière partie, un état de l'art sur les architectures matérielles mettant en oeuvre un schéma de codage de type LDPC sera proposé.

1.1 Le codage de canal

1.1.1 Concepts de base

Un système de communication numérique peut se décomposer selon le schéma présenté sur la figure 1.1. Nous faisons l'hypothèse que l'émetteur est constitué d'une source binaire dont les bits sont i.i.d (indépendamment et identiquement distribués). Ce message binaire est codé par un codeur de canal. Le codage de canal, appelé aussi codage détecteur/correcteur d'erreurs, consiste en l'introduction d'une redondance associée à l'information utile dans le message à transmettre. La redondance et l'information utile sont liés suivant une loi donnée. Le train binaire résultant est alors transmis sur un canal bruité. En réception, le décodeur de canal exploite la redondance produite par le codeur dans le but de détecter, puis de corriger les erreurs introduites lors de la transmission. L'ajout de redondance dans le message à transmettre entraîne une perte d'efficacité du système. En effet, les bits de redondance introduits ne véhiculent pas de l'information utile. Cependant, cette perte à mettre en balance avec le gain de qualité obtenu par l'utilisation du codage. La notion et le

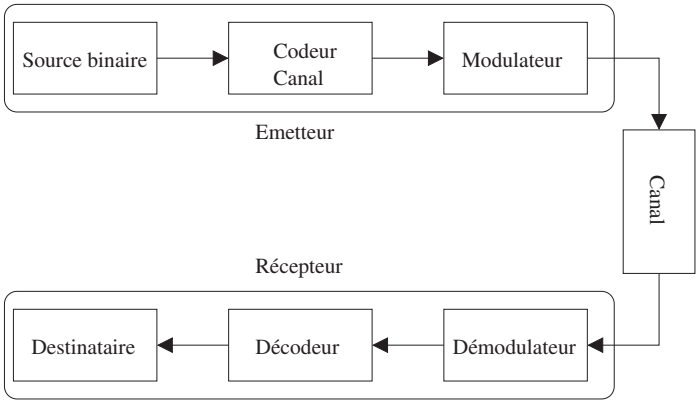


FIG. 1.1 – Modèle de communication numérique.

calcul de ce gain seront illustrés dans les paragraphes suivants.

On distingue deux familles de décodeurs travaillant à partir des données fournies par l'organe de démodulation. Le premier type de décodeur, dit *décodeur à décisions dures*, fonctionne à partir de données fermes ("0"ou "1") fournies par l'organe de démodulation. Le second type de décodeur est dit *décodeur à décisions pondérées*. Dans ce cas, l'organe de démodulation fournit au décodeur une valeur ferme accompagnée d'une mesure de fiabilité qui est exploitée par le décodeur.

1.1.2 Paramétrisation d'un code

Nous proposons d'introduire quelques notations de base qui seront par la suite largement utilisées. Tout d'abord nous considérons le schéma de codage de la figure 1.2. Dans l'ensemble des études proposées dans ce manuscrit, le codage canal est, par hypothèse, réalisé à partir de codes en blocs linéaires de taille N. Le codeur produit un *mot de code* de N bits à partir d'un *mot d'information* de K bits. Le code est linéaire si la fonction de codage est une fonction linéaire. Ainsi, la somme de deux mots de code forme un mot de code. On parle alors de stabilité sous l'addition. Le mot d'information peut se représenter sous la forme d'un vecteur $\underline{c}$ de dimension K. Le mot de code produit peut s'écrire sous la forme d'un vecteur de dimension N que nous appellerons $\underline{x}$. Ce code engendre donc $N - K = M$ bits de redondance que nous représenterons par le vecteur $\underline{p}$. Le code est *systématique* si les éléments de $\underline{c}$ sont inclus dans $\underline{x}$. On appelle *rendement de codage* R, le rapport entre le nombre de bits

FIG. 1.2 – Schéma simplifié d'un codeur de canal qui à partir d'un mot d'information $\underline{c}$ de K bits génère un mot de code $\underline{x}$ et $N = K + M$ bits.

d'information et le nombre de bits du mot de code transmis :

$$R = \frac{K}{N} \tag{1.1}$$

L'opération de codage correspond à l'opération matricielle suivante :

$$\underline{x} = \underline{c}\mathbf{G} \tag{1.2}$$

où la matrice $\mathbf{G}$ est appelée *matrice génératrice* du code. Cette matrice de taille $K \times N$ et de rang plein comporte N colonnes et K lignes. Sous une forme systématique cette matrice peut s'écrire :

$$\mathbf{G} = [\mathbf{I}\ \mathbf{R}] \tag{1.3}$$

où $\mathbf{I}$ est une matrice identité de taille $K \times K$. Le code peut aussi être caractérisé par sa *matrice de contrôle de parité* $\mathbf{H}$ de taille $M \times N$ définie par :

$$\mathbf{G}\mathbf{H}^t = \mathbf{0} \tag{1.4}$$

où $\mathbf{0}$ est une matrice nulle. En manipulant cette relation et l'équation 1.2, on démontre la relation de contrôle de parité suivante :

$$\mathbf{H}\underline{x}^t = \underline{0}^t \tag{1.5}$$

On appelle *syndrome* le vecteur $\underline{s}$ de dimension M défini par :

$$\underline{s}^t = \mathbf{H}\underline{x}^t \tag{1.6}$$

Si $\underline{s}$ est un vecteur nul alors $\underline{x}$ est un mot de code, sinon le vecteur $\underline{x}$ contient des bits erronés. Il est possible de détecter la présence d'erreurs dans un mot de code par le calcul du syndrome.

□ *Exemple:* Soit un code de rendement $R = 4/7$ défini par la matrice de contrôle de parité $\mathbf{H}$ suivante :

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Une matrice génératrice $\mathbf{G}$ de ce code peut s'écrire :

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Soit un mot d'information $\underline{c}$ de $K = 4$ bits à transmettre. En utilisant la matrice génératrice $\mathbf{G}$, on peut déduire le mot de code associé au mot d'information définit

par $\underline{c} = [1\ 0\ 0\ 0]$:

$$\begin{aligned}
\underline{x} &= \underline{c}\mathbf{G} \\
&= \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}
\end{aligned}$$

On considère la transmission de ce mot de code à travers un canal. Au niveau du récepteur, avant le décodeur canal, la séquence binaire estimée $\underline{\hat{x}}$ est :

$$\underline{\hat{x}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Une erreur a affectée le premier bit. Le syndrome associé à cette séquence estimé est donc :

$$\begin{aligned}
\underline{s}^t &= \mathbf{H}\underline{\hat{x}}^t \\
&= \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}^t \\
&= \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}^t
\end{aligned}$$

Le syndrome n'étant pas nul, l'erreur a pu être détectée. □

1.1.3 Propriétés des codes linéaires

On appelle *distance de Hamming* entre deux vecteurs $\underline{x}$ et $\underline{y}$, le nombre de positions où ces deux vecteurs sont différents. On note cette distance $d_H\left(\underline{x}, \underline{y}\right)$. Le *poids de Hamming* d'un vecteur $\underline{x}$, noté $w_H\left(\underline{x}\right)$, est égal au nombre de composantes non nulles. On peut donc aussi exprimer le poids de Hamming comme la distance de Hamming entre $\underline{x}$ et un vecteur nul. Par définition, on appelle w_i le plus petit poids d'un mot de code généré par un mot d'information de poids i. On peut noter que w_1 correspond au plus petit nombre d'éléments non nuls par ligne de la matrice génératrice $\mathbf{G}$[(1)].

La capacité de correction d'un code est mesurable[(2)] si on a connaissance de la *distance minimale* du code notée $d_{\min}$. La distance minimale d'un code est la plus petite distance de Hamming entre deux mots de codes. Comme nous nous intéressons à des codes linéaires (et donc la différence entre deux mots de code est également un mot de code), la distance minimale entre deux mots de code distincts est égale au

[(1)] Cette propriété vient du fait qu'un mot de code est une combinaison linéaire des lignes de $\mathbf{G}$

[(2)] Cette capacité de correction peut être mesurée si par exemple on considère un décodage à maximum de vraisemblance. Ce décodage consiste à décoder le mot reçu par le mot de code le plus proche au sens de la distance de Hamming

plus petit poids de Hamming d'un mot de code non nul. On peut donc exprimer la distance minimale de la façon suivante :

$$d_{\min} = \min_{\forall i \in [1,K]} w_i \tag{1.7}$$

On note $a(i)$, le nombre de mots de code de poids i. Nous définissons le *spectre de distance* d'un code par l'ensemble $\{a(i), i \geq 1\}$. A partir du spectre de distance on peut définir la *fonction énumératrice* des poids (Weight Enumerating Function WEF) donnée par :

$$A(z) = \sum_{l \geq 1} a(l) z^l \tag{1.8}$$

où z est une variable formelle. Cet outil est très utile pour calculer des bornes relatives à la probabilité d'erreur binaire trame, (Frame Error Rate) (FER), pour le canal à bruit additif blanc gaussien (AWGN : Additive White Gaussian Noise) dans la région de rapport signal sur bruit élevé, et sous l'hypothèse d'un décodage à maximum de vraisemblance [4] :

$$\mathrm{FER} \leq \frac{1}{2} \mathrm{A(z)}\Big|_{\mathrm{z^l = erfc}\left(\sqrt{\mathrm{Rl}\frac{\mathrm{E_b}}{\mathrm{N_0}}}\right)} \tag{1.9}$$

où E_b/N_0 est le rapport de l'énergie par bit sur la densité spectrale de puissance du bruit.

1.1.4 Mesure des performances d'un code

L'ajout de redondance dans le message à transmettre entraîne une perte d'efficacité du système. En effet les bits de redondance introduits ne véhiculent pas de l'information utile. Différents systèmes sont comparés en analysant la probabilité d'erreur bit ou paquet en fonction de l'énergie transmise par bit utile. On appelle *gain de codage* l'écart d'énergie par bit utile entre deux systèmes pour un taux d'erreur donné.

Dans le cas de l'utilisation de techniques de codage avancées, l'évolution de la performance du code peut se diviser en trois régions comme illustré sur la figure 1.3. La première région correspond à un comportement où le décodage ne converge pas. Dans certain cas, le décodage dégrade les performances par rapport à un système non codé : on parle de région de non convergence. A partir d'un certain rapport signal à bruit, appelé seuil de convergence, le décodage rentre dans une phase où la probabilité d'erreur diminue très rapidement avec le rapport signal à bruit : on parle de la région du *waterfall*. Enfin, il existe une région où la probabilité d'erreur diminue de manière moins rapide que le région du *waterfall*. Ce comportement est spécifique de la région du "plancher d'erreur"ou *error floor*. La dégradation des performances est due à de nombreux facteurs dont celui de la distance minimale du code. Dans le cas où l'effet de la distance minimale est prépondérant, la courbe de probabilité d'erreur binaire trame vient tangenter la borne décrite équation (1.9).

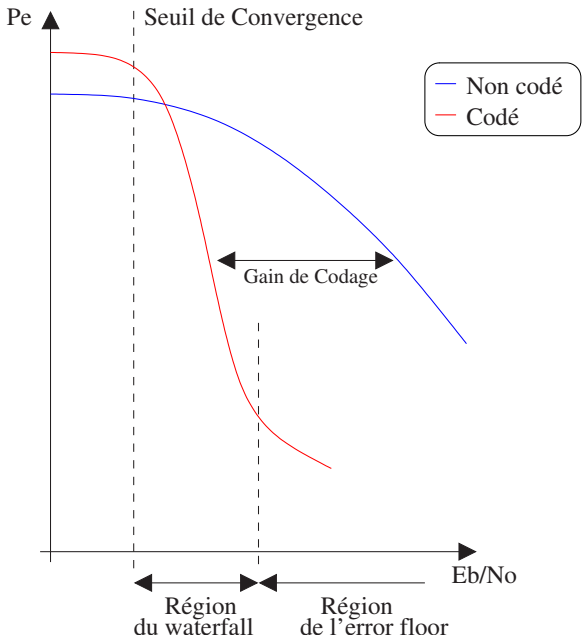


FIG. 1.3 – Illustration des trois régions caractérisant les performances d'un système de codage.

1.1.5 Exemple de codes linéaires

La famille de codes linéaires sans doute la plus connue, est la famille des codes convolutifs. Ces codes inventés en 1954 par P.Elias [5] constituent une famille de codes dont la simplicité et les bonnes performances sont en grande partie à l'origine de leur succès. Leur décodage se réalise très facilement en utilisant l'algorithme de Viterbi [6]. Cet algorithme parcourt le diagramme en treillis du code et détermine le chemin le plus vraisemblable. Ce diagramme en treillis représente l'évolution de l'état du codeur en fonction du temps. Les codes convolutifs sont souvent associés à un code externe formant un code concaténé puissant avec un fort pouvoir de correction. Ces schémas de codes concaténés introduits par Elias et Forney [5][7] ont une complexité de décodage raisonnable obtenue grâce à un décodage séquentiel des codes constitutifs. Les schémas de concatenation les plus connus sont sans doute la concatenation d'un code convolutif avec un code BCH (du nom de leurs auteurs Bose Chaudhuri et Hocquenghem)[8] dit réciproquement, code interne et code externe (exemple figure 1.4). En particulier les codes RS [9](Reed-Solomon)[(3)] sont très largement utilisés dans nombreux standards (DVB-T [10] par exemple). Ces codes RS sont des codes correcteurs très puissants dans lequel les bits du mot de code sont remplacés par un symbole d'un corps de Galois (en pratique on se restreint à GF(2^q)). Une des propriétés intéressantes des codes RS est l'expression analytique de la distance minimale qui est égale à $N - K + 1$.

[(3)] qui peuvent être présentés comme des codes BCH

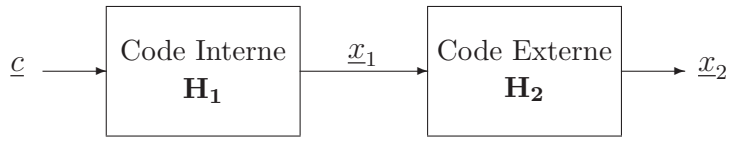


FIG. 1.4 – Concatenation série d'un code interne et d'un code externe.

D'autres familles de codes sont très largement utilisées et non détaillées dans ce manuscrit. Nous citerons par exemple les codes CRC (Cyclic Redundancy Code) qui permettent de vérifier l'intégrité d'un mot formé de plusieurs bits, ou encore les codes de Hamming.

Cette première partie illustre succinctement quelques principes de base nécessaires à la bonne compréhension du document.La suite de ce chapitre a pour but d'introduire les techniques de codage de canal dites avancées et notamment les Turbo-codes et les codes LDPC.

1.2 Principe turbo et Turbo-codes

1.2.1 Principe Turbo

"L'invention des Turbo-codes ne découle pas d'une théorie linéaire et limpide et encore moins d'un beau développement mathématique. Elle est le fruit d'un long tâtonnement [...]*."*[1]. Cette phrase résume l'état d'esprit grâce auquel le principe turbo a vu le jour. La concaténation de deux codes (par exemple deux codes convolutifs) est un moyen simple d'obtenir des distances élevées. Cependant les performances à faible rapport signal à bruit sont dégradées du fait de la répartition de l'énergie de la redondance entre les différents codes constituants. En effet, dans un schéma classique de récepteur où des décodeurs sont concaténés, l'exploitation de l'information n'est pas optimale. Plus généralement, un décodeur composé de sous-décodeurs optimaux ne forment pas un système optimal. Dans le cas d'une concatenation d'un code interne et externe, le décodeur externe ne bénéficie que de l'information contenue dans les symboles de redondance qui lui sont associés. Le second décodeur bénéficie quant à lui des symboles de redondance et du travail du décodeur externe qui le précède. Cette dissymétrie suggère de ré-injecter une information issue du deuxième décodeur dans le premier décodeur. Cette ré-injection de la sortie vers l'entrée est analogue au principe du moteur turbo. Ce principe a été proposé par C. Berrou, A Glavieux et P. Thitimajshima, en 1993 [2] et a a été rendu possible grâce aux travaux de G. Battail [11], J. Hagenauer et P. Hoeher [12] sur le décodage à sorties pondérées. Le principe turbo, initialement introduit pour le codage canal, a été ensuite étendu à l'ensemble

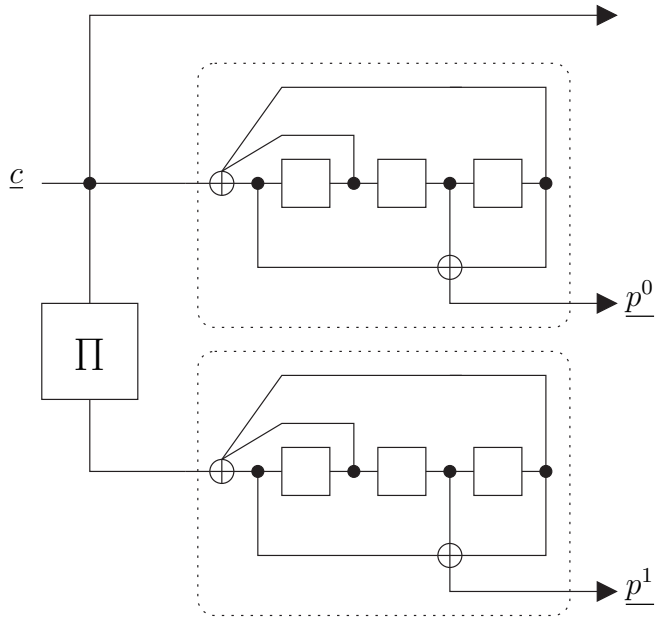


FIG. 1.5 – Schéma de principe d'un Turbo-code, construit à partir de deux codes convolutifs récursifs et d'un entrelaceur Π.

de la chaîne de reception. Ainsi, on parle de turbo synchronisation, turbo estimation de canal, turbo égalisation et plus généralement de récepteurs itératifs.

1.2.2 Turbo-codes

Le principe du Turbo-code fut introduit par C. Berrou, A Glavieux et P. Thitimajshima, en 1993 [2]. Pour la première fois, un code correcteur d'erreurs fonctionnant à moins de 0.5 dB de la limite de Shannon fut démontré. Cette rupture technologique dans le domaine du codage de canal a tout d'abord surpris la communauté scientifique, mais les résultats ont été très rapidement confirmés. La solution proposée, illustrée figure 1.5, consiste en une concaténation parallèle de deux codes convolutifs récursifs systématiques identiques au travers d'un entrelaceur aléatoire. Ainsi, à un mot d'information $\underline{c}$ est associé une redondance $\underline{p}$, qui peut être divisée en une redondance $\underline{p}^0$ issue du codage du code de la première dimension et une redondance $\underline{p}^1$ issue du codage de la deuxième dimension.

L'élément innovant d'une telle concaténation n'est pas tant dans la structure du code que dans le principe de décodage. Pour la première fois, un décodeur itératif est introduit. L'idée, très simple en soi, consiste en un décodeur comportant deux sous-ensembles de décodage s'échangeant de l'information. Le principe de ce récepteur est illustré sur la figure 1.6. Pour expliquer le fonctionnement d'un tel décodeur, la notion d'information extrinsèque fut introduite. Cette information associée à un sym-

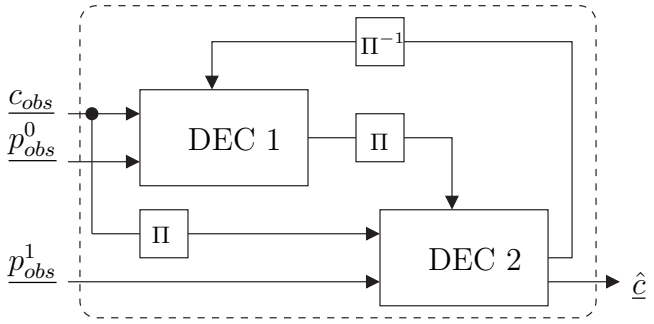


FIG. 1.6 – Schéma de principe d'un turbo décodeur. La notation Π représente la fonction d'entrelacement et Π^{-1} le dé-entrelacement. A partir des symboles reçus $\underline{c_{obs}}$ et $\underline{p^0_{obs}}$ le premier décodeur (DEC 1) calcule une information extrinsèque qui est échangée avec le décodeur 2 (DEC 2). De la même façon, le décodeur 2 calcule à partir des observations du canal $\underline{c_{obs}}$ entrelacées et $\underline{p^1_{obs}}$ une nouvelle information extrinsèque accompagnée d'une estimation des bits d'information. Ce processus peut être itéré autant de fois que le nombre maximum d'itérations choisi.

bole[(4)] est l'information apportée par le décodage du lien entre le symbole considéré et l'ensemble des autres symboles. C'est cette information qui est échangée entre les décodeurs au cours des itérations. Après un certain nombre d'itérations, la décision ferme est prise sur l'information *a posteriori* : cette information regroupe à la fois l'information issue de l'observation du canal et les informations extrinsèques issues des différents décodeurs.

Un Turbo-code est caractérisé par ces codes constituants et la fonction d'entrelacement. Tout d'abord, dans le cas des Turbo-codes parallèles, au moins l'un des codes constituants doit être un code convolutif récursif. En pratique les codes constituants sont choisis identiques. La fonction d'entrelacement permet d'introduire une fonction d'aléa entre les decodeurs. Ainsi, plus l'effet de brassage sera important, plus les informations extrinsèques seront décorrélées, ce qui améliorera la qualité du décodage. Cette fonction a aussi un role important sur la propriété de distance minimale du code [13].

Les Turbo-codes ainsi présentés ont ouvert de nombreuses voies de recherche dans le domaine du codage de canal. On peut citer par exemple les Turbo-codes en bloc [14] où les Turbo-codes séries [15]... Ce bouleversement dans la façon de concevoir un système de décodage a également permis de redécouvrir les travaux de Gallager sur les codes LDPC [3].

(4) la définition du symbole ici est une mesure probabiliste de la confiance sur un bit.

1.3 Codes LDPC

1.3.1 Bref historique

Les codes LDPC (Low Density Parity Check) ont été inventés par Gallager en 1962 [3]. Ces codes sont basés sur des matrices de contrôle de parité pseudo aléatoires de faible densité. Du fait de leur complexité d'encodage, de décodage et des moyens matériels de l'époque, ces codes n'ont pas suscité suffisamment d'intérêt au sein de la communauté de la théorie du codage. Cet oubli durera jusqu'à l'introduction des Turbo-codes et du principe itératif. Ainsi en 1995, MacKay redécouvre les codes LDPC [16]. Il montre tout d'abord que l'algorithme de décodage développé par Gallager, peut également se décrire comme l'algorithme de propagation de croyance (Belief Propagation (BP)) de Pearl [17]. Par la suite Luby, introduit les codes LDPC irréguliers [18] caractérisés par une matrice de contrôle de parité pour laquelle la distribution des nombres d'éléments non nuls par ligne et/ou colonne n'est pas uniforme. Par la suite, une des contributions majeures depuis la redécouverte des codes LDPC a été la description d'outils mathématiques pour l'analyse asymptotique des codes. Il a tout d'abord été montré que le comportement asymptotique des codes était fonction de ses paramètres. Ainsi grâce aux outils proposés dans [19], il est possible d'optimiser un ensemble de paramètres d'un code LDPC de façon à minimiser le seuil de convergence du code[(5)]. Cette méthode permettant de prédire la convergence du décodage en fonction des paramètres des codes LDPC est appelée *évolution de densité* (Density Evolution)[19].

La relative simplicité de l'analyse théorique des codes (du moins par rapport à la complexité d'analyse des Turbo-codes) a motivé la communauté scientifique à poursuivre les travaux dans le domaine des codes LDPC. Cette technologie de codage est restée relativement ouverte en termes de propriété intellectuelle, du fait que l'invention des codes LDPC remonte à 1962, ce qui a motivé les industriels à explorer cette voie. Ainsi en 2004, un code LDPC a été pour la première fois normalisé dans un contexte de diffusion par satellite : DVB-S2 [20]. Plus récemment, les codes LDPC ont été introduits dans les standards IEEE 802.16e [21] (Wimax mobile) et IEEE 802.11n [22] (Wifi). Ces différents facteurs, ainsi que les nouveaux problèmes posés, ont contribué à accroître la popularité des codes LDPC dans le domaine industriel et scientifique.

1.3.2 Définitions et notations

Un code LPDC est un code dont la matrice de contrôle de parité $\mathbf{H}$ est de faible densité. Ainsi le nombre de "1" dans la matrice est faible devant le nombre de "0". On rappelle que la relation de contrôle de parité entre un mot de code et une matrice de

[(5)] on appelle seuil de convergence la plus petite valeur de rapport signal à bruit tel que le décodage d'un mot de code soit réalisé avec une probabilité d'erreur nulle.

contrôle de parité est définie par :

$$\mathbf{H}\underline{x}^t = \underline{0}^t$$

La matrice de contrôle de parité $\mathbf{H}$ est de dimension $M \times N$ définissant donc un code en bloc où le nombre de bits d'information est $K = N - M$.

□ *Exemple:* On considère la matrice de contrôle de parité suivante d'un code de rendement 1/2 et produisant 4 bits de redondance :

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Les équations de parité associées à cette matrice et à un mot de code $\underline{x}= [x_0, \cdots, x_7]$ sont :

$$\begin{aligned} x_0 + x_4 &= 0 \\ x_1 + x_4 + x_5 &= 0 \\ x_2 + x_5 + x_6 &= 0 \\ x_3 + x_6 + x_7 &= 0 \end{aligned}$$

□

Un code LDPC peut également être représenté, en plus de sa matrice de contrôle de parité, sous une forme graphique. Cette représentation est appelée *graphe de Tanner* [23], ou plus généralement *graphe factoriel*. Un graphe factoriel contient deux types de noeuds, les *noeuds de données* et les *noeuds fonctionnels*. Deux noeuds sont reliés par une *branche*. Dans le cas des codes LDPC, les noeuds de données représentent le mot de code et les noeuds fonctionnels correspondent aux contraintes de parité [(6)]. Nous appellerons donc les noeuds fonctionnels, *noeud de contrôle de parité*. Un noeud de données i est relié à un noeud de contrôle j par une branche, si et seulement si, l'élément correspondant à la $j^{\text{ème}}$ ligne et la $i^{\text{ème}}$ colonne de la matrice de contrôle de parité est non nul. Par convention, les noeuds de données seront représentés par des cercles et les noeuds de contrôle par des carrés. Un noeud de données correspondant à un bit du mot de code transmis sera représenté par un cercle blanc. Si un bit du mot de code n'est pas transmis (on parle de bit poinçonné), le noeud sera représenté par un cercle plein noir et appelé noeud poinçonné ou noeud caché.

□ *Exemple:* La figure 1.7 représente le graphe de la matrice de contrôle de parité définie dans l'exemple précédent. Le mot de code transmis est le vecteur $\underline{x}$ où le bit x_6 a été poinçonné. □

[(6)] Les noeuds fonctionnels peuvent représenter d'autres contraintes. Dans l'exemple des codes TLDPC (Tail-biting Trellis Low-Density Parity Check) [24] les contraintes de parité sont remplacées par un seul treillis circulaire à deux états. Les noeuds fonctionnels représentent dans ce cas le treillis

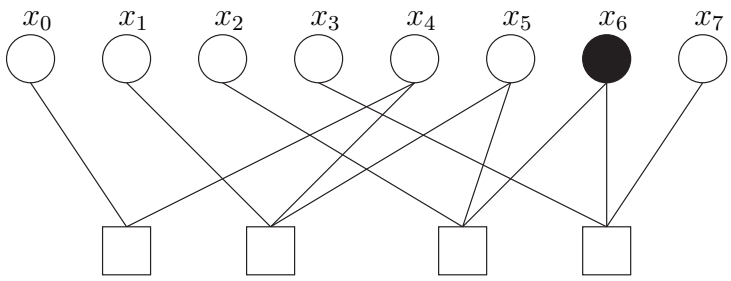


FIG. 1.7 – Graphe factoriel d'un code LDPC.

Le graphe factoriel est une représentation graphique très simple du code. Ce graphe permet notamment d'illustrer les algorithmes de décodage associés aux codes LDPC qui seront présentés par la suite.

Une famille de codes LDPC peut se décrire en terme du taux de connexions des équations de parité. Quand le nombre de "1"par ligne et le nombre de "1"par colonne est constant, on parle de code LDPC *régulier*. Par conséquent chaque bit du mot de code participe à un même nombre d'équations de parité. De même, chacune des équations de parité utilise le même nombre de bits. Par extension, les codes LDPC *irréguliers* sont des codes définis par des matrices de contrôle de parité où le nombre de "1"par ligne et/ou par colonne n'est pas constant. Pour décrire ces codes, il est d'usage de spécifier l'irrégularité d'un code à travers deux polynômes $\lambda(x)$ et $\rho(x)$:

$$\lambda(x) = \sum_{i\geq 1} \lambda_i x^{i-1} \tag{1.10}$$

$$\rho(x) = \sum_{i\geq 2} \rho_i x^{i-1} \tag{1.11}$$

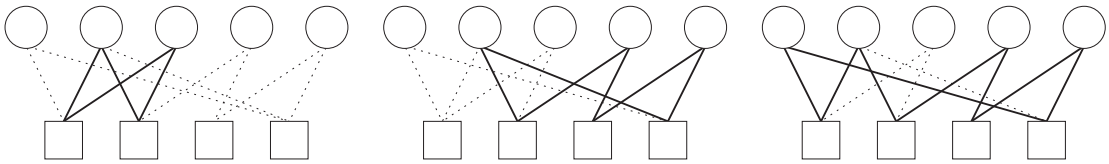
où λ_i (resp. ρ_i) caractérise la proportion du nombre de branches connectées aux noeuds de données (resp. noeuds de contrôle) de degré i par rapport au nombre total de branches. Le *degré* est défini comme le nombre de branches connectées à un noeud. On peut relier le profil d'irrégularité du code au rendement de codage de la façon suivante :

$$R \geq 1 - \frac{\sum_{i\geq 1} \frac{\lambda_i}{i}}{\sum_{i\geq 2} \frac{\rho_i}{i}} \tag{1.12}$$

Il y a égalité quand la matrice de contrôle de parité est de rang plein. On peut aussi caractériser l'irrégularité d'un code à travers deux autres polynômes, $\tilde{\lambda}(x)$ et $\tilde{\rho}(x)$ qui caratérise la proportion de noeuds de même degré (plutôt que la proportion de branches) de la manière suivante :

$$\tilde{\lambda}(x) = \sum_{i\geq 1} \tilde{\lambda}_i x^{i-1} \tag{1.13}$$

$$\tilde{\rho}(x) = \sum_{i\geq 2} \tilde{\rho}_i x^{i-1} \tag{1.14}$$



$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

FIG. 1.8 – Exemple de cycles de longueur 4, 6 et 8

Les coefficients $\tilde{\lambda}_i$ et $\tilde{\rho}_i$ de ces polynômes représentent la proportion de noeuds de données et de contrôles de degré i. Les deux représentations polynômiales sont reliées par :

$$\tilde{\lambda}_i = \frac{1}{i}\frac{\lambda_i}{\sum_{i\geq 1}\lambda_i/i} \tag{1.15}$$

$$\tilde{\rho}_i = \frac{1}{i}\frac{\rho_i}{\sum_{i\geq 2}\rho_i/i} \tag{1.16}$$

□ *Exemple:* le code représenté figure 1.7 a une distribution des degrés égale à :

$$\begin{aligned} \tilde{\lambda}(x) &= \frac{5}{8} + \frac{3}{8}x \\ \tilde{\rho}(x) &= \frac{1}{4}x + \frac{3}{4}x^2 \end{aligned}$$

qui peut aussi être exprimée de la façon suivante :

$$\begin{aligned} \lambda(x) &= \frac{5}{11} + \frac{6}{11}x \\ \rho(x) &= \frac{2}{11}x + \frac{9}{11}x^2 \end{aligned}$$

□

La représentation par graphe d'un code LDPC nous permet d'introduire la notion de *cycle*. Un cycle existe dans un graphe dès lors qu'il y a un chemin pour quitter et revenir à un noeud sans passer par les mêmes branches. Le nombre de branches traversées détermine la longueur du cycle. Un graphe sans cycle est appelé un arbre. La figure 1.8 représente des cycles de longueurs différentes. La position des éléments non nuls dans la matrice de contrôle de parité est également représentée. L'influence de telles configurations dans le graphe du code sera discutée par la suite.

1.3.3 Décodage des codes LDPC

Dans cette section, nous nous intéressons au décodage itératif souple des codes LDPC. L'algorithme de décodage itératif présenté initialement par Gallager [3], revu

ensuite par MacKay[16] dans le cadre de la théorie des graphes, est connu sous le nom d'algorithme de propagation de croyance[7] (Belief Propagation (BP)). Cet algorithme peut être vu comme un algorithme d'échange d'information entre les noeuds du graphe à travers les branches. Ces messages transitant de noeuds en noeuds portent une information probabiliste sur l'état des noeuds.

Le principe de la propagation de croyance est l'application directe de la règle de Bayes sur chaque bit d'une équation de parité. La vérification de parité permet de calculer une estimation de chaque bit. Ces estimations, formant des messages se propageant sur les branches du graphe, sont alors échangées itérativement afin de calculer une information *a posteriori* sur chaque bit. Dans le cas d'une propagation de croyance sur un graphe sans cycle, les messages échangés sont indépendants, ce qui conduit au calcul simple et exact des probabilités *a posteriori* : l'algorithme est dans ce cas optimal. Dans le cas des codes LDPC, le graphe factoriel présente des cycles. Dans ces conditions, l'hypothèse de messages indépendants n'est plus valide. Cependant, plus le graphe est creux (c'est à dire moins la matrice de contrôle de parité est dense), plus l'approximation d'un graphe sans cycle devient valide. C'est donc sous cette hypothèse que l'algorithme de décodage est décrit.

Si on considère une équation de parité c faisant intervenir un ensemble de bits V_c, la règle de Bayes appliquée au bit v permet d'exprimer les probabilités suivantes conditionnellement à la séquence reçue $\{y\}$:

$$Pr(v = 0|\{y\}) = Pr(\sum_{v' \in V_c/v} v' = 0|\{y\}) \tag{1.17}$$

$$Pr(v = 1|\{y\}) = Pr(\sum_{v' \in V_c/v} v' = 1|\{y\}) \tag{1.18}$$

où la somme est réalisée modulo 2. Gallager a démontré dans [3] que ces deux probabilités sont égales à :

$$Pr(v = 0|\{y\}) = \frac{1 + \prod_{v' \in V_c/v} (1 - 2Pr(v' = 1|\{y\}))}{2} \tag{1.19}$$

$$Pr(v = 1|\{y\}) = \frac{1 - \prod_{v' \in V_c/v} (1 - 2Pr(v' = 1|\{y\}))}{2} \tag{1.20}$$

En utilisant la relation :

$$\tanh\left(\frac{1}{2}\ln\frac{1 - Pr(v' = 1|\{y\})}{Pr(v' = 1|\{y\})}\right) = 1 - 2Pr(v' = 1|\{y\}) \tag{1.21}$$

[7] ce terme est issu de la communauté de l'intelligence artificielle et plus particulièrement des travaux de Pearl

on peut calculer le rapport de vraisemblance suivant :

$$\frac{Pr(v=0|\{y\})}{Pr(v=1|\{y\})} = \frac{1+\prod\limits_{v'\in V_c/v} \tanh\left(\frac{1}{2}\ln\frac{Pr(v'=0|\{y\})}{Pr(v'=1|\{y\})}\right)}{1-\prod\limits_{v'\in V_c/v} \tanh\left(\frac{1}{2}\ln\frac{Pr(v'=0|\{y\})}{Pr(v'=1|\{y\})}\right)} \tag{1.22}$$

qui peut être simplifié par :

$$\tanh\left(\frac{1}{2}\ln\frac{Pr(v=0|\{y\})}{Pr(v=1|\{y\})}\right) = \prod_{v'\in V_c/v} \tanh\left(\frac{1}{2}\ln\frac{Pr(v'=0|\{y\})}{Pr(v'=1|\{y\})}\right) \tag{1.23}$$

La fonction tanh étant une fonction monotone impaire, on peut décomposer la relation précédente de la manière suivante :

$$\text{sign}\left(\ln\frac{Pr(v=0|\{y\})}{Pr(v=1|\{y\})}\right) = \prod_{v'\in V_c/v} \text{sign}\left(\ln\frac{Pr(v'=0|\{y\})}{Pr(v'=1|\{y\})}\right) \tag{1.24}$$

$$\tanh\left|\frac{1}{2}\ln\frac{Pr(v=0|\{y\})}{Pr(v=1|\{y\})}\right| = \prod_{v'\in V_c/v} \tanh\left|\frac{1}{2}\ln\frac{Pr(v'=0|\{y\})}{Pr(v'=1|\{y\})}\right| \tag{1.25}$$

En utilisant le fait que :

$$f(x) = -\ln\tanh\left(\frac{x}{2}\right) = \ln\frac{\exp x+1}{\exp x-1} = f^{-1}(x) \tag{1.26}$$

on peut exprimer la valeur absolue du rapport de vraisemblance dans l'espace logarithmique par :

$$\left|\ln\frac{Pr(v=0|\{y\})}{Pr(v=1|\{y\})}\right| = f\left(\sum_{v'\in V_c/v} f\left(\left|\ln\frac{Pr(v'=0|\{y\})}{Pr(v'=1|\{y\})}\right|\right)\right) \tag{1.27}$$

Cette relation va servir de base pour la description de l'algorithme de propagation de croyance. La fonction $f(.)$ est représentée à titre indicatif sur la figure 1.9.

Algorithme de propagation de croyance (BP)

L'algorithme de propagation de croyance peut se décomposer en plusieurs étapes. Une première phase consiste à calculer les messages se propageant d'un noeud de données à un noeud de contrôle (cf figure 1.10). Une seconde étape calcule les messages générés au niveau des noeuds de contrôle. Une fois l'ensemble des messages mis à jour, ceux-ci sont propagés des noeuds de contrôle vers les noeuds de données (cf figure 1.11). Enfin, après un certain nombre d'itérations, l'information *a posteriori* associée à chaque noeud de données est mise à jour avant la prise de décision. Afin de faciliter la lecture de ce manuscrit, les messages probabilistes seront exprimés

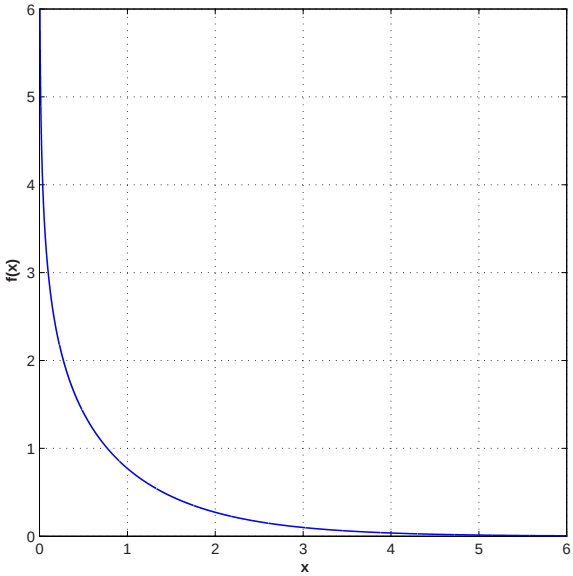


FIG. 1.9 – Représentation de la fonction $f(.)$ définie équation 1.26.

en log-rapports de vraisemblance. Nous noterons m_{vc}[8] les messages se propageant d'un noeud de données à un noeud de contrôle. De la même façon, la notation m_{cv}[9] est utilisée pour désigner les messages issus d'un noeud de contrôle et transmis à un noeud de données.

La mise à jour des messages m_{vc} issus du noeud de données v à l'itération i est calculée de la façon suivante (cf figure 1.10) :

$$m_{vc}^{i} = v_0 + \sum_{c' \in C_v/c} m_{c'v}^{i-1} \tag{1.28}$$

où v_0 représente le log-rapport de vraisemblance issu de l'observation y_v en sortie du canal :

$$v_0 = \ln \frac{Pr(y_v|v=0)}{Pr(y_v|v=1)} \tag{1.29}$$

et où C_v représente l'ensemble des noeuds de contrôle connecté au noeud de données v. A la première itération, les messages provenant des noeuds de contrôle sont nuls.

La deuxième étape de l'algorithme de propagation de croyance consiste à mettre à jour les messages en sortie d'un noeud de contrôle. Les messages m_{cv} sont calculés à

[8] variable to check node
[9] check to variable node

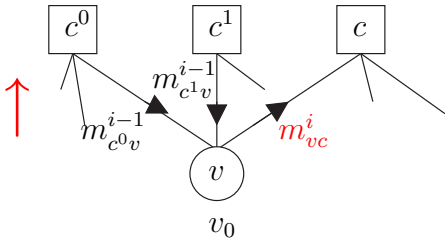


FIG. 1.10 – Illustration de la mise à jour des messages se propageant d'un noeud de données vers un noeud de contrôle m_{vc}.

l'itération i de la façon suivante (cf figure 1.11) :

$$\text{sign}\left(m_{cv}^{i}\right) = \prod_{v' \in V_c/v} \text{sign}\left(m_{v'c}^{i}\right) \tag{1.30}$$

$$\left|m_{cv}^{i}\right| = f\left(\sum_{v' \in V_c/v} f\left(\left|m_{v'c}^{i}\right|\right)\right) \tag{1.31}$$

où V_c représente l'ensemble des noeuds de données connectés au noeud de contrôle c.

Une itération de l'algorithme de propagation de croyance est réalisée lorsque tous les messages se propageant le long des branches ont été calculés par les deux relations précédentes. Après chaque itération, une décision peut être prise sur l'information *a posteriori* A_v^i associée au noeud de données v :

$$A_v^i = v^0 + \sum_{c' \in C_v} m_{c'v}^{i} \tag{1.32}$$

La décision sur la valeur binaire de chaque noeud de données est donc calculée en fonction du signe de l'information *a posteriori*. Le processus itératif est arrêté au bout d'un nombre maximum d'itérations. On peut également arrêter le processus itératif avant le nombre maximum d'itérations en calculant à chaque itération le syndrome. Si celui-ci est nul alors le décodage itératif a convergé vers un mot de code et le processus peut être arrêté.

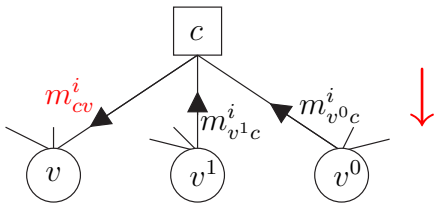


FIG. 1.11 – Illustration de la mise à jour des messages se propageant d'un noeud de contrôle à un noeud de données m_{cv}.

Algorithme BP et dérivés

En pratique, des algorithmes dérivés de l'algorithme BP peuvent être considérés. Ces algorithmes peuvent être vus comme une simplification de l'algorithme BP, et donc sous-optimaux (toujours sous l'hypothèse d'un graphe sans cycle)[10]. La diminution de performance engendrée par la sous-optimalité est à pondérer par le gain obtenu sur la complexité de l'organe de décodage. La majorité des algorithmes issus du BP reposent sur des opérations simplifiées de mise à jour des fiabilités en sortie des noeuds de contrôle.

Un tableau de synthèse, Table 1.1, résume les algorithmes fréquemment rencontrés dans la littérature. La simplification la plus couramment utilisée est celle de l'algorithme **BP-Based** [25], connu aussi sous le nom **Min-Sum**. Cette approximation repose sur le fait que le message calculé en sortie du noeud de contrôle est fortement dépendant du plus petit message entrant, en valeur absolue. La fonction $f(.)$ étant une fonction décroissante positive (cf figure 1.9), on peut écrire :

$$\sum_i f(|x_i|) \geq f(\min_i |x_i|) \tag{1.33}$$

et donc :

$$f\left(\sum_i f(|x_i|)\right) \leq \min_i |x_i| \tag{1.34}$$

La fiabilité du message en sortie du noeud de contrôle peut donc être approximée par celle du message le moins sûr. Cette simplification algorithmique permet une réduction de la complexité de décodage au niveau des noeuds de contrôle. En effet la fonction non linéaire $f(.)$ est remplacée par une simple fonction minimum. De plus, le nombre de messages différents, en valeur absolue, en sortie d'un noeud de contrôle est alors réduit à deux : la plus petite et la deuxième plus petite contribution entrantes. Dans le cas du BP, le nombre de messages en sortie du noeud de contrôle est égal au degré de connexion du noeud. Il en résulte donc une importante réduction des ressources mémoires nécessaires et de la complexité de l'entité de mise à jour. Une seconde propriété très intéressante de cet algorithme réside dans sa robustesse face à une mauvaise estimation des log-rapports de vraisemblance associés aux observations en sortie du canal. En effet, du fait de l'utilisation de l'opérateur minimum, tous les calculs se font à un coefficient multiplicatif près.

Par exemple, dans le cas d'une modulation BPSK (Binary Phase Shift Keying) et d'un canal de propagation à bruit blanc additif gaussien de variance σ^2 (AWGN), le log-rapport de vraisemblance du symbole reçu y, à présenter à l'entrée du décodeur, est égal à $2y/\sigma^2$. Dans le cas d'un décodage par l'algorithme Min-Sum, on peut présenter à l'entrée du décodeur un log-rapport de vraisemblance égal à αy. Ceci permet donc de s'affranchir de l'estimation de la variance du bruit.

[10] En pratique, dans le cas de graphe présentant des cycles, l'algorithme BP peut être moins performant qu'un algorithme dit sous-optimal

Nom de l'algorithme	Mise à jour des noeuds de données	Mise à jour des noeuds de contrôle
BP	$m_{vc}^i = y_v^0 + \sum_{c' \in C_v/c} m_{c'v}^{i-1}$	$\lvert m_{cv}^i \rvert = f\left(\sum_{v' \in V_c/v} f\left(\lvert m_{v'c}^i \rvert\right)\right)$
BP-Based/Min-Sum [25]	idem BP	$\lvert m_{cv}^i \rvert = \min_{v' \in V_c/v} \lvert m_{v'c}^i \rvert$
Offset Min-Sum [26]	idem BP	$\lvert m_{cv}^i \rvert = \min_{v' \in V_c/v} \lvert m_{v'c}^i \rvert + \beta$
Normalized Min-Sum [26]	idem BP	$\lvert m_{cv}^i \rvert = \alpha \min_{v' \in V_c/v} \lvert m_{v'c}^i \rvert$
λ-min [27]	idem BP	$\lvert m_{cv}^i \rvert = f\left(\sum_{v' \in V_c^\lambda/v} f\left(\lvert m_{v'c}^i \rvert\right)\right)$
A-min* [28]	idem BP	si $v = \arg\min_{v' \in V_c} \lvert m_{v'c}^i \rvert$ $\lvert m_{cv}^i \rvert = f\left(\sum_{v' \in V_c/v} f\left(\lvert m_{v'c}^i \rvert\right)\right)$ sinon $\lvert m_{cv}^i \rvert = f\left(\sum_{v' \in V_c} f\left(\lvert m_{v'c}^i \rvert\right)\right)$
APP - check [25]	idem BP	$\lvert m_{cv}^i \rvert = f\left(\sum_{v' \in V_c} f\left(\lvert m_{v'c}^i \rvert\right)\right)$
APP-variable [25]	$m_{vc}^i = y_v^0 + \sum_{c' \in C_v} m_{c'v}^{i-1}$	$\lvert m_{cv}^i \rvert = f\left(\sum_{v' \in V_c/v} f\left(\lvert m_{v'c}^i \rvert\right)\right)$

TAB. 1.1 – Liste des algorithmes de décodage de code LDPC couramment utilisés dans la littérature. Les différentes simplifications utilisées peuvent être combinées pour produire d'autres algorithmes de décodage et atteindre un objectif de compromis performance-complexité

L'utilisation de cette approximation introduit une perte de performances par rapport à l'algorithme BP. Ces pertes seront d'autant plus importantes que le degré de connexion des noeuds est important. En effet, dans ce cas, l'approximation du minimum devient moins pertinente.

Cet algorithme peut être amélioré par traitement des messages en sortie du noeud de contrôle. L'approximation du minimum sur-estime les messages en sortie du noeud de contrôle. Ainsi une pondération, ou une correction, peut être apportée sur chaque

message afin d'atténuer l'effet de la sur-estimation. L'algorithme est alors appelé **normalized Min-Sum** [26] dans le cas d'une pondération et **offset Min-Sum** dans le cas d'une correction [26]. Une multitude de techniques sont proposées pour le choix des facteurs de correction. La détermination des facteurs correctifs est un problème d'optimisation complexe. Celui ci va en général dépendre du code (distribution des degrés...) et du canal de propagation (variance du bruit...). Les auteurs de [26] proposent par exemple, une normalisation basée sur la moyenne des messages en sortie du noeud de contrôle. Le rapport entre la moyenne des messages en sortie d'un noeud de contrôle décodé par un algorithme Min-Sum, et la moyenne des messages en sortie d'un noeud de contrôle décodé par un algorithme BP, permet de calculer un facteur de normalisation. Ce facteur est bien entendu optimisé pour un rapport signal à bruit et un type de noeud de contrôle. Comme le mentionnent les auteurs de [29], même si ces post-traitements apportent de très bonnes performances dans le cas des codes réguliers, cela n'est pas toujours le cas pour des codes irréguliers.

La simplification de l'algorithme Min-Sum repose sur le fait qu'un message en sortie d'un noeud de contrôle est fortement dépendant du plus petit message d'entrée. Le principe de l'algorithme **λ-Min**, introduit dans [27], consiste à calculer la valeur absolue du message de sortie à partir des λ[11] plus petits messages d'entrée (sous entendu en valeur absolue). Les dégradations de performances sont d'autant plus petites que le paramètre λ est grand. De plus, le nombre de messages différents en sortie d'un noeud de contrôle est alors réduit à $\lambda + 1$, minimisant ainsi la complexité de mémorisation des messages.

Suivant le même objectif de réduction du nombre de messages à stocker en sortie d'un noeud de contrôle, l'algorithme **A-min*** a été proposé dans [28]. L'idée consiste à calculer sans approximation le message de la plus petite contribution, c'est à dire le message ayant la plus petite fiabilité. L'ensemble des autres messages sont égaux et calculés à l'aide d'une approximation. Cet algorithme offre un bon compromis, les performances de décodage étant très proches de celles obtenues par un algorithme BP.

Le principe de réduction du nombre de messages différents en sortie des noeuds, avait initialement été introduit dans [25]. Ces algorithmes sont connus sous le nom d'algorithme **APP-based**. L'idée consiste à réduire le nombre de messages (toujours en valeur absolue) en sortie d'un noeud, à une seule valeur. L'algorithme issu de ce principe pour un noeud de contôle est appelé **APP-Check** et, pour un noeud de données, **APP-variable**. Ces algorithmes, quoique très intéressants du point de vue des simplifications offertes, souffrent de pertes de performances importantes en comparaison aux autres algorithmes. Ces pertes sont dues à la corrélation entre les messages d'entrée et de sortie. En effet, dans le cas de ces algorithmes, la contribution apportée par un noeud lui est renvoyée. Cet effet sera d'autant plus important que le degré du noeud est faible. L'auto-contribution apportée dans ce cas est comparable à

[11] le cas $\lambda = 1$ est equivalent à l'approximation de l'algorithme Min-Sum

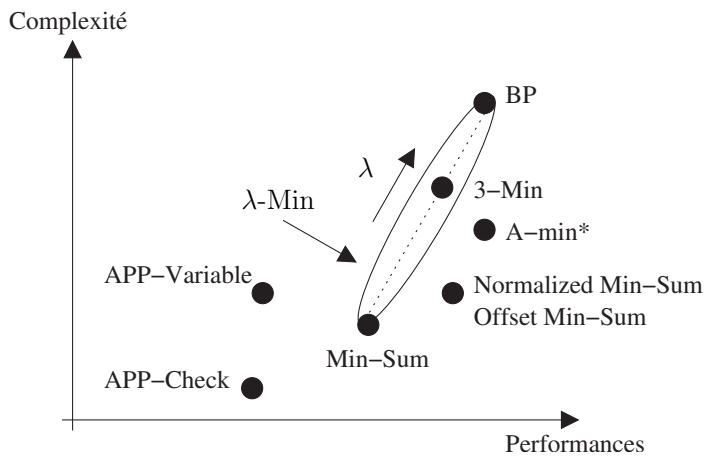


FIG. 1.12 – Comparaison des algorithmes de décodage à travers une représentation dans le plan performance complexité.

celle apportée par les cycles de faibles longueurs dans le graphe.

Il faut noter que les différentes simplifications de l'algorithme BP peuvent être combinées pour former de nouveaux algorithmes adaptés à un contexte donné, ou permettant d'atteindre un compromis performance complexité visé. Par exemple, les auteurs de [29] proposent l'utilisation de l'algorithme Min-Sum avec un post-traitement appliqué à la fois aux messages sortant des noeuds de contrôle et des noeuds de données (2-D).

En guise de conclusion, une comparaison de ces algorithmes est proposée figure 1.12. Chaque algorithme est représenté dans un plan caractérisant son niveau de performance et de complexité.

Outre le choix de l'algorithme de décodage pour la mise à jour des noeuds, l'ordonnancement mis en oeuvre est aussi un paramètre très important. Ce point est abordé dans le paragraphe suivant.

Ordonnancement de l'algorithme de propagation de croyance

Nous verrons dans la suite de ce mémoire l'importance cruciale du choix de l'ordonnancement des mises à jour, à la fois d'un point de vue réalisation matérielle, mais aussi d'un point de vue performance.

On distingue deux grandes familles d'ordonnancement. Le premier type d'ordonnancement, est l'ordonnancement par inondation ou *flooding scheduling*. Cet ordonnancent est le séquencement classique initialement associé à l'algorithme de propagation de croyance. Une première phase consiste à mettre à jour l'ensemble des noeuds de données, suivi par une étape de mise à jour de l'ensemble des noeuds de contrôle. Cet

ordonnancement est très générique et présente l'avantage d'être très simple à mettre en oeuvre. Du fait des mises à jour en deux étapes de chaque type de noeuds, la convergence de l'algorithme est relativement lente. Typiquement, le nombre d'itérations nécessaire est de l'ordre de 50 pour des tailles de codes de l'ordre du kilo bit.

Remarque : un ordonnancement par inondation appliqué à un Turbo-code consisterait à un décodage dans la même demi-itération des deux codes convolutifs, suivi par un échange d'information extrinsèque.

Le second séquencement couramment rencontré dans la littérature est l'ordonnancement brassé dit aussi séquencement *shuffle* . Cet ordonnancement initialement introduit dans [30] et [31], consiste à mettre à jour chaque noeud de données (resp. de contrôle) dans un ordre pré-déterminé. En d'autres termes, chaque noeud sera mis à jour, au cours d'une itération, autant de fois que son degré de connexion. L'idée sous jacente est de mettre à jour un noeud dès qu'une nouvelle information est disponible. On parle de *Horizontal shuffle scheduling* [30] quand les noeuds de contrôle sont mis séquentiellement à jour et de *Vertical shuffle scheduling* pour les noeuds de données [32]. La figure 1.13 illustre ces deux séquencements.

L'avantage de ce type d'ordonnancement est une convergence plus rapide de l'algorithme itératif. Cependant, cette augmentation de la rapidité de convergence est à pondérer avec l'augmentation du nombre d'opérations effectuées pour chaque itération de décodage. En effet, au cours d'une itération, le nombre d'opérations réalisées pour la mise à jour des noeuds est supérieur dans le cas d'un séquencement *shuffle* que dans un ordonnancement par inondation. Nous verrons par la suite que ce type de séquencement est particulièrement intéressant et permet dans certains cas de réduire la complexité de décodage. Les auteurs de [32] suggèrent un ordonnancement *shuffle* non pas noeud mais par groupe de noeuds *g*roup shuffle. Si le code LDPC est construit de telle sorte qu'un noeud dans un groupe ne soit connecté qu'une seule fois, alors la convergence de ce séquencement sera identique à celui de l'ordonnancement *shuffle* noeud par noeud. Il faut noter que la convergence de ces algorithmes et les gains associés ont été analysés théoriquement dans [33, 34, 35].

Remarque : Le décodage d'un Turbo-code tel que présenté précédemment peut être vu comme un séquencement *shuffle* par groupe. Dans la même itération un symbole systématique est mis à jour deux fois. Les travaux de [36] montrent qu'en imposant des règles de constructions sur l'entrelaceur, un ordonnancement de type *shuffle* noeud par noeud est possible quand les deux codes convolutifs sont décodés en parallèle. Dans ce cas, dans la même itération[12] le second décodeur tire partie du décodage du premier et inversement.

[12] dans ce cas une itération consiste au décodage, en parallèle, des deux codes. En terme de latence une itération de ce séquencement correspond à une demi itération de l'algorithme classique.

	Horizontal Shuffle Scheduling	Vertical Shuffle Scheduling
Étape 1	Mise à jour du premier noeud de contrôle	Mise à jour du premier noeud de données
Étape 2	Mise à jour des noeuds de données	Mise à jour des noeuds de contrôle
Étape 3	Mise à jour du noeud de contrôle suivant	Mise à jour du noeud de données suivant
Étape 4	Mise à jour des noeuds de données	Mise à jour des noeuds de contrôle
Étape 5	Mise à jour du noeud de contrôle suivant	Mise à jour du noeud de données suivant
⋮	⋮	⋮

FIG. 1.13 – Illustration de l'ordonnancement Horizontal Shuffle Scheduling et Vertical Shuffle Scheduling au cours de l'itération i

D'autres séquencements existent dans la littérature. On peut citer par exemple l'ordonnancement probabiliste de [37]. Ce séquencement consiste à mettre à jour les noeuds avec une fréquence moyenne proportionnelle à la longueur du plus petit cycle auquel est connecté ce noeud. Ce séquencement apporte de très bonnes performances de décodage puisque la corrélation entre les messages est diminuée. Cependant, en pratique, ce type d'ordonnancement semble très difficile à mettre en oeuvre. Dans le but de diminuer le nombre d'opérations durant le décodage, le concept de noeud dormant (sleeping node) a été introduit dans [38]. Ce séquencement consiste à mettre à jour un noeud de données uniquement dans le cas où l'information *a posteriori* associée à ce noeud, est inférieur à un seuil. De la même manière, ce séquencement semble difficile à mettre en oeuvre en pratique, mais peut d'être adapté dans certain contexte.

Pour des applications et/ou des codes particuliers, des ordonnancements spécifiques peuvent être introduits. Pour une application où une faible latence est requise, le séquencement "replica shuffled" [34] peut être adapté. Ce séquencement particulier a été introduit à la suite de l'observation suivante : lors d'un décodage utilisant un ordonnancement *shuffle* le résultat de décodage dépend de l'ordre d'activation des noeuds. Plus particulièrement, dans le cas d'un séquencement *Vertical shuffle*, plus un noeud sera mis à jour tard dans une itération, plus il sera fiable. Le principe de cet ordonnancement est donc d'utiliser la diversité apportée par deux séquencements différents. Ce type d'approche permet de réduire le nombre d'itérations et donc la latence de décodage. Cependant, l'architecture associée doit comprendre autant de décodeurs que de diversité souhaitée.

Il existe aussi des séquencements qui peuvent être adaptés à une structure de code particulière. Dans [39], le code LDPC proposé permet une transformation en un treillis à deux états où chaque bit peut intervenir un certain nombre de fois. Le séquencement proposé est alors le décodage par fenêtre du treillis suivi par une mise à jour des symboles. Ce type d'ordonnancement sera développé plus précisément par la suite.

Le décodage des codes LDPC offre un grand choix d'algorithmes et d'ordonnancements. Il parait donc possible de choisir judicieusement un couple algorithme ordonnancement afin d'atteindre le meilleur compromis complexité-performance. Nous verrons par la suite que cet objectif peut également être atteint en travaillant sur la définition conjointe du code, du séquencement et de l'architecture de décodage.

1.3.4 Encodage des codes LDPC

Les codes LDPC ont la particularité d'être définis par leur matrice de contrôle de parité. Comme nous l'avons mentionné auparavant, du fait de leur complexité d'encodage et de décodage et des moyens matériels de l'époque, ces codes ont eu peu d'impact sur la communauté de la théorie du codage au moment de leur découverte. En effet, la manière triviale de determiner le mot de code est d'utiliser la matrice génératrice $\mathbf{G}$, facilement calculable à partir de la matrice de contrôle de parité $\mathbf{H}$. Dans la majorité des cas, la matrice génératrice associée à un code dont la matrice de contrôle de parité est de faible densité est dense. Dans le cas où le code n'a pas de structure, la complexité d'encodage associé est alors importante. Pour réduire cette complexité, des approches que l'on peut classer dans deux grandes familles ont été développées. La première consiste à post-traiter la matrice de contrôle de parité de façon à introduire une forme facilement encodable. La seconde est basée sur la construction d'une matrice de contrôle de parité contrainte, construite à l'origine pour faciliter l'encodage.

En ce qui concerne la première approche, les auteurs de [40] préconisent une transformation de la matrice de contrôle de parité par combinaisons linéaires de lignes et de colonnes en une autre matrice de contrôle de parité de forme semi-triangulaire.

La complexité d'encodage dépend alors d'un paramètre caractérisant l'écart entre la matrice semi-triangulaire et la matrice triangulaire. Une fois les bits intervenant dans la forme semi-triangulaire obtenue, les autres bits de redondance sont obtenus par substitution.

La seconde méthode consiste en la définition d'une structure de code contrainte. Une première construction très largement répandue, consiste à construire une matrice de contrôle de parité définie par :

$$\mathbf{H} = [\mathbf{H_s}\ \mathbf{H_p}] \tag{1.35}$$

Le mot de code $\underline{x}$ est alors divisé en un mot d'information $\underline{c}$ et un mot de redondance $\underline{p}$. La relation de parité s'écrit alors :

$$\begin{aligned} \mathbf{H}\underline{x}^t &= \underline{0}^t \\ [\mathbf{H_s}\ \mathbf{H_p}] \begin{bmatrix} \underline{c}^t \\ \underline{p}^t \end{bmatrix} &= \underline{0}^t \\ \mathbf{H_p}\underline{p}^t &= \mathbf{H_s}\underline{c}^t \end{aligned} \tag{1.36}$$

(1.37)

Pour simplifier les notations, nous appellerons vecteur de projection le vecteur $\underline{v}$ défini par :

$$\underline{v}^t = \mathbf{H_s}\underline{c}^t \tag{1.38}$$

Il en résulte donc que l'ensemble des bits de parité peut se déduire de la façon suivante :

$$\underline{p}^t = \mathbf{H_p}^{-1}\underline{v}^t \tag{1.39}$$

Cette relation montre que la première contrainte sur le code est l'existence de la matrice inverse $\mathbf{H_p}^{-1}$. La matrice $\mathbf{H_p}$ peut être de forme triangulaire permettant par simple substitution le calcul des bits de redondance. Plus particulièrement, les matrices $\mathbf{H_p}$ de type bi-diagonale sont intéressantes pour obtenir un encodage simple. Dans le cas où la matrice est strictement bi-diagonale (cf figure 1.14 (a)) le code LDPC (figure 1.15(a)) peut être vu indifféremment comme :

- un code de la famille des codes Repeat Accumulate [41][42] (figure 1.15(c)).
- une concaténation série d'un code de parité et d'un code convolutif récursif (accumulateur) (figure 1.15(b)).
- un code de la famille self concatenated code [43], dont le code de base est un accumulateur (figure 1.15(c)).

La détermination des bits de redondance peut se faire en calculant dans un premier temps le vecteur de projection puis par une accumulation de ce vecteur :

$$p_k = p_{k-1} + v_k \tag{1.40}$$

Une autre forme très intéressante de matrice $\mathbf{H_p}$ de type bi-diagonale est celle illustrée figure 1.14 (b). Ce type de structure a été retenu dans le cadre de la normalisation

$$\begin{bmatrix} 1 & & & & & & \\ 1 & 1 & & & & & \\ & 1 & 1 & & & & \\ & & 1 & 1 & & & \\ & & & \ddots & \ddots & & \\ & & & & 1 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & & & & & \\ & \ddots & \ddots & & & & \\ 1 & & 1 & 1 & & & \\ & & & \ddots & \ddots & & \\ & & & & 1 & 1 \\ 1 & & & & & 1 \end{bmatrix}$$

DVB-S2 (a) IEEE 802.16e, 802.11n (b)

FIG. 1.14 – Forme particulière de la matrice $\mathbf{H_p}$ normalisée dans des standards.

IEEE 802.16e (Wimax Mobile) et IEEE 802.11n (Wifi). Le premier bit de redondance est obtenu en sommant l'ensemble des bits du vecteur de projection $\underline{v}$:

$$p_0 = \sum_i v_i \tag{1.41}$$

L'ensemble des autres bits de redondance est alors calculé par simple substitution. Cette forme de matrice peut être interprétée comme une forme circulaire de l'accumulateur. La présence de trois éléments non nuls dans une colonne provient du fait que, en pratique, l'accumulateur ne peut pas être circulaire sans un petite modification du code. Ce principe a été illustré par Berrou *et al.* , dans le but d'améliorer la distance minimale d'un Turbo-code [44].

Une autre famille de codes permet un encodage simple : ces codes font partie de la famille des codes Quasi-cyclic (QC) [45]. Il a été démontré que pour certaines familles

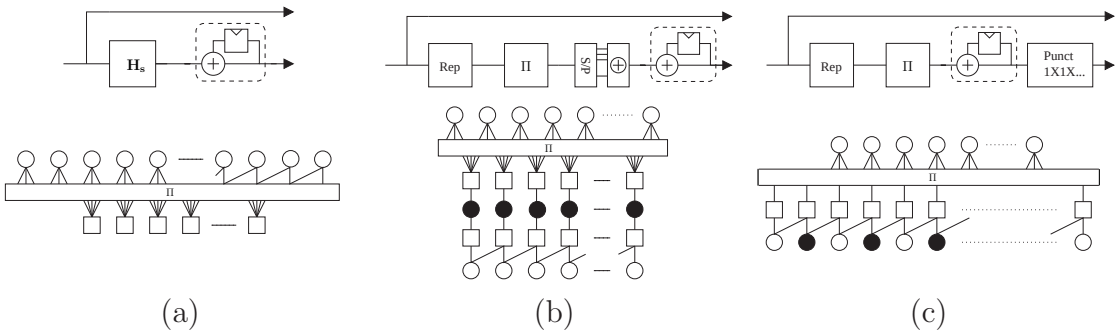


FIG. 1.15 – Illustrations des différentes représentations d'un code de type Repeat Accumulate. Ces codes peuvent être vus comme des codes LDPC (a). Ils peuvent aussi être représentés par une concaténation d'un code de parité, alimenté par un organe de répétition (Rep) dont les sorties sont entrelacées (Π), et d'un accumulateur (b). Enfin, ces codes peuvent être vus comme la concaténation série d'un répéteur, d'un accumulateur. Les sorties sont alors poinçonnées (Punct) pour obtenir le rendement codage choisi (c).

de codes QC-LDPC, l'encodage peut se faire de manière très simple avec des registres à décalages. De nombreux exemples sont décrit dans [46].

Nous avons vu que les méthodes d'encodage des codes LDPC sont fortement liées à la construction des codes. Ainsi chaque famille de codes aura un structure de codage particulière. Dans le paragraphe suivant, nous proposons d'établir un rapide tour d'horizon sur les méthodes de construction de codes.

1.3.5 Construction des codes LDPC

Construire un code LDPC revient à définir la position de tous les éléments non nuls dans la matrice de contrôle de parité. La principale difficulté dans la construction des codes LDPC réside dans le choix des paramètres définissant le code. Ainsi, pour une taille et un rendement donné, il faut choisir dans un premier temps le profil d'irrégularité des noeuds de données et des noeuds de contrôle. Des outils puissants, qui ne seront pas détaillés dans ce manuscrit mais dont un bon résumé se trouve dans [47], permettent sous certaines hypothèses (souvent des cas asymptotiques) de fixer un profil d'irrégularité minimisant le seuil de convergence. Ces outils sont de la famille des algorithmes d'évolution de densité (Density Evolution) dont la base sont les travaux de Ten Brink développés initialement dans le cadre de l'analyse des Turbo-codes parallèles [48] (EXIT chart). Dans le cas des codes LDPC, l'évolution de densité avec approximation gaussienne a été réalisée par Chung, Richardson et Urbanke [49]. Cependant, ces résultats obtenus dans le cas asymptotique sont à utiliser avec précautions dans le cas de codes de longueur finie. Ces méthodes constituent néanmoins un bon point de départ pour la détermination des profils d'irrégularité. Ce point sera développé par la suite dans ce manuscrit.

Une fois le profil d'irrégularité fixé, la position de chaque élément non nul doit être déterminée selon la structure de code choisie. Un grand nombre d'algorithmes existe dans la littérature. La plupart des algorithmes essayent de construire un code ayant une bonne distribution des cycles. Un algorithme particulièrement intéressant est l'algorithme PEG (Progressive Edge Growing) [50] et autres versions dérivées. Cet algorithme travaille sur la matrice de contrôle de parité colonne par colonne. A chaque nouvelle colonne, l'algorithme place les éléments non nuls, suivant la distribution des noeuds de données spécifiée, de manière à maximiser localement le cycle du noeud de données considéré, sous la contrainte d'une distribution des noeuds de contrôle fixée. Cette famille d'algorithme de construction sera plus précisément détaillée dans le chapitre 2.

Alors que dans le cas asymptotique les codes LDPC construits aléatoirement sont très bons, dans le cas de taille finies, les constructions déterministes semblent fournir les meilleurs candidats. Ainsi les constructions de type Repeat Accumulate ou par expansion offrent de très bonnes performances. Une brève description de ces deux

familles est proposée à titre d'exemple. De nombreuses autres familles existent dans la littérature et ne sont pas adressées dans ce manuscrit.

Construction par expansion

La construction de code LDPC par expansion consiste à définir un graphe de base associé à une matrice de contrôle de parité de base. L'expansion de cette matrice est réalisée en remplaçant chaque élément non nul de la matrice par une matrice d'expansion. Ce concept a initialement été proposé par Gallager [3]. Plus particulièrement, de nombreux travaux ont porté sur le cas où les matrices d'expansion sont des matrices identité permutées [45][51][52]. A titre d'exemple, la structure de matrice de contrôle de parité proposée dans [51] est la suivante :

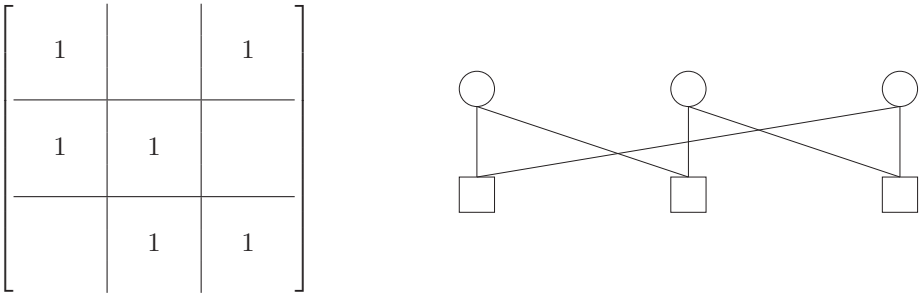
$$\mathbf{H} = \begin{pmatrix} \mathbf{I}_0 & \mathbf{I}_0 & \cdots & \mathbf{I}_0 \\ \mathbf{I}_0 & \mathbf{I}_{p_{1,1}} & \cdots & \mathbf{I}_{p_{1,L-1}} \\ \vdots & & \ddots & \vdots \\ \mathbf{I}_0 & \mathbf{I}_{p_{J-1,1}} & \cdots & \mathbf{I}_{p_{J-1,L-1}} \end{pmatrix} \tag{1.42}$$

où $\mathbf{I}_x$ représente une matrice identité de taille $z \times z$ circulairement permutée de x positions vers la droite. Le code ainsi défini est un code régulier quasi-cyclique de taille $N = Lz$ et de rendement $R = 1 - J/L$. Les coefficients de permutation $p_{i,j}$ de chaque matrice identité peuvent être déterminés aléatoirement ou suivant des lois [51]. En particulier pour cette structure, il a été démontré que la longueur du plus petit cycle est bornée par 12. Des conditions nécessaires sont proposées dans [51] pour atteindre cette borne. Précédemment, Tanner avait proposé, dans une forme quasi-cyclique, des codes Repeat Accumulate [53]. A partir d'une structure bi-diagonale, chaque "1" d'une matrice de base est remplacé par une matrice identité permutée. Ce concept a été par la suite repris et généralisé : on parle alors de *protographe* [54]. Un code LDPC dérivé d'un protographe est un code dont le graphe est une copie permutée d'un graphe de base (cf figure 1.16). Nous verrons par la suite que ce type de code peut être particulièrement intéressant pour une réalisation matérielle. Ce type de représentation permet de généraliser bon nombre de constructions de code existant dans la littérature et notamment les constructions quasi-cycliques.

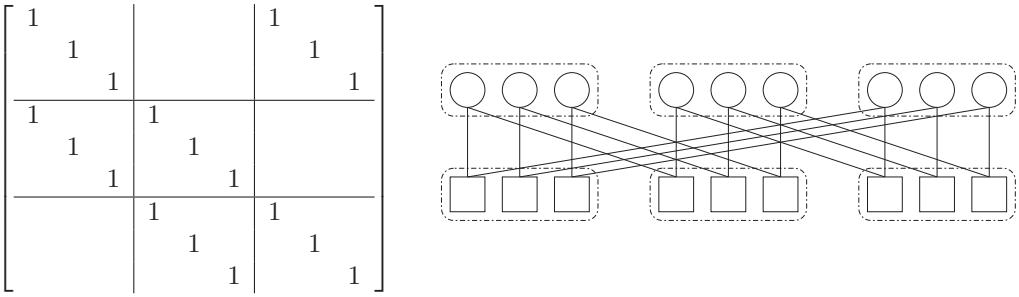
Construction de type Repeat Accumulate

Cette famille de codes décrite brièvement dans la section précédente, consiste en une matrice de contrôle de parité divisée en deux, dont une partie est de type bi-diagonale. On rappelle que la matrice de contrôle de parité peut s'écrire :

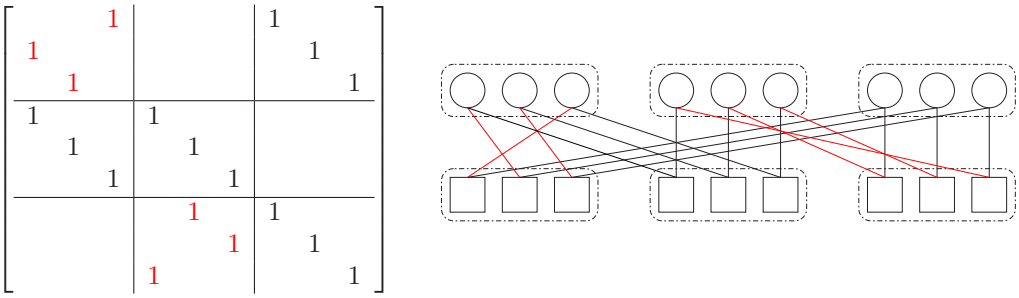
$$\mathbf{H} = [\mathbf{H_s}\ \mathbf{H_p}]$$



(a) - Exemple d'un protographe



(b) - Le protographe est répété trois fois



(b) - Le graphe dérivé est obtenu par permutation de certaines branches

FIG. 1.16 – Exemple de construction d'un code à partir d'un protographe et de matrices d'expansion de type identité permutées.

Dans le cas où la matrice $\mathbf{H_p}$ est strictement bi-diagonale et que la matrice $\mathbf{H_s}$ est régulière[13], on parle de codes de la famille Repeat Accumulate [41]. Ces codes sont des codes LDPC irréguliers avec un proportion de noeuds de degré 2 égale au nombre de bit de parité[14]. Dans le cas où la matrice $\mathbf{H_s}$ est irrégulière on parle alors d'Irregular Repeat Accumulate Code [42]. De nombreux travaux ont porté sur l'étude de ces codes, et des performances comparables à celles des meilleurs codes LDPC irréguliers ont été obtenues [55]. Ils sont d'autant plus intéressants que le rendement

[13] Par abus de langage, le fait de parler de matrice régulière revient à dire que le graphe associé à cette matrice est régulier

[14] Plus précisément, M-1 noeuds sont de degré 2 et un noeud est de degré 1 (cf figure 1.14 (a))

de codage est fort [56]. Les premiers codes LDPC standardisés dans le cadre de la norme DVB-S2 appartiennent à cette catégorie de codes. Nous reviendrons par la suite plus précisément sur cette structure et ses particularités.

De nombreuses constructions ont été proposées pour la construction de la matrice $\mathbf{H_s}$. Nous pouvons citer par exemple les structures de type Pi-rotation [57] et dual Pi-rotation [58] qui permettent une analyse, une construction et un encodage très simple. D'une manière générale, les codes de type Repeat Accumulate peuvent être représentés sous la forme illustrée figure 1.15. Comme décrit précédemment, des constructions hybrides telles que représentées sur la figure 1.14 (b) sont envisageables. Nous pouvons citer par exemple les codes de type Accumulate Repeat Accumulate codes [59][60] qui peuvent être vus comme des codes Repeat Accumulate précodés.

Dans la suite de ce manuscrit nous nous intéresserons plus particulièrement à ce type de code dérivé des codes Repeat Accumulate. Le chapitre 2 décrira pourquoi cette famille est à nos yeux très intéressante.

1.4 Réalisation de décodeurs LDPC

La réalisation matérielle de décodeurs LDPC suscite beaucoup d'intérêt au niveau de la communauté scientifique. Du fait du nombre important de degrés de liberté offerts par un code LDPC, de la définition du code au choix de l'ordonnancement, les solutions de réalisations sont nombreuses. Dans une optique de synthèse, nous nous intéresserons tout d'abord à la méthodologie de conception. Dans un deuxième temps, quelques notions de base sur la réalisation de décodeurs seront présentées.

1.4.1 Méthodologie de conception

Deux approches sont régulièrement utilisées pour la conception d'architecture de décodage. La première approche consiste, dans un premier temps, à optimiser un ensemble de codes sur des critères de performances. Dans une seconde phase, une reflexion est menée sur la réalisation matérielle du code. D'une manière générale, il est rare que de tels codes soient facilement adaptables à une architecture. Souvent une architecture est définie pour une famille de codes très restreinte, le décodeur ne pouvant traiter qu'une seule taille de code ou qu'un seul rendement de codage. C'est le cas par exemple de l'architecture proposée dans [61]. Cette architecture entièrement parallèle et très efficace, est composée d'autant de processeurs de calcul que de noeuds dans le graphe. Cette réalisation est la transposition directe du graphe de Tanner du code sur le silicium. Ce décodeur permet d'atteindre des débits de 1Gbit/s, mais seulement pour un seul code de taille $N = 1024$ bits et de rendement $R = 1/2$...

Globalement, la réalisation d'un décodeur sans aucune information *a priori* sur le code n'est pas sans poser de nombreux problèmes. D'un coté, une architecture de

décodeur générique, basée sur des processeurs séries très flexibles, peut souffrir d'une latence de calcul élevée. De l'autre, une réalisation très spécifique permet d'atteindre des débits élevés mais au détriment de la flexibilité. Ainsi dans ce cas, paralléliser l'architecture peut poser des problèmes d'accès aux organes de mémorisation, une donnée, ne pouvant être lue et écrite au même moment dans la même mémoire. Ce problème, adressé dans [62], peut être résolu pour n'importe quel code par la mise en place d'une organisation mémoire et d'un ordonnancement en relation avec la matrice de contrôle de parité du code. Cette solution, quoique très intéressante en pratique, semble difficile à mettre en oeuvre dans le cas d'un décodeur ayant pour but de supporter un grand nombre de tailles et de rendements de codage.

Cette solution nous amène naturellement vers une seconde méthode de conception d'architecture. Cette méthode, très largement mise en oeuvre dans la littérature, consiste en une approche conjointe entre la détermination d'une famille de codes et une architecture de décodage. Cette méthode de conception bien connue dans le cadre de la construction d'entrelaceurs de Turbo-code (voire [63] par exemple), a été explorée dans le cadre des codes LDPC par Boutillon *et al.* dans [64]. L'architecture d'un décodeur est tout d'abord choisie. Dans un deuxième temps, des contraintes sur la construction du code LDPC sont imposées en adéquation avec l'architecture choisie. Ces méthodes de conception conjointes sont d'autant plus intéressantes que le code est structuré. Ainsi les constructions déterministes, de type constructions par expansion, sont particulièrement adaptées. On peut citer par exemple les travaux de [65], où la matrice de contrôle de parité est construite à partir de matrices identité permutées de taille $z \times z$. Ce type de construction permet la définition d'une architecture semi-parallèle efficace. En effet, la construction par bloc de taille $z \times z$ permet un parallélisme de facteur z, qui s'accompagne d'un découpage de la mémoire par bloc de z éléments. Une donnée n'intervenant qu'une seule fois dans un groupe de z équations de parité, les éventuels problèmes d'accès mémoires sont évités. Il faut aussi noter que ce type de construction est particulièrement bien adapté à un séquencement de type *shuffle*. L'illustration de cette propriété sera présentée dans les chapitres suivants.

Plus particulièrement, dans le cas de codes construits à partir de matrices identité permutées, des règles de construction sur les coefficients de permutations peuvent être dérivées à partir d'un ordonnancement spécifique [66]. Cette méthodologie sera présentée dans la suite du manuscrit. On peut noter que certains codes normalisés au sein du standard IEEE 802.16e ont été construits pour permettre un séquencement de type *shuffle* pipeliné[15] [67]. Les codes normalisés dans le cadre du IEEE 802.11n sont aussi construits dans un souci de réalisation comme décrit dans [68].

Dans le même esprit, Mansour *et al.* proposent dans [69], une architecture semi parallèle associée à un séquencement de type *shuffle* pour des codes Repeat Accumulate construits à partir de matrices identité permutées. L'architecture présentée permet un

[15] Cette propriété est même explicitement décrite dans la norme et sera détaillé par la suite

niveau de parallélisme égal à la taille d'une matrice identité. Il faut cependant noter que l'architecture de décodage ne tire pas partie de la forme bi-diagonale de la matrice de contrôle de parité. Cet particularité des codes Repeat Accumulate est utilisée dans l'architecture proposée dans [70]. L'architecture fait la distinction entre les noeuds de données correspondant aux bits de redondance, et les autres noeuds de données. A ces deux types de noeuds sont associés deux types de processeurs différents. Ces deux types de processeurs sont connectés au travers de deux réseaux de permutations à un groupe de processeurs de contraintes de parité. Du fait de la structure bi-diagonale de la matrice, le réseau de permutation entre les processeurs de contraintes de parité et les processeurs associés aux noeuds de données de redondance est simplifié. Les auteurs de [70] parlent alors de réseaux de permutations en zig-zag, en référence au graphe de Tanner des codes Repeat Accumulate. Pour éviter les problèmes d'accès multiples aux organes de mémorisation causés par la parallélisation, des règles de constructions sont définies sur le réseau de permutations connectant les noeuds de contrôle et les noeuds de données associés à des bits systématiques. Cette architecture particulière est à l'origine de la première publication sur la réalisation d'un décodeur appliqué aux codes LDPC Repeat Accumulate de la norme DVB-S2 [71].

1.4.2 Éléments de réalisation

Comme nous avons pu le mentionner précédemment, un décodeur LDPC peut se modéliser d'une manière assez générique par deux types de processeurs et un ou plusieurs réseaux de permutations. Un processeur associé à un noeud de données sera noté VNP (Variable Node Processor). Le processeur associé à un noeud de contrôle sera identifié par le sigle CNP (Check Node Processor). D'une manière générale, ces deux processeurs peuvent être réalisés soit par une architecture série, soit par une architecture parallèle. Un exemple d'architecture parallèle de processeur est illustré sur la figure 1.17. Cette architecture charge en parallèle autant de données que le degré de connexion du noeud. Cette contrainte impose donc la possibilité de lecture/écriture en parallèle des organes de mémorisation. La latence d'un tel processeur est faible et dépend de la technologie choisie. Cependant, sa complexité sera d'autant plus importante que le degré de connexion sera grand. Une telle architecture pose aussi le problème de la flexibilité du processeur. Ainsi, quand le degré de connexion des noeuds n'est pas constant (typiquement quand un système supporte plusieurs rendement de codage) l'architecture parallèle des processeurs peut souffrir du manque de généricité. Ce point sera abordé plus précisément par la suite.

Une architecture série des processeurs de noeuds peut être aussi envisagée. Un exemple de réalisation est illustré figure 1.18. Contrairement à l'architecture parallèle, les données sont chargées en série et mémorisées (dans une mémoire de type First In First Out (FIFO) par exemple). Ce type de processeur est très générique si la profondeur de la mémoire est suffisamment grande. Cependant, les données de sortie du processeur ne sont disponibles qu'après une latence proportionnelle au degré de connexion du noeud. Cette propriété peut être d'autant plus gênante que le processeur

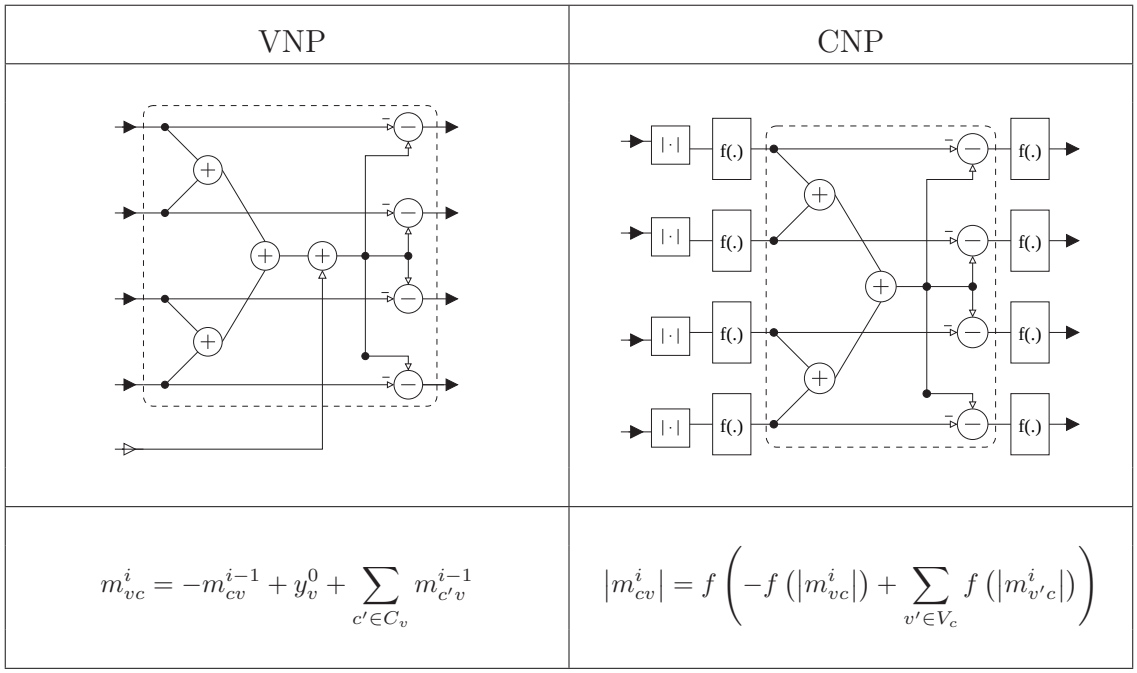


FIG. 1.17 – Exemple d'architecture parallèle des processeurs associés aux noeuds de données et de contrôle. Le décodage se fait suivant l'algorithme BP. Seul le calcul des fiabilité des messages de sorties du noeud de contrôle est illustré. Comme décrit dans [27], le traitement par la fonction $f(.)$ peut être indifféremment réalisé soit au niveau du noeud de contrôle, soit un niveau du noeud de données

n'est pas "pipeliné", et que le degré du noeud est important.

Dans la famille des processeurs série, on peut citer les travaux de topologie présentés dans [72]. Un exemple d'architecture série de processeur de noeuds de contrôle est proposé en utilisant une représentation en treillis. En effet, toute équation de parité peut se représenter sous la forme d'un treillis à deux états comme illustré figure 1.20. La représentation en treillis se déduit par l'insertion de noeuds cachés représentant l'état du treillis.

Comme mentionné dans [72], un algorithme de type aller retour (Forward Backward) peut donc être mis en oeuvre pour la mise à jour des noeuds de contrôle. L'architecture associée à ce type d'algorithme est illustrée figure 1.19. Dans une première étape, les métriques aller sont calculées et stockées dans une mémoire de profondeur égale au degré du noeud. Dans un second temps, les métriques retour sont estimées et combinées avec les métriques aller pour obtenir les messages de sortie du noeud.

L'opérateur représentant la fonction de décodage d'une relation de parité entre deux entrées et une sortie sera décrit par la fonction mathématique $g(x, y)$. Dans le cas par exemple d'un algorithme BP, la fonction $g(x, y)$ peut être réalisée à partir de

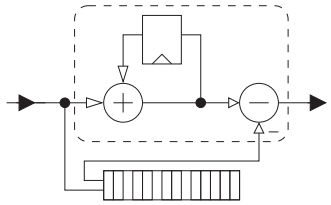
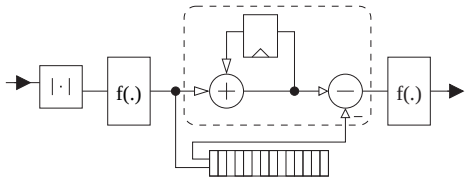
VNP	CNP
	
$m_{vc}^{i} = -m_{cv}^{i-1} + y_v^0 + \sum_{c' \in C_v} m_{c'v}^{i-1}$	$\lvert m_{cv}^{i} \rvert = f\left(-f\left(\lvert m_{vc}^{i} \rvert\right) + \sum_{v' \in V_c} f\left(\lvert m_{v'c}^{i} \rvert\right)\right)$

FIG. 1.18 – Exemple d'architecture série des processeurs associés aux noeuds de données et de contrôle. Le décodage se fait suivant l'algorithme BP. Seul le calcul des fiabilités des messages de sorties du noeud de contrôle est illustré.

la relation suivante :

$$g(x, y) = \text{sign}(x)\ \text{sign}(y)\ f\left(f\left(|x|\right) + f\left(|y|\right)\right) \tag{1.43}$$

Les approximations de type Min-Sum, offset Min-Sum ou normalized Min-Sum sont aussi très facilement réalisables. Un exemple de réalisation utilisant ce dernier type d'approximation est illustré dans [68]. Ce type d'architecture permet une réalisation très flexible du processeur CNP à condition que la profondeur des mémoires soit bien déterminée. Cependant, comme mentionné dans [68], le dimensionnement du processeur CNP pour le pire cas peut demander une assez grande complexité de l'opérateur. Cette complexité peut devenir critique quand ce processeur est dupliqué un certain nombre de fois.

Cette architecture série à base de représentation en treillis a été également proposée dans [73]. Le code LDPC est alors représenté comme une concaténation parallèle de codes de parité représentés par un treillis et séparés par un ensemble d'entrelaceur.

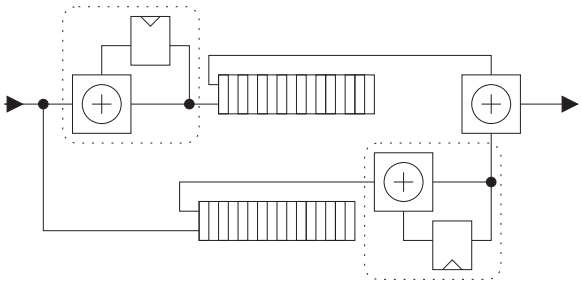


FIG. 1.19 – Exemple d'architecture série associée à un décodage suivant un treillis.

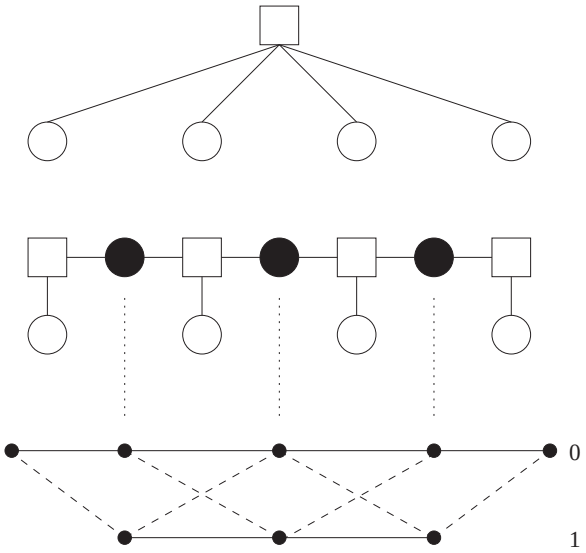


FIG. 1.20 – Représentation en treillis d'une équation de parité. La première étape de la transformation consiste en l'introduction de noeuds cachés (noeuds noirs) qui représentent l'état du treillis. Finalement la représentation en treillis est déduite. Chaque branche représente l'état d'un noeud de données ("1" trait pointillé, "0" trait plein).

Cette représentation est exactement celle d'un Turbo-code parallèle avec, pour code constituant, un code de parité. L'auteur introduit donc le concept de *turbo-decoding message-passing*. Ce séquencement est la combinaison d'un ordonnancement *shuffle*, avec une mise à jour des noeuds de contrôle réalisée par une architecture série basée sur un treillis.

1.5 Conclusion

Dans ce premier chapitre nous avons présenté les éléments de base relatifs aux techniques de codage canal et plus particulièrement aux codes de type LDPC. Nous avons proposé un bref état de l'art sur la construction, le décodage et les architectures relatives aux codes LDPC. Nous avons insisté sur les points que nous considérons comme essentiels pour la bonne compréhension de la suite du mémoire. Nous avons pu montrer qu'il existait un lien très fort entre les concepts relatifs aux Turbo-codes et codes LDPC. Ainsi un grand nombre de travaux sur la réalisation de turbo décodeurs peut servir de base pour l'étude et la réalisation de décodeurs LDPC. Dans la suite de ce mémoire, nous nous intéresserons tout particulièrement aux codes LDPC structurés, qui offrent à nos yeux un excellent compromis entre performance et simplicité de réalisation. Le chapitre suivant s'intéresse donc à cette famille de codes.

Chapitre 2

Codes LDPC structurés

Résumé

Ce deuxième chapitre introduit et développe les spécificités des codes LDPC structurés et plus particulièrement des codes de type Repeat Accumulate. Dans un premier temps, les considérations qui nous ont conduits à étudier ce type de codes sont détaillées. Dans un second temps, à partir des résultats sur la détection de cycles dans le graphe du code, et l'analyse des poids des mots de codes générés par des mots d'information de poids faible, un algorithme de construction de codes est présenté. Dans une dernière partie, deux séquencements de décodage en relation avec les spécificités du code sont présentés.

2.1 Étude d'une structure particulière

Le chapitre précédent a brièvement illustré l'intérêt des codes LDPC structurés. Tout d'abord, les codes structurés permettent un encodage simple et linéaire : la structure du code est en partie choisie dans ce but. L'existence d'une structure permet aussi une caractérisation simple du code. Tandis que, dans le cas d'une construction aléatoire, l'ensemble des positions des éléments non nuls de la matrice doit être mémorisé, seul un nombre réduit de paramètres doit être stocké dans le cas de codes structurés. Nous verrons par la suite que l'existence d'une structure dans le code a également un intérêt pour la définition de décodeurs, qui pourront être optimisés suivant certains critères (parallélisation ...).

2.1.1 Motivations

Bien que les constructions de code aléatoires soient les plus performantes dans le cas asymptotique, dans le cas de codes de longueurs finies certaines règles doivent être respectées. Une partie de ces règles ont été énoncées par Richardson *et al.* dans [19].

Les premières règles décrites concernent la distribution des degrés des noeuds. Les auteurs de [19] ont proposé une distribution de l'irrégularité des noeuds de contrôle dit *sous forme concentrée*. Ils ont montré que cette distribution permet d'obtenir de très bonnes performances et améliore la vitesse de convergence du code [49]. La distribution sous forme concentrée peut s'exprimer par :

$$\rho(x) = \rho_i x^{i-1} + (1 - \rho_i)x^i \tag{2.1}$$

Le faible nombre de degrés différents est aussi intéressant au stade de la réalisation matérielle de décodeurs, notamment dans le cas de la définition des processeurs CNP parallèle (cf Chapitre 1).

Concernant le degré des noeuds de données, l'importance des noeuds de degré 2 dans le décodage itératif des codes LDPC a été démontrée. D'une part, pour approcher au plus près la capacité, une proportion importante de noeuds de degré 2 est nécessaire [19]. D'autre part, il a été démontré que ces noeuds avaient une influence dans le phénomène d'*error floor* des codes, influence d'autant plus grande que leur proportion est élevée [74]. Ces deux phénomènes peuvent être incompatibles et donc, le nombre de noeuds de degré 2 doit être ajusté finement. Plus particulièrement, on peut noter qu'il existe des familles de codes LDPC irréguliers ayant de bonnes performances dont le nombre de noeuds de degré 2 est inférieur ou égal à M [75].

Une fois les distributions des degrés fixées, la matrice de contrôle de parité peut être construite. Une règle de construction largement utilisée consiste à construire un graphe, associé au code, sans cycles de faibles longueurs. Plus particulièrment, les cycles de faible longueur faisant intervenir des noeuds de données de degré faible doivent être proscrits [19]. En effet l'optimalité de l'algorithme de décodage est conditionnée par l'indépendance des messages entrant dans les noeuds. Cette indépendance est strictement respectée dans le cas d'un graphe sans cycle. Quand des cycles existent, leur influence sur l'indépendance des messages sera d'autant plus importante que le cycle est de faible longueur et que les noeuds de données intervenant dans celui ci sont de faible degré. Outre cet effet, la présence de cycles faisant intervenir des noeuds de faible degré peut introduire des mots de code et des pseudo-mots de code[(1)] de poids faibles. En effet, si par exemple des noeuds de données de degré 2 sont connectés entre eux dans un cycle de longueur $2l$, alors la distance minimale du code est bornée par l. Dans l'exemple illustré sur la figure 2.1 le code défini par le graphe a une distance minimale bornée par 4. Si la valeur binaire 1 est associée à chaque noeud de degré 2, toutes les équations de parité sont vérifiées : il s'agit donc d'un mot de code.

Remarque : Ces constatations sont à l'origine de l'algorithme de construction proposé dans [76]. La méthode introduite propose de construire un code de manière à éviter les cycles de faibles longueurs connectés à des noeuds de données de faibles degrés.

[(1)] la notion de pseudo-mot de code sera introduite par la suite

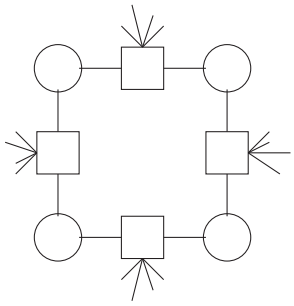


FIG. 2.1 – Exemple d'un cycle de longueur 8 faisant intervenir quatre noeuds de donnés de degré 2 entraînant un mot de poids 4.

Cette optimisation est rendue possible par l'introduction d'une métrique appelée "Approximate Cycle Extrinsic message degree" (ACE). La métrique ACE pour un cycle de longueur $2l$ est une fonction linéaire de la somme des degrés des noeuds de données.

Cet ensemble de règles suggère ainsi la connexion des noeuds de données de degré 2 dans un seul et même cycle [19, 74]. Une matrice de contrôle de parité de forme bi-diagonale appartient à l'ensemble des constructions respectant ces règles. La proportion de noeuds de degré 2 est strictement égale à M et ces noeuds sont connectés entre eux sous la forme d'une chaîne (cf figure 2.2). La recherche de codes satisfaisant l'ensemble de ces règles nous conduit à considérer une classe de codes structurés, les codes LDPC de type Repeat Accumulate.

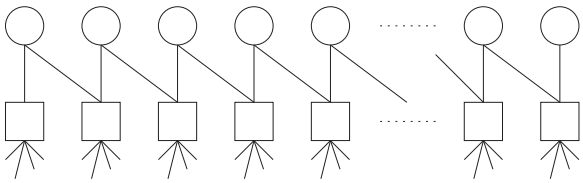


FIG. 2.2 – Exemple de noeuds de degré 2 connectés sous forme d'une chaîne.

2.1.2 Définition d'une structure

Suite aux observations précédentes, nous nous focalisons sur des codes LDPC de type Repeat Accumulate. Nous rappelons dans un premier temps quelques notations et contraintes liées à ce type de construction.

2.1.2.1 Construction de code de type Repeat Accumulate : définitions et contraintes

La matrice de contrôle de parité d'un code de type Repeat Accumulate peut s'écrire sous la forme :

$$\mathbf{H} = [\mathbf{H_s}\ \mathbf{H_p}]$$

où la matrice $\mathbf{H_p}$ est une matrice de taille $M \times M$ de forme bi-diagonale. Le mot de code $\underline{x}$ est alors divisé en un mot d'information $\underline{c}$ et un mot de redondance $\underline{p}$. La relation de parité s'écrit alors :

$$\mathbf{H_p}\underline{p}^t = \mathbf{H_s}\underline{c}^t$$

Pour simplifier les notations, nous appellerons vecteur de projection le vecteur $\underline{v}$ défini par :

$$\underline{v}^t = \mathbf{H_s}\underline{c}^t$$

Il en résulte donc que l'ensemble des bits de parité peut se déduire de la façon suivante :

$$\underline{p}^t = \mathbf{H_p}^{-1}\underline{v}^t$$

La structure de la matrice de contrôle de parité impose certaines contraintes sur la distribution des degrés. Ainsi la proportion de noeuds de données de degré 2 est caractérisée par :

$$\tilde{\lambda_2} \geq 1 - R \tag{2.2}$$

Quand il y a égalité, la condition de stabilité est respectée [74][(2)]. Par souci d'allegement des notations, nous considérerons que le nombre de noeuds de données de degré 2 est égal à M. Il faut noter que dans le cas d'une matrice bi-diagonale, une colonne comporte un seul élément non nul.

Par hypothèse nous nous intéresserons aux codes dont la distribution des degrés des noeuds de contrôle est sous une forme strictement concentrée. Le nombre de 1 dans chaque ligne de la matrice $\mathbf{H_s}$ sera considéré comme constant et égal à J. En utilisant ces hypothèses, on peut montrer que la distribution du nombre d'éléments non nuls par colonne de $\mathbf{H_s}$ suit la relation suivante :

$$\sum_{i \geq 1} \tilde{\lambda_i^s} i = \frac{1-R}{R} J \tag{2.3}$$

où $\tilde{\lambda_i^s}$ correspond au nombre moyen de noeuds de degré i dans le graphe associé à la matrice $\mathbf{H_s}$.

Nous définirons des codes de cette famille comme des codes *réguliers* si le nombre d'éléments non nuls par colonne de $\mathbf{H_s}$ est strictement constant. Nous appellerons code *irrégulier* de type (q, x) l'ensemble des codes dont le graphe défini parla matrice H_s comporte x degrés différents des noeuds de données et où q est le plus petit degré.

□ *Exemple:* On considère la famille des codes irréguliers de type $(q, 2)$. $\mathbf{H_s}$ a donc des colonnes ayant q et y éléments non nuls. La distribution des degrés s'exprime donc :

$$\begin{array}{ccccc} \tilde{\lambda_q^s} & + & \tilde{\lambda_y^s} & = & 1 \\ q\lambda_q^s & + & y\lambda_y^s & = & \frac{1-R}{R}J \end{array}$$

(2) Ce concept introduit dans [19] contrôle le comportement de l'evolution de densité lorsque la probabilité d'erreur s'approche de zéro.

Cette relation nous permet d'écrire :

$$\begin{aligned} \tilde{\lambda}_y^s &= \frac{y - \frac{1-R}{R}J}{y-q} \\ \tilde{\lambda}_q^s &= 1 - \tilde{\lambda}_y^s \end{aligned} \tag{2.4}$$

Il existe un ensemble de solutions si les conditions suivantes sont respectées :

$$\begin{aligned} y &> q \quad \text{et} \\ y &> \frac{1-R}{R}J \end{aligned} \tag{2.5}$$

A partir de ces relations, on peut déduire la distribution d'irrégularité des noeuds de données du code dans le graphe total en utilisant le fait que :

$$\tilde{\lambda}_i = R\tilde{\lambda}_i^s$$

Un exemple de distributions pour un code de rendement $R = 1/2$, $J = 4$ et, $q = 3$ est illustré table 2.1. Les distributions d'irrégularités s'expriment :

$$\begin{aligned} \lambda(x) &= \lambda_y x^{y-1} + \lambda_3 x^2 + \lambda_2 x \\ \rho(x) &= x^{J+1} \end{aligned}$$

y	$\tilde{\lambda}_3^s$	$\tilde{\lambda}_y$	$\tilde{\lambda}_3$	$\tilde{\lambda}_2$	λ_y	λ_3	λ_2
5	1/2	1/4	1/4	1/2	5/12	1/4	1/3
6	2/3	1/6	1/3	1/2	1/3	1/3	1/3
7	3/4	1/8	3/8	1/2	7/24	3/8	1/3
8	4/5	1/10	2/5	1/2	4/15	2/5	1/3
9	5/6	1/12	5/12	1/2	1/4	5/12	1/3
10	6/7	1/14	3/7	1/2	5/21	3/7	1/3
11	7/8	1/16	7/16	1/2	11/48	7/16	1/3
12	8/9	1/18	4/9	1/2	2/9	4/9	1/3

TAB. 2.1 – Exemple de paramètres pour un code irrégulier de type $(q = 3, x = 2)$ de rendement $R = 1/2$, avec $J = 4$.

□

2.1.2.2 Caractérisation du code

Une fois les distributions et la structure du code choisies, la dernière étape de la construction du code consiste à définir les positions des éléments non nuls dans la matrice $\mathbf{H_s}$. Plus particulièrement, nous souhaitons que cette matrice $\mathbf{H_s}$ puisse être définie par un faible nombre de paramètres. Cette propriété facilitera le stockage des matrices de contrôle de parité dans un système de codage/décodage. Nous

souhaitons aussi que le code puisse être décodé avec un algorithme où l'ordonnancement est du type *shuffle* par groupe de noeuds, afin d'accélérer la convergence de décodage. Comme nous l'avons vu précédemment, les matrices de contrôles de parité définies à partir de matrices identité permutées répondent à ces contraintes. Plus particulièrement, la famille de code pour laquelle la matrice de contrôle de parité est définie à partir de matrices identité circulairement permutées est particulièrement intéressante[(3)]. La caractérisation du code est simple et autorise un ordonnancement permettant une convergence rapide du décodage itératif. Notre choix se portera donc sur cette construction pour la suite de cette étude.

La matrice $\mathbf{H_s}$ sera donc construite à partir d'une matrice de k colonnes et m lignes de matrices identité circulairement permutées de taille $z \times z$. Le paramètre z sera appelé *facteur d'expansion*[77, 78, 79]. Par définition, une matrice identité permutée circulairement de p positions, aura un élément non nul à la position p de la première ligne. La permutation se fait alors de p positions vers la droite. Il est bien entendu que toutes les permutations sont réalisées modulo la taille de la sous matrice identité. Une matrice identité circulairement permutée de p positions sera notée $\mathbf{I_p}$. Un exemple de matrice identité circulairement permutée est illustré figure 2.3.

$$\mathbf{I_1} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

FIG. 2.3 – Exemple d'une matrice identité de taille 4×4 circulairement permutée de $p = 1$ position vers la droite.

Nous proposons donc d'exprimer la matrice $\mathbf{H_s}$ à partir de matrices identité circulairement permutées et de matrices nulles. Le fait d'autoriser l'existence de matrices nulles permet d'introduire de l'irrégularité dans le code et de réduire la densité de la matrice. Une matrice nulle sera caractérisée par un coefficient de permutation négatif. La matrice $\mathbf{H_s}$ peut donc s'écrire de la manière suivante :

$$\mathbf{H_s} = \begin{bmatrix} \mathbf{I}_{\delta(0,0)} & \mathbf{I}_{\delta(0,1)} & \cdots & \mathbf{I}_{\delta(0,k-1)} \\ \mathbf{I}_{\delta(1,0)} & \mathbf{I}_{\delta(1,1)} & \cdots & \mathbf{I}_{\delta(1,k-1)} \\ \vdots & \vdots & \mathbf{I}_{\delta(i,j)} & \vdots \\ \mathbf{I}_{\delta(m-1,0)} & \mathbf{I}_{\delta(m-1,1)} & \cdots & \mathbf{I}_{\delta(m-1,k-1)} \end{bmatrix} \tag{2.6}$$

où $\delta(i,j)$ correspond au coefficient de permutation appliqué à la matrice de la ligne i et de la colonne j. Nous appelons $\mathbf{S}$ la *matrice de base* ou *masque* associée à $\mathbf{H_s}$, la

[(3)] Nous verrons par la suite que ce type de construction est bien adapté à la réalisation : l'adressage et la répartition des mémoires se trouve simplifié

matrice contenant les coefficients de permutation. Cette matrice s'exprime par :

$$\mathbf{S} = \begin{bmatrix} \delta(0,0) & \delta(0,1) & \cdots & \delta(0,k-1) \\ \delta(1,0) & \delta(1,1) & \cdots & \delta(1,k-1) \\ \vdots & \vdots & \delta(i,j) & \vdots \\ \delta(m-1,0) & \delta(m-1,1) & \cdots & \delta(m-1,k-1) \end{bmatrix} \tag{2.7}$$

□ *Exemple:* Nous proposons de construire la matrice $\mathbf{H_s}$ à partir d'un facteur d'expansion $z = 3$ et la matrice de base suivante :

$$\mathbf{S} = \begin{bmatrix} 1 & -1 \\ 2 & 0 \end{bmatrix}$$

A partir de cette représentation, nous pouvons construire la matrice $\mathbf{H_s}$ de taille 6×6 :

$$\mathbf{H_s} = \left[\begin{array}{ccc|ccc} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right]$$

□

La définition d'une telle structure pour $\mathbf{H_s}$ oblige à redéfinir la matrice $\mathbf{H_p}$. En effet, si $\mathbf{H_p}$ est une matrice bi-diagonale telle que présentée précédemment, le code défini souffre d'une très mauvaise distance minimale. Pour mettre en évidence cette propriété, on considère un mot d'information de poids 2 avec deux éléments successifs égaux à un. On considère aussi un code régulier de distribution $\tilde{\lambda}^s(x) = x^{q-1}$ et un matrice $\mathbf{H_s}$ de taille $qz \times z$ (cette matrice est donc une matrice colonne de matrices identité circulairement permutées). Le vecteur de projection, caractérisé par la projection du mot d'information sur la matrice $\mathbf{H_s}$, a donc $2q$ éléments non nuls. Dans certains cas, le vecteur de projection se compose donc de q couples "11"qui génèrent un mot de poids q après codage par l'accumulateur. En d'autres termes, la distance minimale d'une telle structure est bornée par $2 + q$, ce qui est relativement faible si on considère que q peut prendre de faibles valeurs (3 par exemple). De plus, la structure telle que définie précédemment ne permet pas facilement un décodage avec un ordonnancement de type *shuffle* par groupe de z noeuds. Si on considère z lignes successives de la matrice $\mathbf{H_p}$, le nombre d'éléments non nuls par colonne est égal à 2 ce qui ne permet un décodage de type *shuffle*. Ainsi les avantages d'une construction par expansion ne peuvent pas être exploités avec cette structure.

Nous proposons donc de redéfinir la matrice $\mathbf{H_p}$ à partir de matrices identité circulairement permutées. Nous proposons donc d'utiliser la représentation suivante

pour la matrice $\mathbf{H_p}$:

$$\mathbf{H_p} = \begin{bmatrix} \mathbf{I} & & & & \mathbf{I}_x \\ \mathbf{I} & \mathbf{I} & & & \\ & \mathbf{I} & \ddots & & \\ & & \ddots & \mathbf{I} & \\ \mathbf{0} & & & \mathbf{I} & \mathbf{I} \end{bmatrix} \tag{2.8}$$

où $\mathbf{H_p}$ est une matrice de type bi-diagonale de matrices identité permutées. Pour garder les propriétés d'encodage, il est nécessaire que cette nouvelle matrice soit inversible. Il faut donc qu'au moins une colonne n'ait qu'un seul élément non nul. De plus nous imposons la contrainte que l'ensemble des noeuds de degré 2 ne soient pas connectés dans un même cycle. Ces deux propriétés nous conduisent donc à définir $\mathbf{H_p}$ par :

$$\mathbf{H_p} = \begin{bmatrix} \mathbf{I} & & & & \mathbf{I'}_1 \\ \mathbf{I} & \mathbf{I} & & & \\ & \mathbf{I} & \ddots & & \\ & & \ddots & \mathbf{I} & \\ \mathbf{0} & & & \mathbf{I} & \mathbf{I} \end{bmatrix} \tag{2.9}$$

où $\mathbf{I'}_1$ est une matrice identité non circulairement permutée de une position vers la gauche :

$$\mathbf{I'}_1 = \begin{bmatrix} 0 & 0 & & 0 & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & & 0 & 0 & 0 \\ & & \ddots & & \vdots & \\ 0 & 0 & & 1 & 0 & 0 \\ 0 & 0 & & 0 & 1 & 0 \end{bmatrix} \tag{2.10}$$

Il faut noter que cette nouvelle expression de la matrice $\mathbf{H_p}$ est identique à celle décrite précédemment à un entrelacement près. Ce type de structure a été aussi proposée dans [53] où ce type de code est appelé *Quasi-Cyclic Repeat Accumulate*. Cette famille de code a également été proposée dans [69] dans un but de simplification de réalisation.

La structure de la matrice de contrôle de parité étant construite par blocs, de nouvelles contraintes apparaissent sur la distribution des noeuds. En effet, une première contrainte impose que m soit supérieur au degré maximum des noeuds de données. La seconde stipule que les produits $\tilde{\lambda}_i(m+k)$ soient des nombres entiers. La granularité du nombre de noeud d'un certain degré est alors multiple de z. Ces deux règles contraignent et réduisent l'ensemble de cette famille de code. A titre d'exemple, si on considère les distributions de la table 2.1, on peut déduire la plus petite valeur ainsi que l'ensemble des paramètres m possibles. La table 2.2 illustre l'ensemble de ces paramètres.

y	$\tilde{\lambda}_3^s$	$\tilde{\lambda}_y$	$\tilde{\lambda}_3$	$\tilde{\lambda}_2$	$m_{\min}$	m
5	1/2	1/4	1/4	1/2	6	$2k$
6	2/3	1/6	1/3	1/2	6	$3k$
7	3/4	1/8	3/8	1/2	8	$4k$
8	4/5	1/10	2/5	1/2	10	$5k$
9	5/6	1/12	5/12	1/2	12	$6k$
10	6/7	1/14	3/7	1/2	14	$7k$
11	7/8	1/16	7/16	1/2	16	$8k$
12	8/9	1/18	4/9	1/2	18	$9k$

TAB. 2.2 – Illustration du choix du paramètre m en fonction des distributions des degrés.

Dans un souci de synthèse, la figure 2.4 illustre le chemin par lequel nous avons été amené à considérer la famille de codes proposée. Dans la suite de ce manuscrit, nous nous intéresserons donc à cette structure de code qui peut être vue, à un entrelacement près, comme des codes Repeat Accumulate. Une analyse de cette structure est proposée dans la section suivante. Cette analyse a pour but de définir des algorithmes de construction de code.

2.2 Analyse de la structure étudiée

Nous avons précédemment décrit la structure de code choisie. Le choix d'une telle famille de codes résulte de considérations de performances et de réalisations matérielles. Avant de s'interroger sur la réalisation proprement dite d'un schéma de décodage, nous nous focalisons sur la construction de codes ou, en d'autres termes comment sélectionner les paramètres et les codes les plus appropriés parmi un ensemble prédéterminé. Nous proposons dans un premier temps de nous intéresser aux propriétés de distance de Hamming du code. Dans un deuxième temps, une étude sur la distribution des cycles dans le graphe du code sera proposée. A partir de ces observations, un algorithme de construction de code prenant en compte les spécificités de la structure sera proposé. Il est important de noter que cette partie n'a pas la prétention d'illustrer une analyse théorique et rigoureuse de la structure de code mais de donner aux lecteurs les points clefs dans la compréhension de certains phénomènes qui amèneront à la définition d'un algorithme de construction de codes.

2.2.1 Propriétés de distance

Les propriétés de distance de la structure proposée sont intéressantes à étudier pour bien appréhender le comportement du code. Tout d'abord, nous rappelons que la structure proposée peut être vue comme la concaténation série d'un code de parité et d'un accumulateur. Les propriétés de distance de l'accumulateur sont bien connues. Ainsi, la fonction énumératrice des poids de sortie en fonction des poids d'entrée

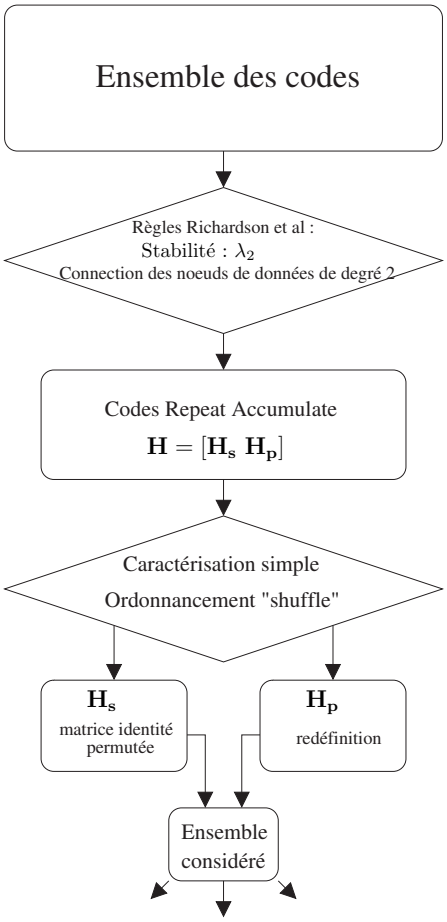


FIG. 2.4 – Synthèse de la méthode et des arguments en faveur de la structure proposée.

(IOWEF, Input-Output Weight Enumerator) s'exprime par la relation [41] :

$$A_{w,h}^{acc} = \begin{pmatrix} N-K-h \\ \lfloor w/2 \rfloor \end{pmatrix} \begin{pmatrix} h-1 \\ \lceil w/2 \rceil - 1 \end{pmatrix} \tag{2.11}$$

où

$$\begin{pmatrix} x \\ y \end{pmatrix} = \frac{x!}{(x-y)!y!} \tag{2.12}$$

dans laquelle w représente le poids du mot d'entrée et h le poids du mot de sortie. La densité de probabilité associée à cette fonction énumératrice peut donc en être déduite. Cette densité est illustrée dans le cas d'un code accumulateur de taille $N = 1000$ sur la figure 2.5. Cette densité de probabilité montre que les mots de poids faibles seront **statistiquement** générés par des mots d'entrée de poids faibles.

L'expression de la fonction IOWEF de cette famille de codes peut être déduite en utilisant le concept d'entrelaceur uniforme [80] appliquée à une concatenation série

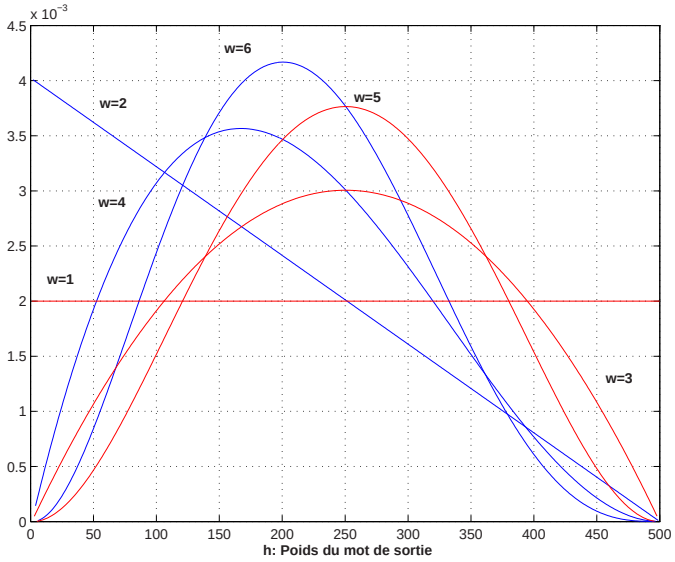


FIG. 2.5 – Illustration de la densité de probabilité relative à la fonction IOWEF d'un accumulateur pour une longueur $N = 1000$.

[15] :

$$A_{w,h} = \sum_{l=0}^{M} \frac{A^0_{w,l} A^{acc}_{l,h}}{\binom{M}{l}} \tag{2.13}$$

où $A^0_{w,l}$ correspond à la fonction d'IOWEF du code de parité défini par $\mathbf{H_s}$. Le concept d'entrelaceur uniforme consiste à répartir un mot donné de taille M et de poids l en $\binom{M}{l}$ permutations distinctes avec une probabilité uniforme de $P = 1/\binom{M}{l}$

L'expression de la fonction d'IOWEF $A^0_{w,l}$ correspondant au code de parité, peut s'évaluer asymptotiquement [81] pour une distribution de degré donnée, où statistiquement si certaines structures le permettent [57, 58]. Une conclusion commune de ces analyses montre que les mots de poids faibles sont **statistiquement** générés par des mots d'information de poids faibles. Il est important de noter que ces études sont des études statistiques et reflètent donc le comportement moyen d'une famille et non d'un code donné.

Cette propriété statistique nous amène à étudier le comportement de la famille de codes proposée dans le cas où des mots d'information de poids faibles sont codés. Dans un premier temps, quelques concepts et notations sont introduits puis nous analyserons le comportement de la famille de codes quand des mots de poids un et deux sont codés.

2.2.1.1 Notations et concepts

On considère un mot d'information $\underline{c}$ de dimension z et de poids unitaire. Le vecteur de projection $\underline{v}$ issu de la projection de $\underline{c}$ sur une matrice identité circulairement permutée de δ positions est aussi un vecteur de poids unitaire. Si l'élément non nul dans $\underline{c}$ est à la position j, l'élément non nul dans $\underline{v}$ est en position $(z-\delta+j)\text{mod } z$.

Soit un vecteur de projection $\underline{v}$ de dimension mz issu de la projection de $\underline{c}$ de dimension kz sur la matrice $\mathbf{H_s}$ de taille $mz \times kz$. Un élément de $\underline{v}$ en position j correspond à la $i^{\text{ème}}$ entrée de l'accumulateur. Cette transformation se caractérise par la relation suivante :

$$i = m\,(j\text{mod } z) + \left\lfloor \frac{j}{z} \right\rfloor \tag{2.14}$$

Cette transformation peut être interprétée comme un simple entrelacement ligne-colonne du vecteur de projection.

Le poids en sortie de l'accumulateur peut se déduire à partir des positions des éléments non nuls du vecteur d'entrée. En effet, le poids en sortie peut être très facilement calculé en utilisant le concept de séquence de retour à zéro (RTZ, Return To Zero) [82]. Une séquence RTZ est définie par une liste de symboles faisant dans un premier temps diverger le codeur de l'état 0, puis dans un deuxième temps, le faisant revenir à l'état nul. La séquence RTZ de l'accumulateur est une séquence de poids 2 de la forme suivante :

$$...0\mathbf{1}0...0\mathbf{1}0...$$

Le poids du mot en sortie de l'accumulateur peut donc se calculer en exprimant la différence entre deux postions d'éléments non nuls successifs. Un exemple est illustré sur la figure 2.6

Position	.	.	.	j	$j+1$	$j+2$	$j+3$	.	.
Entré	.	0	0	1	0	0	1	0	.
Sortie	.	0	0	1	1	1	0	0	.

FIG. 2.6 – Exemple de calcul du poids de sortie en fonction des positions non nulles du mot présentées à l'entrée de l'accumulateur. Le mot codé est dans ce cas de poids 3.

2.2.1.2 Analyse des distances de Hamming issues du codage d'un mot de poids 1

Le fait de s'intéresser aux distances de Hamming issues du codage d'un mot de poids 1 nous permet de considérer une matrice $\mathbf{H_s}$ de taille $mz \times z$. Par hypothèse, cette matrice est construite à partir de q matrices identité permutées. La matrice $\mathbf{H_s}$ peut être définie à partir de deux ensembles :

- un ensemble Δ caractérisant les q coefficients de permutation : $\Delta := \{\delta_k, k = 0 \dots q-1\}$
- un ensemble Ω caractérisant la position des q coefficients de permutation δ_k dans la matrice de base $\mathbf{S}$: $\Omega := \{P_k, k = 0 \dots q-1\}$

Le poids du vecteur de projection issu de la multiplication d'un vecteur $\underline{c}$ de poids unitaire par la matrice $\mathbf{H_s}$ est donc égal à q. Les q éléments non nuls dans $\underline{v}$ sont en positions :

$$zP_k + (j + z - \delta_k)\text{mod } z, \quad \forall k = 0 \dots q-1, \quad \forall j = 0 \dots z-1 \tag{2.15}$$

En utilisant la relation de transformation (équation 2.14), on peut déduire la position des éléments non nuls à l'entrée de l'accumulateur :

$$P_k + m(j + z - \delta_k)\text{mod } z, \quad \forall k = 0 \dots q-1, \quad \forall j = 0 \dots z-1 \tag{2.16}$$

Cette relation nous indique la présence d'un élément non nul à la position P_k du $(j + z - \delta_k)$mod z groupe de m bits. Le poids de sortie de l'accumulateur est la différence entre les positions des éléments non nuls du vecteur d'entrée. Si on considère $q = 2$, la différence entre deux positions u et v d'éléments non nuls successifs dans la séquence d'entrée de l'accumulateur, est égale à :

$$P_u - P_v + m\left[(j - \delta_u) \bmod z - (j - \delta_v) \bmod z\right] \tag{2.17}$$

où P_u,P_v et δ_u,δ_v sont la caractérisation des deux positions successives u et v. Tout d'abord, en remarquant que quand :

$$(j - \delta_u) \bmod z - (j - \delta_v) \bmod z > 0 \tag{2.18}$$

ce qui revient à dire que les coefficients δ_u et δ_v sont différents, on peut minorer ce terme par :

$$(j - \delta_u) \bmod z - (j - \delta_v) \bmod z \geq \gamma, \quad \forall j = 1 \dots z-1 \tag{2.19}$$

où :

$$\gamma = \min\left((\delta_u - \delta_v) \bmod z; (\delta_v - \delta_u) \bmod z\right) \tag{2.20}$$

De plus, en notant que :

$$-m + 1 \leq P_u - P_v \leq m - 1 \tag{2.21}$$

on peut minorer le poids de sortie de l'accumulateur w_1 par :

$$w_1 \geq 1 + m(\gamma - 1) \tag{2.22}$$

Dans un cas plus général, deux situations sont à distinguer, le cas où q est pair et le cas où q est impair.

Cas q pair :
Le cas où q est pair est assez simple à analyser. Le poids du mot en sortie de l'accumulateur est la différence, deux à deux, des positions des éléments non nuls d'entrée. En particulier, le plus petit poids du mot en sortie de l'accumulateur, w_1, peut être minoré par :

$$w_1 \geq \frac{q}{2}\left[(\gamma_{\min} - 1)m + 1\right] \tag{2.23}$$

dans le cas où l'ensemble des coefficients δ_k sont différents. $\gamma_{\min}$ représente la plus petite différence entre deux coefficients de permutation, où la distance entre deux coefficients s'exprime par :

$$\min\left((\delta_u - \delta_v) \bmod z; (\delta_v - \delta_u) \bmod z\right) \tag{2.24}$$

Cas q impair :
Quand q est impair, le poids du mot en sortie de l'accumulateur est la différence, deux à deux, des positions des éléments non nuls d'entrée plus un terme correspondant à la propagation du dernier élément non nul de la séquence (dans ce cas un élément à "1"reste dans la bascule). En reprenant le raisonnement décrit précédemment, le plus petit poids de sortie est généré par une combinaison où un terme $(j + z - \delta_k) \bmod z$ est égal à $z - 1$ (un élément non nul se trouve donc dans le dernier groupe de m bits à l'entrée de l'accumulateur). A partir de cette observation, nous pouvons déduire un minorant du plus petit poids de sortie de l'accumulateur w_1 par :

$$w_1 \geq 1 + \frac{q-1}{2}\left[(\gamma - 1)m + 1\right] \tag{2.25}$$

sous la condition que les coefficients δ_k soient différents.

Cette première analyse de distance nous montre l'importance du choix des paramètres du code. Ainsi, une attention particulière doit être prise dans le choix des coefficients de permutation dans une même colonne de la matrice de base $\mathbf{S}$, de manière à s'affranchir des mots de poids faibles générés par un mot d'information de poids unitaire. Nous nous attacherons donc à construire des codes où les coefficients de permutation positifs dans un colonne de la matrice de base $\mathbf{S}$ sont équi-répartis dans l'intervalle $[0, z-1]$ afin de garantir une valeur γ proche du rapport z/q. Cette contrainte sera d'autant plus importante que le nombre q sera petit et impair. Le respect de ces règles permet de minimiser la probabilité qu'un mot de poids faible soit généré par un mot d'information de poids faible.

2.2.1.3 Analyse des distances de Hamming issues du codage d'un mot de poids 2

L'analyse des distances issues du codage d'un mot de poids 2 est aussi intéressante à étudier pour comprendre le comportement de cette famille de codes. Dans le cas

de mot d'entrée de poids 2, deux cas sont à distinguer. Le premier cas correspond à un mot de poids 2 dont les éléments non nuls sont projetés à travers deux ensembles différents de matrices identité permutées. Le deuxième cas correspond donc à un mot de poids 2 dont les éléments non nuls sont projetés à travers le même ensemble de matrices identité permutées. Nous nous intéressons dans un premier temps sur ce dernier cas.

Soit une matrice $\mathbf{H_s}$ de taille $mz \times z$. Par hypothèse, cette matrice est construite à partir de q matrices identité permutées. La projection d'un mot de poids 2 à travers cette matrice, nous amène à considérer un ensemble de coefficients de permutation Δ défini par :

$$\Delta = \{(\delta_k + i)\,;(\delta_k + j)\}\,,\ i \neq j,\ \forall k = 0 \ldots q \tag{2.26}$$

A partir de la définition de cet ensemble plusieurs cas sont à considérer. Tout d'abord si les éléments de l'ensemble Δ sont différents, alors le plus petit poids de sortie est inférieur ou égal à mq. Si on considère $j = i+1$ alors les éléments non nuls à l'entrée de l'accumulateur sont distants de m positions. Cette première constatation nous montre que le plus petit poids de sortie de l'accumulateur généré par un mot d'information de poids 2, w_2, est bornée par :

$$w_2 \leq mq \tag{2.27}$$

La distance minimale d'une telle structure sera donc bornée par :

$$d_{\min} \leq 2 + mq \tag{2.28}$$

Contrairement au cas des distances de Hamming issues du codage d'un mot de poids 1, des règles sur les coefficients de permutation ne permettent pas d'obtenir une distance relativement grande. Ce résultat est donc très important puisqu'il donne une contrainte sur le choix du paramètre m en fonction de l'irrégularité du code. Le paramètre q pouvant être petit, m devra être choisi suffisamment grand pour garantir une certaine distance minimale.

Cette relation nous permet aussi de conclure que cette famille de codes n'est pas asymptotiquement bonne si la taille du code augmente linéairement avec le facteur d'expansion z[4]. Cela ne veut pas dire pour autant que cette famille de codes est mauvaise. En effet, on peut trouver un paramètre m, choisi en fonction de la taille, du rendement, et du profil d'irrégularité du code, qui peut satisfaire aux contraintes d'une application donnée. Par exemple, dans le cas de systèmes de type Hybrid Automatic Repeat Request (H-ARQ)[5] le taux d'erreur trame requis pour le fonctionnement du système est de l'ordre de 10^{-4}. Dans ce cas, un code avec une grande distance minimale n'est pas forcément requis. A l'opposé un système de diffusion par satellite (ex DVB-S2) requiert des taux d'erreur très faibles (on parle de région *quasi error free*,

[4] ce qui est le cas pour les codes retenus pour la norme IEEE 802.11n et 802.16e

[5] Ce protocole consiste à envoyer lors d'une retransmission, des informations de parité supplémentaires, c'est-à-dire qui n'avaient pas été envoyées lors des transmissions précédentes.

BER $\simeq 10^{-11}$), nécessitant donc un code avec une grande distance minimale.

L'existence de cette borne pose la question de comment l'atteindre. Il faut dans un premier temps que le plus petit poids de mots de code généré par un mot d'information de poids 1, w_1 soit supérieur à cette borne. Si l'on veut que le plus petit poids w_1 soit supérieur à la borne $2 + mq$, on a alors la relation suivante sur la métrique γ :

$$\gamma > 3 + \frac{2}{mq} - \frac{1}{m}, \quad \text{quand } q \text{ pair} \tag{2.29}$$

$$\gamma > 1 + \frac{2q}{q-1} - \frac{1}{m}, \quad \text{quand } q \text{ impair} \tag{2.30}$$

Cette relation garantit que les poids des mots issus du codage d'un mot de poids 1 ont peu d'influence sur les premiers éléments du spectre des distances, et donc sur les performances à haut rapport signal à bruit. D'autre part, le cas où $j = i + 1$ est à considerer avec attention. Dans ce cas, l'ensemble Δ est égale à :

$$\Delta = \{(\delta_k + i)\,;(\delta_k + i + 1)\}\,,\ \forall k = 0 \ldots q \tag{2.31}$$

Une condition pour atteindre la borne est que l'ensemble de ces coefficients soient différents. Autrement dit, la plus petite distance γ entre deux coefficients parmi les q doit être strictement supérieure à 1. On peut remarquer que cette règle est redondante avec celles établies pour garantir un certain poids w_1. Enfin, la borne définie précédemment peut être atteinte dans le cas où d'un mot de poids 2 dont les éléments non nuls sont projetés à travers deux ensembles différents Δ^0 et Δ^1 de matrices identité permutées, si les deux ensembles sont choisis conjointement. Il faut donc construire l'ensemble Δ^1 en prenant en compte l'ensemble Δ^0. Quand q est petit il est possible d'écrire un algorithme basé sur les résultats précédents permettant d'atteindre la borne. Cependant le problème devenant vite extrêmement complexe quand q devient grand, ce type d'approche ne sera possible que pour des valeurs faibles de q.

En pratique, nous nous attacherons à garantir une certaine distance entre les coefficients de permutations dans une colonne de la matrice de base. Une manière simple de garantir cette distance est d'équi-repartir les coefficients dans l'intervalle $[0, z-1]$, ce qui revient à définir les q coefficients par la relation $\delta_k = \rho + kq/z$. L'application stricte de cette relation ne permet pas d'atteindre la borne définie. En effet si on considère l'ensemble Δ, et deux éléments non nuls dans le mot d'information à coder en position i et $i = j + q/z$, il existe alors seulement q éléments différents dans Δ. Cette propriété peut entraîner dans certains cas un poids de sortie de l'accumulateur inférieur à mq. En conclusion, il est donc nécessaire que les q coefficients δ_k soient suffisamment équi-réparties dans l'intervalle $[0, z-1]$, sans l'être strictement.

2.2.1.4 Synthèse de l'analyse des distances de Hamming issus du codage de mots de poids faibles

Des règles sur les coefficients de permutation ont été dérivées pour garantir un poids minimum de mots de code issu du codage de mots de poids faibles. Lors de la

construction d'un code, une attention particulière sera donc prise pour garantir une distance suffisante entre les coefficients de permutation. En particulier, lorsque q est petit, on s'attachera à choisir un paramètre m et un espacement entre les coefficients suffisamment grand. Cette étude a également montré l'importance du choix du paramètre m du code, notamment quand le paramètre q est petit. En effet le poids de sortie du codage d'un mot de poids deux est borné par ces deux paramètres. Nous verrons par la suite que l'ensemble des résultats présentés peuvent être pris en compte dans un algorithme de construction de codes.

2.2.2 Analyse des cycles dans le graphe du code

Quand des cycles de faibles longueurs sont présents dans le graphe du code, les performances de décodage en utilisant un algorithme de type BP ne sont plus optimales. Dans certains cas, la présence de cycles de faibles longueurs entraîne aussi des mots de code de poids faibles. Nous recherchons donc à analyser le comportement de la structure vis à vis de la répartition des cycles. Plus particulièrement nous nous intéressons à la définition de règles de construction pour éviter certaines configurations.

Dans le cas de structures particulières, tels que les codes QC présentés dans [51], des règles de constructions sur les coefficients de permutation des matrices identité sont proposées. Nous proposons dans ce manuscrit d'illustrer une approche géométrique pour l'analyse des cycles dans le graphe. A partir de cette analyse, un algorithme de construction de type PEG sera décrit.

2.2.2.1 Concepts et notations

Soit le repère orthonormal décrit figure 2.7 caractérisant la position d'un élément non nul dans la matrice identité de taille $z \times z$ circulairement permutée de p positions vers la droite. Les éléments non nuls dans la matrice considérée sont donc reperés par le point A du plan de coordonnées :

$$A := \begin{pmatrix} x \\ (x+p) \bmod z \end{pmatrix} := \begin{pmatrix} (y-p) \bmod z \\ y \end{pmatrix} \tag{2.32}$$

On considère un matrice $\mathbf{T}$ de taille $2z \times z$ construite à partir de deux coefficients de permutation δ_A et δ_B définie par :

$$\mathbf{T} := \begin{bmatrix} \mathbf{I}_{\delta_A} \\ \mathbf{I}_{\delta_B} \end{bmatrix} \tag{2.33}$$

On associe à chaque matrice un repère et deux points, A et B. Ces points caractérisent la position d'un élément non nul dans les matrices identité permutées tel que B soit la projection du point A sur un élément non nul de la matrice $\mathbf{I}_{\delta_\mathbf{B}}$. Cette projection est illustrée figure 2.8(a). A partir des coordonnées du point A, (x_A, y_A), on peut déduire

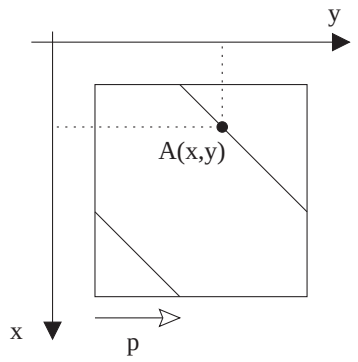


FIG. 2.7 – Illustration du repère orthonormal associé à une matrice identité permutée.

$$\mathbf{T} := \begin{bmatrix} \mathbf{I}_{\delta_A} \\ \mathbf{I}_{\delta_B} \end{bmatrix} \qquad \mathbf{T} := \begin{bmatrix} \mathbf{I}_{\delta_A} & \mathbf{I}_{\delta_B} \end{bmatrix}$$

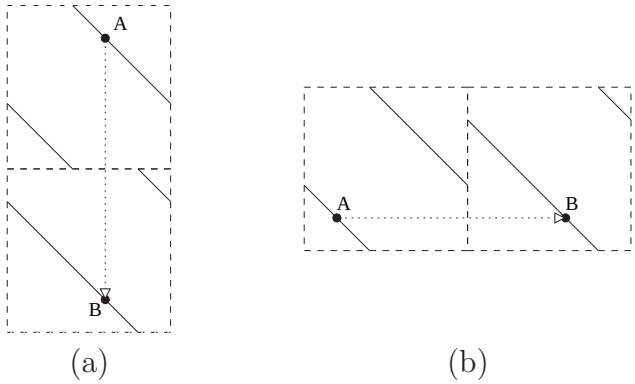


FIG. 2.8 – Projection verticale (a) et horizontale (b) du point A sur la matrice identité $\mathbf{I}_{\delta_B}$

les coordonnées du point B dans le repère qui lui est associé de la manière suivante :

$$B := \begin{pmatrix} (y_A - \delta_B) \bmod z \\ y_A \end{pmatrix} \tag{2.34}$$

On appellera la projection du point A au point B une projection verticale. De même, on considère un matrice $\mathbf{T}$ de taille $z \times 2z$ construite à partir de deux coefficients de permutation δ_A et δ_B définie par :

$$\mathbf{T} := \begin{bmatrix} \mathbf{I}_{\delta_A} & \mathbf{I}_{\delta_B} \end{bmatrix} \tag{2.35}$$

On associe à chaque matrice un repère et deux points, A et B. Ces points caractérisent la position d'un élément non nul dans les matrices identité permutées tel que B soit la projection du point A sur un élément non nul de la matrice $\mathbf{I}_{\delta_\mathbf{B}}$. Cette projection est illustrée figure 2.8(b).

A partir des coordonnées du point A, (x_A, y_A), on peut déduire les coordonnées du point B dans le repère qui lui est associé de la manière suivante :

$$B := \begin{pmatrix} x_A \\ (x_A + \delta_B) \bmod z \end{pmatrix} \tag{2.36}$$

A partir de cette représentation géométrique nous proposons d'étudier la repartition des cycles dans le graphe d'un code ayant la structure proposée.

2.2.2.2 Analyse des configurations du graphe faisant intervenir des cycles de longueur 4

Un cycle de longueur 4 intervenant dans un graphe provient d'une configuration faisant intervenir deux noeuds de données et deux noeuds de contrôles. Cette configuration correspond elle même à une disposition faisant intervenir quatre coefficients de permutation répartis dans la matrice de contrôle de parité de la manière suivante :

$$\begin{bmatrix} \mathbf{I}_{\delta_{A_0}} & \mathbf{I}_{\delta_{A_3}} \\ \mathbf{I}_{\delta_{A_1}} & \mathbf{I}_{\delta_{A_2}} \end{bmatrix} \tag{2.37}$$

Soit A_0, A_1, A_2 et A_3 les points caractérisant les éléments non nuls des matrices identité permutées $\mathbf{I}_{\delta_{A_0}}$, $\mathbf{I}_{\delta_{A_1}}$, $\mathbf{I}_{\delta_{A_2}}$ et $\mathbf{I}_{\delta_{A_3}}$ tel que le point A_{i+1} est la projection du point A_i sur un élément non nul de la matrice $\mathbf{I}_{\delta_{A_{i+1}}}$. Un exemple est illustré sur la figure 2.9.

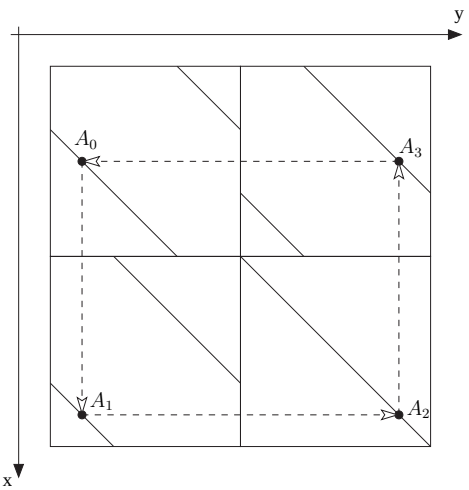


FIG. 2.9 – Exemple d'une configuration avec un cycle de longueur 4.

Ces points peuvent être caractérisés dans leur propre repère de la manière suivante :

$$\begin{aligned} A_0 &:= \begin{pmatrix} x \\ (x + \delta_{A_0}) \bmod z \end{pmatrix} & A_3 &:= \begin{pmatrix} (y_{A_3} - \delta_{A_3}) \bmod z \\ y_{A_3} \end{pmatrix} \\ A_1 &:= \begin{pmatrix} (y_{A_0} - \delta_{A_1}) \bmod z \\ y_{A_0} \end{pmatrix} & A_2 &:= \begin{pmatrix} x_{A_1} \\ (x_{A_1} + \delta_{A_2}) \bmod z \end{pmatrix} \end{aligned} \tag{2.38}$$

Il existe au moins un cycle de longueur 4 si et seulement si le point A_0 est la projection du point A_3 sur les éléments non nuls de $\mathbf{I}_{\delta_{A_0}}$ (cf figure 2.9). Autrement dit, il existe au moins un cycle de longueur 4 si et seulement si :

$$x_{A_0} = x_{A_3} \tag{2.39}$$

En utilisant les relations précédentes, on démontre qu'il existe au moins un cycle de longueur 4 si et seulement si :

$$\exists! x \in [0 \dots z-1] \text{ tel que } x = (x + \delta_{A_0} - \delta_{A_1} + \delta_{A_2} - \delta_{A_3}) \bmod z \tag{2.40}$$

Cette relation nous donne donc une contrainte sur les coefficients de permutation. En effet, il existe un cycle de longueur 4 si :

$$(\delta_{A_0} - \delta_{A_1} + \delta_{A_2} - \delta_{A_3}) \bmod z = 0 \tag{2.41}$$

Plus particulièrement dans ce cas, il existera z cycles de longueur 4. Ce résultat nous montre l'importance de construire un graphe sans cycle de longueur 4. En effet, autant la sous-optimalité de l'algorithme de décodage engendrée par un seul cycle de longueur 4 peut être négligeable, autant dans le cas de la structure considérée, cette sous-optimalité ne le sera plus au vu de l'existence d'au moins z cycles de longueur 4.

2.2.2.3 Généralisation à des cycles de longueur $2l$

En reprenant le même raisonnement que celui décrit pour l'analyse des configurations du graphe faisant intervenir des cycles de longueurs 4, nous proposons une généralisation à des configurations faisant intervenir des cycles de longueurs $2l$.

Soit $\mathbf{h}$ la matrice de base associée à la matrice de contrôle de parité $\mathbf{H}$, caractérisant chaque sous matrice de taille $z \times z$ comme une matrice identité circulairement permutée d'un certain coefficient, ou une matrice nulle. On associe à cette matrice un graphe, appelé graphe de base, qui est une représentation compressée du graphe du code. Ainsi, on associe à chaque branche du graphe de base la valeur du coefficient de permutation. Ce graphe de base peut être interprété comme le *protographe* du code [54].

□ *Exemple:* On considère une matrice de contrôle de parité $\mathbf{H}$ de taille $3z \times 4z$ et sa matrice de base associée définie par :

$$\mathbf{h} := \begin{bmatrix} 4 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 \\ 2 & -1 & 0 & 0 \end{bmatrix} \tag{2.42}$$

Le facteur d'expansion z est fixé à 8. Le graphe de base associé à ce code est illustré figure 2.10.

□

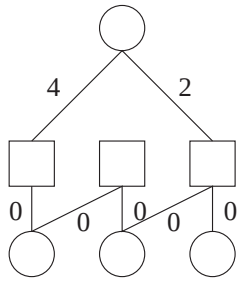


FIG. 2.10 – Illustration d'un graphe de base.

On considère un cycle de longueur $2l$ dans le graphe de base faisant intervenir une configuration de $2l$ coefficients δ_{A_i}. Soit A_i un point correspondant à la position d'un élément non nul dans la matrice $\mathbf{I}_{\delta_{A_i}}$. Par définition, on considère le point $A_{i+1}, i > 0$ comme étant la projection du point A_i sur les éléments non nuls de $\mathbf{I}_{\delta_{A_{i+1}}}$. De plus, on considère la projection du point A_{2k} vers le point A_{2k+1} comme une projection verticale. De la même manière, on considère la projection du point A_{2k+1} vers le point A_{2k+2} comme une projection horizontale. En utilisant les relations décrites précédemment, on peut déduire la position du point A_{2l-1} :

$$x_{A_{2l-1}} = \left(x + \sum_{i=0}^{2l-1} (-1)^i \, \delta_{A_i} \right) \bmod z, \forall x \in [0 \ldots z-1] \tag{2.43}$$

Il existe z cycles de longueur $2l$ dans cette configuration si et seulement si le point A_0 est la projection du point A_{2l-1} sur les éléments non nuls de la matrice $\mathbf{I}_{\delta_{A_0}}$. Il existe donc z cycles de longueur $2l$ si et seulement si :

$$\alpha = \left(\sum_{i=0}^{2l-1} (-1)^i \, \delta_{A_i} \right) \bmod z \tag{2.44}$$

Cette généralisation nous permet donc de détecter des cycles de longueur $2l$ dans le graphe, par simple étude du graphe de base et des coefficients de permutation.

Dans le cas où α n'est pas nul, il n'existe pas de cycle de longueur inférieure à $2l$ dans la configuration considérée. Dans ce cas, la projection du point A_{2l-1} sur le éléments non nuls de la matrice $\mathbf{I}_{\delta_{A_0}}$ est le point A_{2l} dont l'abscisse est égale à :

$$x_{A_{2l}} = (x + \alpha) \bmod z = 0, \forall x \in [0 \ldots z-1] \tag{2.45}$$

Il existe donc un cycle de longueur $2kl$, quand k est le plus petit entier strictement positif tel que le point A_0 soit la projection du point A_{2kl-1} sur les éléments non nuls de la matrice $\mathbf{I}_{\delta_{A_0}}$. Dans ce cas, si k est le plus petit entier strictement positif tel que :

$$(k\alpha) \bmod z = 0 \tag{2.46}$$

il existe alors un cycle de longueur $2kl$. Plus précisément, il existe dans ce cas z/k cycles de longueur $2kl$.

Dans le cas où il existe un cycle de longueur $2l$ dans le graphe de base, il peut être intéressant de déterminer le coefficient δ_{A_j}, connaissant les $2l-2$ coefficients $\{\delta_{A_i},\ \forall i \neq j\}$, qui maximise le cycle dans la configuration considérée. Soit γ_j la somme partielle définie par :

$$\gamma_j = \left(\sum_{\substack{i=0 \\ i \neq j}}^{2l-1} (-1)^i \, \delta_{A_i} \right) \bmod z \tag{2.47}$$

La longueur du cycle dans la configuration considérée sera maximisée si :

$$\left(k \left(\gamma_j + (-1)^j \, \delta_{A_j} \right) \right) \bmod z \neq 0, \quad \forall k = 1 \ldots z-1 \tag{2.48}$$

Autrement dit, la longueur du cycle est maximisée si les entiers $\left(\gamma_j + (-1)^j \, \delta_{A_j} \right)$ et z sont premiers entre eux. Cette relation est très importante puisqu'elle permet en travaillant sur le graphe de base de maximiser localement un cycle simplement par le choix des coefficients de permutation. En dérivant cette relation il est aussi possible de trouver un ensemble de coefficients répondant à une contrainte donnée sur un cycle de longueur minimale.

□ *Exemple:* On considère une matrice de contrôle de parité $\mathbf{H}$ de taille $2z \times 2z$ et sa matrice de base associée définie par :

$$\mathbf{h} := \begin{bmatrix} \delta & 0 \\ 4 & 0 \end{bmatrix}$$

Le facteur d'expansion z est fixé à 10. Le graphe de la matrice de base a un cycle de longueur 4 ($l = 2$). Les coefficients δ qui maximisent le cycle sont l'ensemble des coefficients tels que $\delta - 4$ et 10 soient premiers entre eux. Ces coefficients sont donc $\delta = 1, 3, 5, 7$. Si au lieu de maximiser la longueur du cycle, l'on cherche à garantir une longueur de cycle au moins supérieure à 8 ($k = 2$), les contraintes sont les suivantes :

$$\begin{aligned} (\delta - 4) \bmod 10 &\neq 0 \\ (2\,(\delta - 4)) \bmod 10 &\neq 0 \end{aligned}$$

Les coefficients interdits sont donc 4 et 9. □

Il faut noter qu'il existe dans la littérature des analyses similaires [51, 83] basés sur des approches différentes. L'ensemble de ces relations permettent de faire un choix sur les coefficients de permutation en fonction d'une contrainte de longueur de cycle donnée. A partir de ces résultats, un algorithme de construction de codes peut être décrit.

2.2.2.4 Définition d'un algorithme de construction de code

Un algorithme de construction de code, basé sur les résultats présentés dans les paragraphes précédents est proposé. Le point de départ de l'algorithme de construction est la définition des distributions d'irrégularités et des paramètres du codes m, z et n. A partir de ces données et de la structure de code choisie, les positions des coefficients positifs de la matrice de base sont déterminées. Ce choix revient donc à définir un graphe de base. Ce graphe peut être construit en utilisant l'algorithme PEG [50], de manière à maximiser la longueur des cycles dans le graphe de base. Plus précisément, la longueur des cycles faisant intervenir les noeuds de données de faible degré sera maximisée.

Une fois les positions des coefficients positives définies, l'algorithme de construction de code doit choisir les valeurs des coefficients de permutation. Une première solution consiste à déterminer de manière pseudo-aléatoire chaque coefficient (suivant une loi par exemple). Une fois cet ensemble déterminé, le code peut ensuite être testé de manière à vérifier ses propriétés de cycles et de distance. Cette première méthode, très simple à mettre en oeuvre, n'est cependant pas très performante. En effet, la majorité des codes ont un graphe avec des cycles de faibles longueurs.

Une deuxième solution consiste à construire le code de manière incrémentale : chaque coefficient est déterminé successivement selon des critères donnés, appliqués localement[(6)]. L'algorithme peut contraindre par exemple le graphe à un cycle minimum $g_{\min}$. Lors d'une première étape, toutes les configurations du graphe de base pouvant engendrer un cycle inférieur à $g_{\min}$ sont détectées. La seconde étape consiste à analyser ces configurations et, à partir des informations sur les coefficients de permutation déjà définis, construire une liste de permutations interdites. Dans un dernier temps, le coefficient est choisi d'une manière pseudo aléatoire parmi la liste des coefficients possibles.

[(6)] La construction incrémentale est la méthode utilisée dans l'algorithme PEG

En résumé, l'algorithme de construction de code peut s'exprimer de la façon suivante :

Initialisation : Determination des positions des coefficients à determiner et cycle minimum admissible : $g_{\min}$
Génération incrémentale de la matrice de base associée à $\mathbf{H}$:
for $j = 0$ **to** $n - 1$ **do**
  for $i = 0$ **to** $m - 1$ **do**
    Liste_Coefficient = 0 to $z - 1$
    if $\delta(i, j) \geq 0$ **then**
      Liste_Interdite = Recherche_Coefficients_Interdit($g_{\min}$)
      Liste_Coefficient = Liste_Coefficient - Liste_Interdite
      if *card*(Liste_Coefficient) > 0 **then**
        $\delta(i, j)$ = Liste_Coefficient(Rand())
      else
        Exit - Pas de solution
      end if
    end if
  end for
end for

Il faut noter que cet algorithme peut prendre en compte d'autres critères pour la construction de la liste interdite, tel que ceux relatifs aux propriétés de distance du code.

□ *Exemple:* Dans le but d'illustrer l'algorithme de construction incrémentale, nous proposons un exemple simple où le facteur d'expansion z est fixé a 8 et où la matrice de contrôle de parité du code est définie par :

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_{\delta_0} & \mathbf{I}_{\delta_3} & \mathbf{I}_{\delta_6} & \mathbf{I_0} & & \mathbf{I'}_1 \\ \mathbf{I}_{\delta_1} & \mathbf{I}_{\delta_4} & \mathbf{I}_{\delta_7} & \mathbf{I_0} & \mathbf{I_0} & \\ \mathbf{I}_{\delta_2} & \mathbf{I}_{\delta_5} & \mathbf{I}_{\delta_8} & & \mathbf{I_0} & \mathbf{I_0} \end{bmatrix}$$

Nous proposons de construire un code dont le plus petit cycle est de longueur 6. La construction incrémentale se déroule de la manière suivante :

Étape	Matrice à analyser	Coefficients interdits	Choix du coefficient
1	$\begin{bmatrix} \mathbf{I}_{\delta_0} & \bullet & \bullet & \mathbf{I_0} & & \mathbf{I'}_1 \\ \bullet & \bullet & \bullet & \mathbf{I_0} & \mathbf{I_0} & \\ \bullet & \bullet & \bullet & & \mathbf{I_0} & \mathbf{I_0} \end{bmatrix}$	$\emptyset$	$\delta_0 = 0$
2	$\begin{bmatrix} \mathbf{I}_0 & \bullet & \bullet & \mathbf{I_0} & & \mathbf{I'}_1 \\ \mathbf{I}_{\delta_1} & \bullet & \bullet & \mathbf{I_0} & \mathbf{I_0} & \\ \bullet & \bullet & \bullet & & \mathbf{I_0} & \mathbf{I_0} \end{bmatrix}$	$[0]$	$\delta_1 = 6$
3	$\begin{bmatrix} \mathbf{I}_0 & \bullet & \bullet & \mathbf{I_0} & & \mathbf{I'}_1 \\ \mathbf{I}_6 & \bullet & \bullet & \mathbf{I_0} & \mathbf{I_0} & \\ \mathbf{I}_{\delta_2} & \bullet & \bullet & & \mathbf{I_0} & \mathbf{I_0} \end{bmatrix}$	$[7, 6]$	$\delta_2 = 3$
4	$\begin{bmatrix} \mathbf{I}_0 & \mathbf{I}_{\delta_3} & \bullet & \mathbf{I_0} & & \mathbf{I'}_1 \\ \mathbf{I}_6 & \bullet & \bullet & \mathbf{I_0} & \mathbf{I_0} & \\ \mathbf{I}_3 & \bullet & \bullet & & \mathbf{I_0} & \mathbf{I_0} \end{bmatrix}$	$\emptyset$	$\delta_3 = 0$
5	$\begin{bmatrix} \mathbf{I}_0 & \mathbf{I}_0 & \bullet & \mathbf{I_0} & & \mathbf{I'}_1 \\ \mathbf{I}_6 & \mathbf{I}_{\delta_4} & \bullet & \mathbf{I_0} & \mathbf{I_0} & \\ \mathbf{I}_3 & \bullet & \bullet & & \mathbf{I_0} & \mathbf{I_0} \end{bmatrix}$	$[0, 6]$	$\delta_4 = 7$
$\vdots$			
9	$\begin{bmatrix} \mathbf{I}_0 & \mathbf{I}_0 & \mathbf{I}_0 & \mathbf{I_0} & & \mathbf{I'}_1 \\ \mathbf{I}_6 & \mathbf{I}_7 & \mathbf{I}_3 & \mathbf{I_0} & \mathbf{I_0} & \\ \mathbf{I}_3 & \mathbf{I}_1 & \mathbf{I}_{\delta_8} & & \mathbf{I_0} & \mathbf{I_0} \end{bmatrix}$	$[3, 0, 1, 5, 7]$	$\delta_8 = 6$

La détection des cycles dans le graphe de base se fait par la construction d'un arbre dont le nombre d'étages est une fonction de la longueur du cycle à détecter. La figure 2.11 illustre la construction de l'arbre à l'étape 3 pour la recherche de cycles de longueur 4. Dans le cas présenté deux cycles sont détectés. Connaissant les coefficients de permutation sur les branches de l'arbre, il est possible de déterminer les coefficients interdits. □

L'analyse des cycles présents dans la structure étudiée a permis de dériver des règles de base pour la construction de code. A partir de la représentation géométrique proposée, il est possible de trouver un ensemble de coefficients répondant à une contrainte donnée sur la longueur d'un cycle. Les relations décrites dans ce paragraphe permettent d'écrire un algorithme de construction très simple, fonctionnant directement sur le graphe de base associé à la matrice de contrôle de parité. Cet algorithme permet la construction de codes à partir de matrices identité permutées sous la contrainte d'une longueur de cycle minimale.

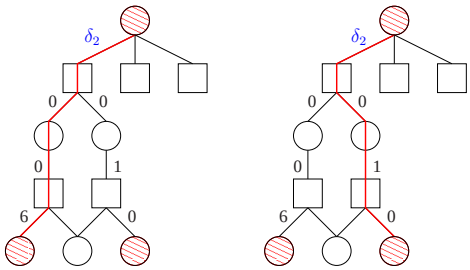


FIG. 2.11 – Illustration de la construction de l'arbre du graphe de base à l'étape 3 pour la recherche de cycles de longueur 4. Dans le cas présenté deux cycles sont détectés.

2.2.3 Construction des codes : analyse expérimentale

Dans les deux paragraphes précédents, nous nous sommes intéressés aux propriétés de distance et de cycle de la structure de code définie. Dans cette section, nous détaillerons quelques remarques issues d'observations expérimentales des codes construits à partir des règles illustrées précédemment. Dans un premier temps, le concept de pseudo-mot de code sera présenté. Dans un second temps, des remarques sur le choix des distributions d'irrégularités des codes seront développées.

2.2.3.1 Pseudo-mot de code

Un des inconvénients des codes LDPC irréguliers est l'apparition assez rapide du phénomène d'*error floor*. Si on considère un Turbo-code, cet effet est principalement dû aux mots de code de poids faibles. Dans le cas des codes LDPC décodés en utilisant un algorithme de type BP, il a été démontré dans [84, 74] que le phénomène d'*error floor* était non seulement dû aux mots de code de poids faible, mais aussi à des ensembles pièges dits *trapping sets*, qui font diverger le décodeur. L'effet de ces configurations peut être mis en évidence en déterminant le nombre d'erreurs dans un mot de code en fonction du nombre d'itérations (figure 2.12). Dans le cas d'un échec du décodage, le nombre d'erreurs tend vers une asymptote. Dans le cas d'un ensemble piège, le décodeur devient instable et le nombre d'erreurs oscille.

Les *trapping sets* sont aussi connus sous la dénomination de *pseudo-mots de code* [85]. Un pseudo-mot de code (w^p, v) associé à une matrice de contrôle de parité $\mathbf{H}$, est un vecteur $\underline{x}$ de dimension N et de poids w^p dont le syndrome défini par $\underline{s}^t = \mathbf{H}\underline{x}^t$ est de poids v. Les pseudo-mots de code où w^p et v ont des valeurs relativement faibles peuvent être des ensembles pièges pour le décodeur BP. Un exemple de pseudo-mot de code $(w^p = 7, v = 1)$ est illustré figure 2.13.

Remarque : Il faut noter que l'importance des configurations pièges sur les performances du code est liée à l'algorithme de décodage. En effet suivant l'ordonnancement choisi, les configurations pièges n'auront pas la même importance. Par exemple une configuration piège de l'algorithme BP avec un ordonnancement par inondation peut

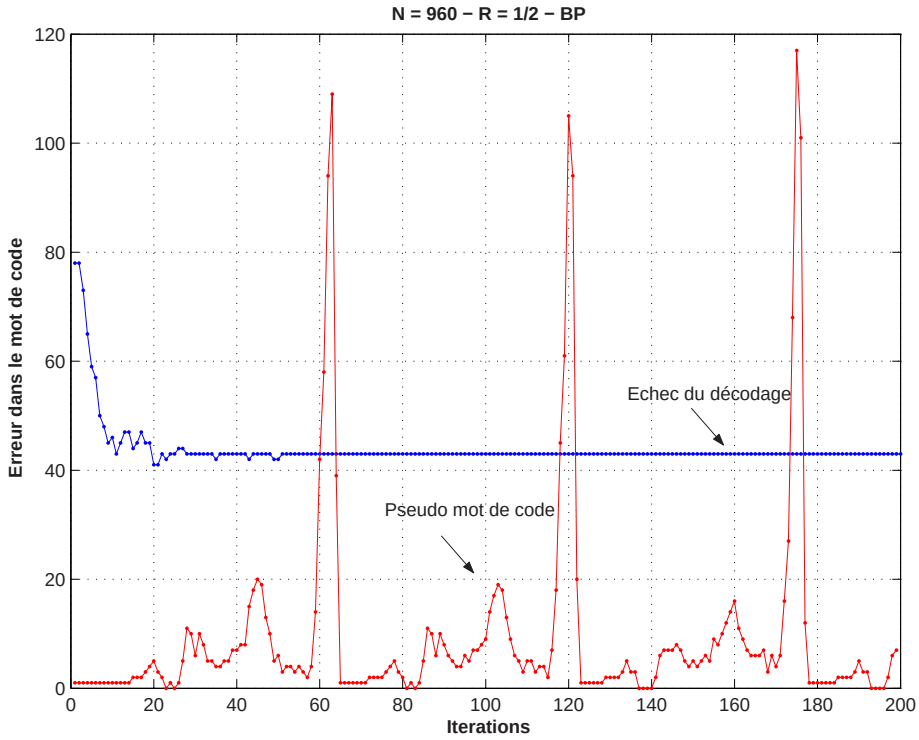


FIG. 2.12 – Exemple de comportement d'un décodeur BP vis à vis de deux configurations différentes. Une des configurations du mot d'entrée a excité un ensemble piège, résultant en une instabilité du décodeur.

ne pas l'être pour un ordonnancement de type *shuffle*. Cet effet peut expliquer les résultats de [39] où les performances sont améliorées par simple changement d'ordonnancement de l'algorithme de décodage.

Les pseudo-mots de code de poids faibles sont principalement générés par des combinaisons de noeuds où apparaissent des noeuds de données de degrés faibles. Plus particulièrement, ces pseudo-mots de code sont générés par des combinaisons faisant intervenir des noeuds de données de degrés 2 et 3 [86, 87]. L'existence de pseudo-mots de code de poids faibles est intimement liée à la présence de cycles courts faisant intervenir des noeuds de données de degré faible. Il faut donc non seulement garantir une longueur de cycle minimum dans le graphe du code mais aussi garantir que les cycles faisant intervenir des noeuds de données de degré faible soient de longueur suffisante. Cette observation est à l'origine des algorithmes de construction de code basés sur l'algorithme PEG [87], ou ACE (Approximate Cycle Extrinsic message) [76] qui prennent en compte ce critère.

Dans le cas de la structure étudiée, nous nous intéressons aux pseudo-mots de code faisant intervenir une combinaison de noeuds de degré 2 et q. Ces pseudo-mots de code seront notés (n', w^p, v), où n' correspond au nombre de noeuds de degré différent de

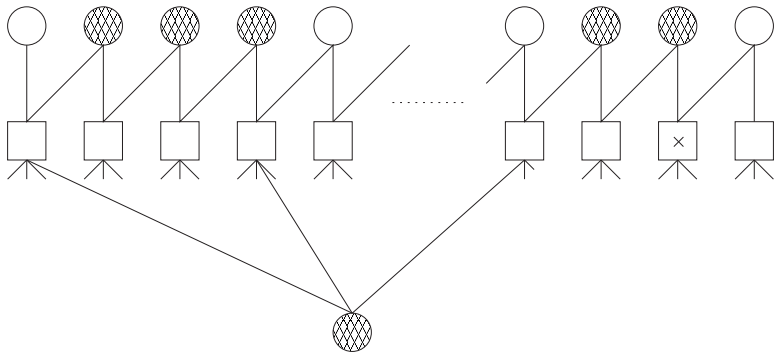


FIG. 2.13 – Exemple d'un pseudo-mot de code $(7, 1)$. Les noeuds de données grisés correspondent à une valeur binaire de 1. Les noeuds de données blancs correspondent à une valeur binaire de 0. Dans ce cas, une seule équation de parité n'est pas respectée.

2 intervenant dans la configuration. A titre d'exemple, le pseudo-mot de code illustré figure 2.13 est un pseudo-mot de code $(n' = 1, 7, 1)$. Comme mentionné dans [87, 86], nos observations ont montré que les pseudo-mots de code de poids et syndrome faibles sont principalement générés par des configurations où $q = 3$ et $n' = 1, 2, 3, 4$. On peut expliquer ce phénomène par le fait que les mots de poids faibles sont statistiquement générés par la projection de mot d'information de poids faibles sur des colonne de $\mathbf{H_s}$ où le nombre de 1 est faible. Les pseudo-mots de code ayant une structure partiellement identique à ceux d'un mot de code, ils sont aussi statistiquement généré par des mots d'information de poids faibles.

Il existe une manière très simple de détecter ces configurations. On considère l'ensemble des mots d'informations de poids n_q multiplié par la matrice $\mathbf{H_s}$ et résultant en un vecteur $\underline{v}$. Le codage de ce vecteur par l'accumulateur permet de déduire les poids des mots de code issus de cette configuration. Le codage de l'ensemble des vecteurs $\underline{v'}$ où v éléments non nuls ont été remplacés par un 0, permet de déduire l'ensemble des pseudo-mots de code (n', w^p, v). L'analyse des poids de ces pseudo-mots de code est similaire à l'analyse des distances. En utilisant ces mêmes principes, le poids du pseudo-mot de code sera une fonction de la différence des positions des éléments non nuls dans la séquence d'entrée de l'accumulateur. En conséquence, une attention particulière doit être prise pour que quand n' est petit, les positions des éléments non nuls soient suffisamment espacées. Autrement dit, si une distance suffisamment grande est imposée entre les q coefficients de permutation d'une colonne de la matrice de base associé à $\mathbf{H_s}$, on peut alors garantir une certain poids w^p du pseudo-mot de code. En particulier dans le cas où $q = 3$ si on garantit que la métrique Γ définie par :

$$\Gamma(\delta_u, \delta_v) = \min\left\{(\delta_u - \delta_v) \bmod z; (\delta_v - \delta_u) \bmod z\right\}, \forall u \neq v \tag{2.49}$$

est supérieur à γ, le poids du pseudo-mot de code $(n' = 1, w^p, 1)$ est majoré par :

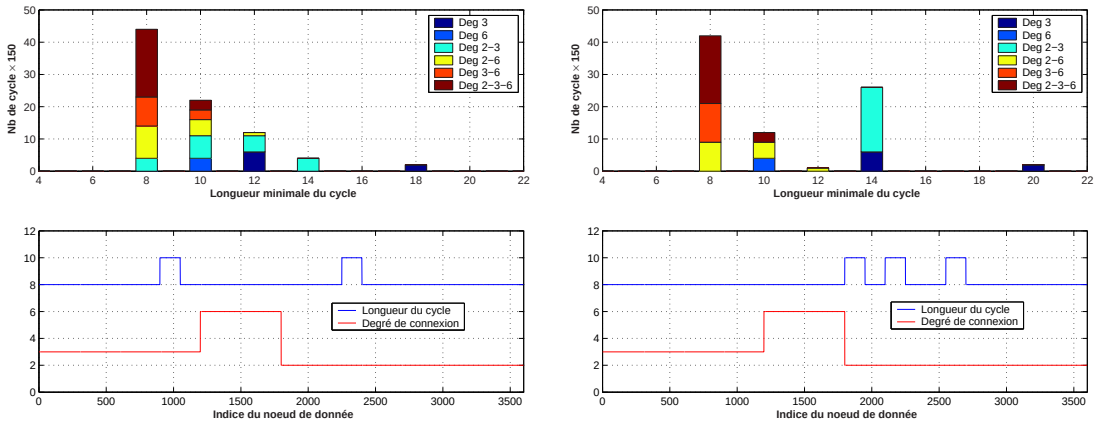
$$w^p \geq (\gamma - 1)m \tag{2.50}$$

Cette condition est redondante avec celle déjà énoncée dans le cadre de l'analyse des distances. Il faut noter que cette condition revient aussi à garantir un cycle minimum faisant intervenir un noeud de données de degré 3 et des noeuds de données de degré 2.

Ces observations nous permettent de considérer des étapes supplémentaires dans le choix des codes. La première conséquence de la mise en évidence des effets des pseudo-mots de code est la définition du paramètre $g_{\min}$, correspondant au plus petit cycle admissible dans le graphe, de l'algorithme de construction de code. En effet plutôt que de garantir une certaine longueur de cycle dans le graphe, il parait plus pertinent que ce paramètre varie en fonction du degré de connexion du noeud. En pratique, il est donc préférable que la construction incrementale du code soit menée suivant les degrés croissants. En effet, les premières configurations considérées seront celles faisant intervenir des noeuds de degrés faibles. Il semble aussi préférable que l'effort d'optimisation relatif au paramètre $g_{\min}$ soit plus important pour ces configurations. En utilisant cette méthode, on garantira non seulement une longueur de cycle minimum, mais aussi une certaine longueur de cycle faisant intervenir des noeuds de degrés faibles. Ces observations sont équivalentes à celles qui ont motivés la définition d'algorithmes de construction de code basés sur l'algorithme PEG tel que ceux présentés dans [87] ou [76].

□ *Exemple:* On considère un code de rendement $R = 1/2$ et de taille $N = 3600$. Le profil d'irrégularité du code est illustré sur la figure 2.14(a). Un premier code est déterminé par l'algorithme d'optimisation où le paramètre $g_{\min}$ est fixé à 8, quelque soit le degré de connexion des noeuds. L'histogramme de la figure 2.14(a) illustre la longueur et la multiplicité des plus petits cycles dans différents sous graphes. On considère dans un premier temps les sous graphes ne contenant que les noeuds de degrés identiques puis les sous graphes issus de combinaisons entre noeuds de degrés différents. La distribution des cycles du graphe total est noté (Deg 2-3-6). Ce même histogramme est illustré sur la figure 2.14(b) où le code a été obtenu en utilisant un paramètre $g_{\min}$ variant en fonction des degrés des noeuds de données. Au cours de la détermination incrémentale des coefficients de permutations associés à des noeuds de données de degré 3, le paramètre $g_{\min}$ est fixé à 14. Globalement, les distributions des cycles dans le graphe total des codes sont proches. On peut cependant mettre en évidence la différence de distribution des cycles ne faisant intervenir que des noeuds de degrés 3 et 2. Comme le montrent les performances illustrées sur la figure 2.15, le code ayant la meilleure distribution de cycles faisant intervenir des noeuds de degré faible est le code offrant les meilleures performances à haut rapport signal sur bruit. Cette propriété peut être également mise en évidence en calculant les distances $w_i,\ i = 1, 2, 3$ et le poids des pseudo-mots de codes. Ces paramètres sont illustrés dans la table 2.3. Le code ayant les moins bonnes performances a un pseudo-mot de code de poids 6.

□



(a) (b)

FIG. 2.14 – Distribution des cycles dans le graphe du code et dans des sous graphes regroupant les noeuds de même degrés.

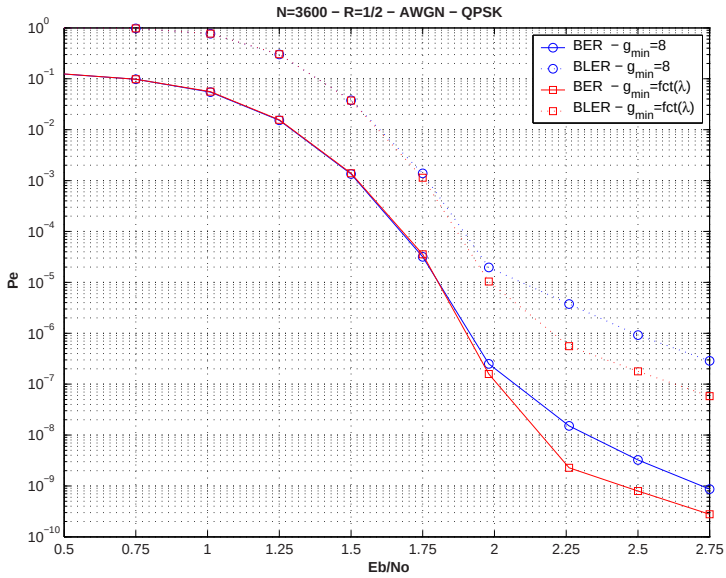


FIG. 2.15 – Simulation des deux codes issus de l'algorithme d'optimisation avec des paramètres de constructions différents

	$g_{\min}$	w_1	w_2	w_3	$\min\limits_{n'=1;s=1} w^p$	$\min\limits_{n'=2;s=1} w^p$
Code 1 - (○)	8	156	38	30	152	6
Code 2 - (□)	variable	129	38	50	124	14

TAB. 2.3 – Paramètres des codes simulés. On rappelle que w_i correspond au plus petit poids des mots de codes issus du codage de l'ensemble des mots d'information de poids i.

Remarque sur le choix des distributions d'irrégularité

L'étude de certaines configurations permet de fixer des règles sur les coefficients de permutation de façon à éviter la génération de mots de code ou de pseudo-mots de code de poids faibles. Nous avons mentionné que ces configurations sont statistiquement générées par des mots d'entrée de poids faibles. Pour mettre en évidence ce phénomène, des résultats de simulation de quatre codes de rendement $R = 1/2$ et de taille $N = 3600$ avec différents profils d'irrégularités sont illustrés figure 2.16. L'ensemble des paramètres définissant les codes sont décrits dans la table 2.4. Le seuil de convergence est calculé par évolution de densité sous approximation gaussienne [47].

Les résultats de simulation montrent que plus la proportion de noeuds de données de degré 3 est élevée, plus le phénomène de plancher est important (Code 2 et Code 3). Ce plancher n'est pas visible quand $q = 4$ (Code 1). Il faut cependant noter que ce code est celui ayant le seuil de convergence le plus élevé. A titre d'information, le code 3 est le code irrégulier de type $(q = 3, x = 2)$ qui minimise le seuil pour la distribution des noeuds de contrôle proposée.

Ce phénomène, conséquence directe des résultats de l'analyse, montre qu'il faut chercher à minimiser le seuil de convergence, tout en minimisant la proportion de noeuds de données de degré q. Ce critère a pour but de diminuer les configurations possibles de mots de code et de pseudo-mots de code de poids faibles, qui sont principalement engendrées par des configurations avec des noeuds de données de faibles degrés de connexion. Par exemple, le code 4 est un code irrégulier de type $(q = 3, x = 3)$ optimisé sous la contrainte $\tilde{\lambda}_3 = 1/4$, $\tilde{\rho}(x) = x^5$, $m = 12$ et dont le degré de connexion maximum des noeuds de données est de 6.

	Code 1	Code 2	Code 3	Code 4
m	12	12	12	12
z	150	150	150	150
q	4	3	3	3
$\tilde{\lambda}_3$	-	1/4	1/3	1/4
$\tilde{\lambda}_4$	1/2	-	-	1/8
$\tilde{\lambda}_5$	-	1/4	-	-
$\tilde{\lambda}_6$	-	-	1/6	1/8
$\tilde{\lambda}_2$	1799/3600			
$\tilde{\lambda}_1$	1/3600			
$\tilde{\rho}(x)$	$1799/1800x^5 + 1/1800x^4$			
Seuil (dB)	0.81	0.69	0.63	0.66

TAB. 2.4 – Paramètres des codes simulés figure 2.16

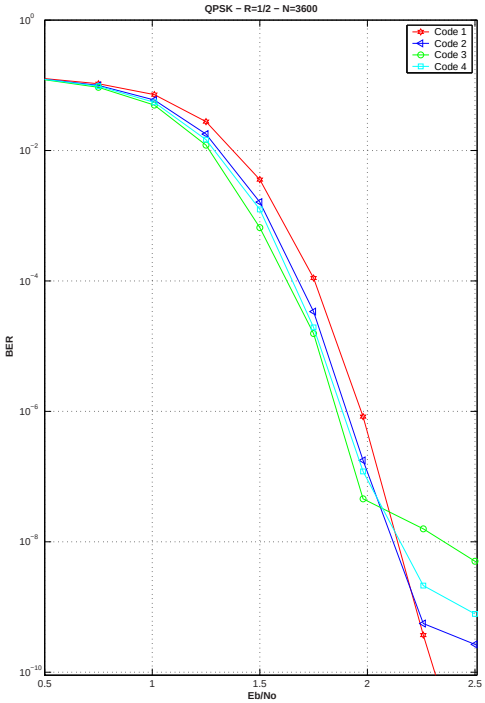


FIG. 2.16 – Comparaison des performance sur canal AWGN de codes LDPC de taille $N = 3600$ et de rendement $R = 1/2$. Les paramètres de chaque code sont décrits dans la table 2.4. Les performances ont été mesurées sur un décodeur matériel.

Comme le montrent les performances, ce code est un compromis entre l'effet de plancher du code 2 et le seuil de convergence du code 2.

2.2.3.2 Conclusion

L'analyse de la structure de code choisie nous a permis de comprendre certains phénomènes qui agissent sur le comportement du code. L'analyse des poids de Hamming de mots de code générés par des mot d'information de poids faibles nous a permis de dériver certaines règles de constructions. Ces règles sont aussi liées à celles dérivées de l'analyse de la distribution des cycles dans le graphe du code. Ces travaux ont permis, par la définition du structure de codage simple, la définition d'un algorithme efficace de choix de code[(7)]. Même si l'algorithme est très simple à mettre en oeuvre, sa paramétrisation est quand à elle plus complexe. En fonction de l'application et du profil d'irrégularité et de la taille du code certains paramètres doivent être ajustés empiriquement pour obtenir les meilleurs performances (la valeur de la longueur du plus petit cycle en fonction du degré de connexion par exemple).

[(7)] On peut noter qu'une approche quasi similaire a été indépendamment proposée dans [88]

2.3 Étude d'un algorithme de décodage adapté à la structure étudiée

Dans le chapitre précédent, il a été souligné l'importance du choix de l'ordonnancement de l'algorithme de décodage. Le choix de l'ordonnancement doit se faire suivant des considérations de performance et de réalisation matérielle. L'intérêt des ordonnancements de type *shuffle* a été largement démontré dans la littérature. Dans ce chapitre, nous proposons d'étudier deux ordonnancements. Le premier ordonnancement présenté est l'application directe du concept du séquencement de type *shuffle* dans le cas de codes construits à partir de matrices identité permutées. Cet ordonnancement est communément appelé *Layered BP* (ou *group shuffle*). Le second ordonnancement est issu de considérations spécifiques aux codes étudiés. Il sera détaillé dans la seconde partie et dénoté ordonnancement *Turbo BP*.

2.3.1 Algorithme Layered BP

L'algorithme *Layered BP* est l'application directe de l'ordonnancement de type *shuffle* introduit dans [30]. Comme mentionné précédemment, cet ordonnancement est aussi connu sous le nom de *Horizontal shuffle*, où encore *Turbo Like message passing algorithm* [73].

2.3.1.1 Description détaillée de l'algorithme

Ce séquencement consiste à considérer le code LDPC comme une concaténation parallèle de M codes de parités. Le décodage du code se fait à l'image d'un Turbo-code parallèle : les M codes de parités sont décodés les uns après les autres. Contrairement à l'ordonnancement classique où un noeud est mis à jour une fois par itération, un noeud connecté à d_v équations de parité sera mis à jour d_v fois au cours d'une même itération. Le principe de ce séquencement est donc de toujours utiliser la dernière information mise à jour. Cette stratégie de décodage explique l'augmentation de la vitesse de convergence de l'algorithme de décodage en comparaison à l'algorithme BP avec un séquencement par inondation.

Cet algorithme de décodage peut se décrire en trois étapes. Une première étape consiste à calculer les messages m_{vc} se propageant des noeuds de données vers le noeud de contrôle c considéré :

$$\begin{aligned} m_{vc} &= \underbrace{y_v + \sum_{c' \in C_v} m_{c'v}}_{} - m_{cv} \\ m_{vc} &= \underbrace{\qquad A_v \qquad}_{} - m_{cv} \end{aligned} \tag{2.51}$$

On rappelle que A_v est l'information *a posteriori* associée au noeud v. Contrairement aux équations de décodage présentées précédemment, l'information *a posteriori* est

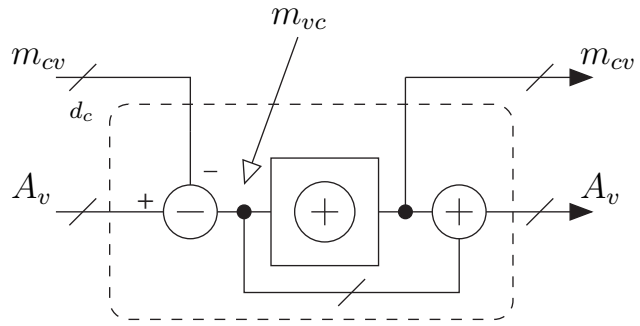


FIG. 2.17 – Schéma du décodeur associé à un décodage d'une équation de parité dans un séquencement *Layered BP*. d_c représente le nombre de bits intervenant dans l'équation de parité.

mise à jour plusieurs fois dans la même itération.

A partir des messages m_{vc} correspondant à la somme de l'observation du canal et des informations *a priori*, l'équation de parité est vérifiée. La résolution de l'équation de parité permet dans un deuxième temps la génération des informations extrinsèques m_{cv} associées à chaque élément intervenant dans l'équation de parité. La dernière étape consiste donc à mettre à jour l'information *a posteriori* associée à chaque noeud v :

$$\begin{aligned} A_v &= \underbrace{y_v + \sum_{c' \in C_v/c} m_{c'v}}_{} + m_{cv} \\ A_v &= \underbrace{\qquad m_{vc} \qquad}_{} + m_{cv} \end{aligned} \tag{2.52}$$

Ces deux équations de décodage nous permettent de considérer le schéma de décodeur illustré sur la figure 2.17. L'écriture de l'algorithme de décodage met en évidence deux points importants. Tout d'abord le décodeur devra travailler avec deux mémoires. La première, de profondeur N, mémorise l'information à postériori (A_v), alors que la seconde stocke l'information extrinsèque associée à chaque noeud intervenant dans une équation de parité. Le nombre d'éléments à mémoriser est donc égal au nombre d'éléments non nuls dans la matrice de contrôle de parité ($N \sum_i \tilde{\lambda}_i i$). Cette réalisation permet d'illustrer l'importance des algorithmes de décodage sous-optimaux qui réduisent le nombre de messages différents générés par un noeud de contrôle.

□ *Exemple:* On considère un code de taille N et de distribution $\rho(x) = x^{d_c-1}$. Quelque soit l'algorithme de décodage utilisé, la mémoire stockant l'information *a posteriori* (A_v) est une mémoire de N mots de Q_A bits. Si on considère un algorithme BP pour la résolution de l'équation de parité, le nombre de messages différents par équation de parité est égal à d_c. La mémoire stockant l'information extrinsèque est donc une mémoire de Md_c mots de Q_m bits. Dans le cas de la résolution de l'équation de parité par un algorithme Min-Sum, les messages générés par le noeud de contrôle sont composés de :

- d_c signes différents.
- 2 amplitudes différentes correspondant au plus petit et au deuxième plus petit message entrant dans le noeud en valeur absolue

Il est donc possible de dimensionner une mémoire contenant les M bits de signe et $2M(Q_m - 1)$ bits correspondant à la mémorisation des deux plus petits messages. De manière à pouvoir reconstruire l'ensemble de l'information extrinsèque associé à une équation, il est nécessaire de connaître le noeud associé à la plus petite valeur (ou à la deuxième plus petite). Cette caractérisation nécessite $M \log_2\lceil d_c \rceil$ bits. En résumé la résolution des équations de parité sans approximation nécessitera une mémoire de Md_cQ_m bits alors que le fait d'utiliser un algorithme Min-Sum permet de réduire la mémoire à $M\,(1 + 2(Q_m - 1) + \log_2\lceil d_c \rceil)$. Enfin, il est important de noter que la réduction de mémoire sera d'autant plus importante que d_c sera grand. □

Le seconde remarque concernant cet ordonnancement est le fait que la réalisation du processeur de décodage est indépendante du degré de connexion des noeuds de données[(8)]. En effet, la notion de degrés de connexion des noeuds de données n'intervient qu'à travers un nombre de mises à jour plus important de certains noeuds et non plus en terme d'architecture d'un processeur de noeuds de données.

2.3.1.2 Contraintes sur la construction du code

L'utilisation d'un tel algorithme de décodage a pour hypothèse la résolution des équations de parité d'une manière séquentielle. Un élément intervenant dans une équation de parité ne doit pas être traité dans un même temps par une autre équation de parité. Pour pouvoir garantir un certain débit de fonctionnement, il est nécessaire que plusieurs équations de parité soient décodées en parallèle. La matrice de contrôle de parité doit donc être contrainte de manière à garantir que la résolution de p équations de parité ne fasse intervenir qu'une seule fois un noeud de donnée. Cette contrainte se traduit par le fait que M/p sous-matrices construites à partir des p lignes décodées simultanément n'ont au maximum qu'un seul élément non nul par colonne. Le facteur de parallélisme maximum doit donc être inférieur à $\lfloor M\ d_v^{\max} \rfloor$ où $d_v^{\max}$ est le degré maximum des noeuds de données.

Un sous-ensemble de codes respectant cette contrainte est l'ensemble des codes dont les matrices de contrôle de parité sont construites à partir de matrices identité permutées de tailles $p \times p$. Dans le cas de la structure étudiée, les matrices identité permutées sont de taille $z \times z$ permettant ainsi un degré de parallélisme égal à z. Cette construction est donc particulièrement adaptée à un algorithme de type *Layered BP* (dont la traduction littérale est BP par couche). Un schéma de principe du déroulement du décodage est illustré sur la figure 2.18.

[(8)] la seule exception porte sur la quantification des messages qui peut être une fonction du degré de connexion maximum d'un noeud de donnée.

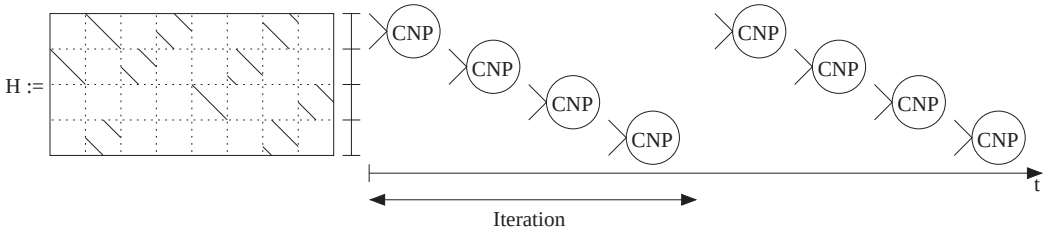


FIG. 2.18 – Schéma du décodage par l'algorithme *Layered BP* d'un code dont la matrice de contrôle de parité **H** est construite à partir de matrice identité permutées. Le concept de décodage par couche (Layer) est ici illustré.

2.3.1.3 Performance de l'algorithme

Il est communément admis [30] qu'il existe un gain d'un facteur 2 entre le nombre d'itérations nécessaires pour un algorithme *Layered BP* et un algorithme BP avec un ordonnancement par inondation. On peut aussi noter que le nombre d'opérations nécessaire au décodage d'une itération de *Layered BP* est plus important que dans le cas du BP. A titre d'exemple, des simulations illustrant le taux d'erreur binaire en fonction du nombre d'itérations pour différents rapports signal à bruit sont illustrées sur la figure 2.19. Le code est de taille $N = 960$ et de rendement $R = 1/2$. Les paramètres z et m sont respectivement égaux à 40 et 12. Ces simulations illustrent le fait que l'algorithme *Layered BP* converge plus vite que son homologue BP. A 2.25dB, un BER de 10^{-4} est atteint avec 10 itérations en utilisant l'algorithme *Layered BP* contre 20 pour un BP avec un ordonnancement par inondation. On peut aussi noter une légère différence de performances entre les deux algorithmes quand le nombre d'itérations augmente. Ces différences peuvent s'expliquer par la meilleure robustesse de l'algorithme *Layered BP* face aux configurations pièges. Un pseudo-mot de code $(w_p, 1)$ sera une configuration piège pour l'algorithme BP, mais pourra être décodée par l'algorithme *Layered BP* si par exemple la première équation décodée est celle qui n'est pas respectée.

L'algorithme et le séquencement *Layered BP* sont particulièrement bien adaptés à la structure étudiée et plus généralement aux codes dont les matrices de contrôle de parité sont construites à partir de matrices identité permutées. Cette combinaison explique l'intérêt que suscite cette famille de codes dans les groupes de normalisation [22, 21], et dans le cadre de la réalisation matérielle de décodeurs [67, 68].

L'utilisation de manière efficace de cet algorithme de décodage n'est rendue possible qu'avec un code construit suivant les critères illustrés précédemment. Cette méthode de décodage utilise donc des propriétés liées à la structure du code. La structure étudiée dans ce manuscrit est non seulement définie à partir de matrices identité permutées mais comporte aussi une structure de type bi-diagonale. Cette propriété peut également être utilisée pour la définition d'un ordonnancement spécifique.

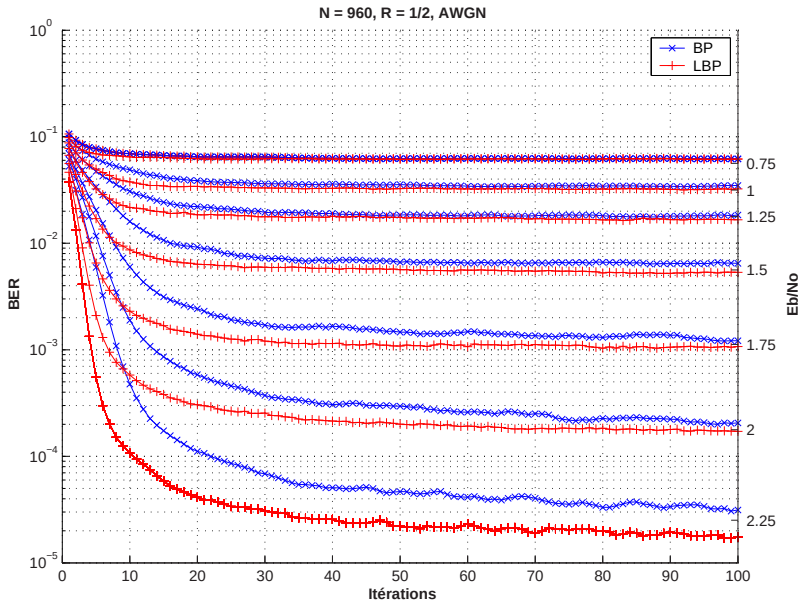


FIG. 2.19 – Simulation d'un code de taille $N = 960$ et de rendement $R = 1/2$ lorsque celui ci est décodé par un algorithme BP avec un ordonnancement par inondation (BP), et par un algorithme *Layered BP* (LBP). La vitesse de convergence de l'algorithme *Layered BP* (LBP) est supérieur à celle de l'algorithme BP.

2.3.2 Turbo BP

L'algorithme de propagation de croyance est exact dans le cas d'un code dont le graphe ne contient pas de cycles. Les auteurs de [89] et [55] ont suggéré qu'un bon ordonnancement peut être défini en isolant des parties suffisamment grandes du graphe qui forment des chaînes et, en les décodant en utilisant un algorithme de propagation de croyance sur un treillis. Avant d'illustrer l'application de ce concept sur la structure étudiée, nous proposons d'étudier l'équivalence qu'il existe entre le décodage d'une chaîne et celui d'un treillis.

2.3.2.1 Équivalence de décodage sur une chaîne et un treillis

Un ensemble de k noeuds de données formant une chaîne suit la relation de parité suivante :

$$\left(\sum_{i=0}^{k-1} b_i\right) \bmod 2 = 0 \tag{2.53}$$

où b_i correspond à la valeur binaire associée au noeud i. Un exemple de chaîne est illustré figure 2.20. La relation définissant une chaîne est en réalité une simple relation d'accumulation. Toute chaîne peut donc être modélisée comme un code convolutif récursif dont la fonction de transfert est :

$$\frac{1}{1+D} \tag{2.54}$$

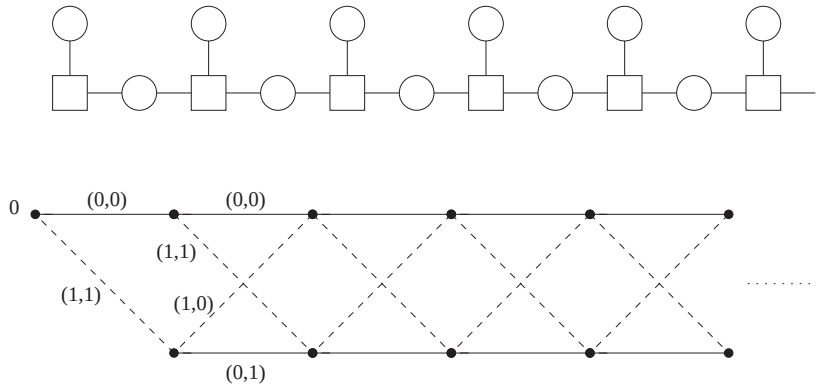


FIG. 2.20 – Illustration de l'équivalence entre une chaîne et un treillis.

Le décodage d'une telle structure peut donc se réaliser sur un treillis à deux états où directement sur le graphe de la chaîne. Il a été montré dans [90] qu'il est plus intéressant de décoder ce type de configuration en utilisant la représentation par un graphe. Le nombre d'opérations est réduit par rapport à un algorithme de type BCJR[9] [91]. Le décodage sur le graphe peut être modélisé de la même manière que le décodage sur un treillis. Une première étape consiste à calculer les messages se propageant sur les branches dans le sens Aller (Forward) F_i (cf Figure 2.21 (a)) de la manière suivante :

$$F_{i+1} = y_{p_{i+1}} + g(v_i, F_i) \tag{2.55}$$

La seconde étape permet de calculer les messages se propageant sur les branches dans le sens Retour (Backward) B_i (cf Figure 2.21 (b)) de la manière suivante :

$$B_i = y_{p_{i+1}} + g(v_{i+1}, B_{i+1}) \tag{2.56}$$

La dernière étape consiste à extraire les informations extrinsèques associées à chaque noeud :

$$E_{v_i} = g(F_i, B_i) \tag{2.57}$$

$$E_{p_i} = F_i + B_{i-1} - 2y_{p_i} \tag{2.58}$$

Dans la suite de ce manuscrit, nous noterons cet algorithme/ordonnancement *Algorithme Aller Retour* où *Forward Backward Algorithm* (FBA).

On peut noter que considérer le décodage le long d'un cycle revient à considérer un treillis à deux états circulaires. Comme nous l'avions illustré dans le premier chapitre (figure 1.20), le décodage d'une équation peut aussi être réalisé avec un algorithme FBA, en considérant une chaîne avec des noeuds poinçonnés. L'équation de parité est

[9] En réalité les deux algorithmes sont équivalents. Dans le cas d'un algorithme de type max log-MAP et d'un treillis à deux états, il existe une équivalence entre les opérations de différence de deux opérateurs max et un opérateur min.

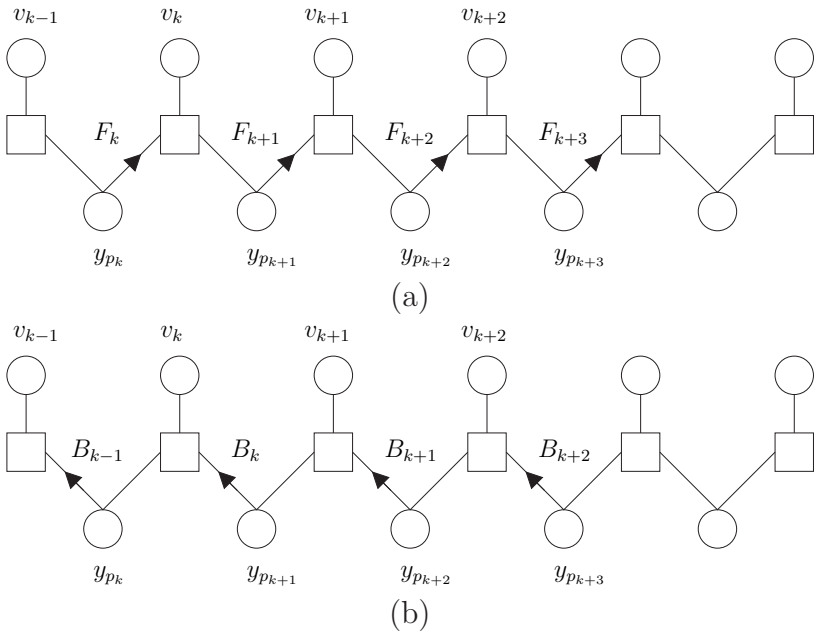


FIG. 2.21 – Illustration du calcul Aller (Forward) des messages de branches (a) et le calcul Retour (Backward).

alors interprétée comme un code accumulateur poinçonné.

A partir de cette illustration du décodage d'une chaîne, nous proposons d'illustrer un ordonnancement integrant ce concept. Celui-ci sera noté *Turbo BP*.

2.3.2.2 Séquencement Turbo BP

De nombreux auteurs ont souligné la possibilité d'un décodage "Turbo like" des codes de type Reapeat Accumulate [55]. Le principe consiste à considérer le décodeur comme le dual de l'encodeur. En effet le code Repeat Accumulate peut se voir comme la concaténation série d'un code LDPC et d'un accumulateur. La matrice de contrôle de parité du code peut alors être ré-écrite de la manière suivante :

$$\mathbf{H} = \begin{bmatrix} \mathbf{H_s} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{H_p} \end{bmatrix} \tag{2.59}$$

où la matrice définie par $[\mathbf{H_s}\ \mathbf{I}]$ peut être interprétée comme un code LDPC (et plus particulièrement on parle de code LDGM, Low Density Generator Matrix [92]). La matrice construite à partir de $[\mathbf{I}\ \mathbf{H_p}]$, qui isole une chaîne, représente un accumulateur. Cette représentation est donc équivalente à une concaténation série et ainsi, un décodeur fonctionnant suivant le principe turbo peut être dérivé. Ce décodeur repose sur un échange d'information entre un décodeur interne représentant le code LDPC $[\mathbf{H_s}\ \mathbf{I}]$ et un code externe, l'accumulateur.

Paradoxalement, le décodage de codes LDPC utilisant ces concepts n'a été, à notre connaissance, que très peu exploré dans la littérature. Une première explication peut être la difficulté d'isoler une chaîne dans un code. Même si d'un point de vue des performances il a été montré l'intérêt de la connexion des noeuds de degrés 2 suivant une chaîne, cette propriété est difficilement exploitable au niveau du décodeur si une forme bi-diagonale de matrice n'apparaît pas explicitement (elle peut être mise en évidence après permutation de lignes et de colonnes de la matrice de contrôle de parité). Une seconde raison semble être le peu d'intérêt pratique qu'il existe à décoder d'abord le code interne puis le code externe au vue de la complexité engendrée en terme de mémoire et de la convergence relativement lente du processus itératif.

Cependant, ce type d'ordonnancement peut devenir très intéressant, d'un point de vue de réalisation matérielle et de performances, si certains principes sont mis en oeuvre. En particulier l'introduction d'un algorithme de type Layered peut se révéler particulièrement pertinent. Le principe au coeur de cet ordonnancement est que la convergence du processus itératif peut être améliorée si les noeuds de données sont mis à jour autant de fois que leurs degrés de connexion. Pour pouvoir garantir cette propriété, il faut être capable de trouver un lien entre l'ordonnancement de décodage du code interne et celui du code externe. Dans la suite du paragraphe nous proposons donc un nouvel ordonnancement pour les codes de type Repeat Accumulate. Ce nouveau séquencement est issu d'une réflexion basée à la fois sur les travaux relatifs aux algorithmes de type *Layered BP* et aux propriétés offertes par le décodage sur un treillis.

Pour permettre une mise à jour des noeuds la plus rapide possible, il est nécessaire de considérer un certain nombre de sections de treillis du code externe (où un certain nombre de noeuds dans la chaîne). Ce type de décodage est possible au niveau du code externe si l'on considère un décodage par fenêtre [93]. Il faut donc être capable de décoder les équations de parité du code interne en relation avec la fenêtre de décodage. En utilisant la relation duale de l'équation 2.14 qui caractérise la transformation entre le vecteur de projection et la séquence d'entrée de l'accumulateur, les équations de parité associées à une fenêtre de décodage k de taille L du code externe correspondent aux lignes :

$$(i \bmod m)\, z + \left\lfloor \frac{i}{m} \right\rfloor, \quad \forall i = kL \ldots (k+1)L - 1 \tag{2.60}$$

de la matrice $[\mathbf{H_s}\ \mathbf{I}]$ définissant le code interne. En particulier il peut être intéressant de fixer la taille de la fenêtre L comme un multiple de m, $L = lm$. Les équations de parité à résoudre correspondent alors à des groupes de l lignes successives des matrices identité permutées. Un exemple est illustré figure 2.22 où $L = m$.

Pour permettre une convergence rapide de l'algorithme de décodage, il est nécessaire que les équations de parité intervenant dans une fenêtre de taille lm ne fassent intervenir un noeud de donnée qu'une seule fois. Cette règle appliquée à la structure étudiée

$$\mathbf{H} = \begin{bmatrix} \mathbf{H_s} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{H_p} \end{bmatrix}$$

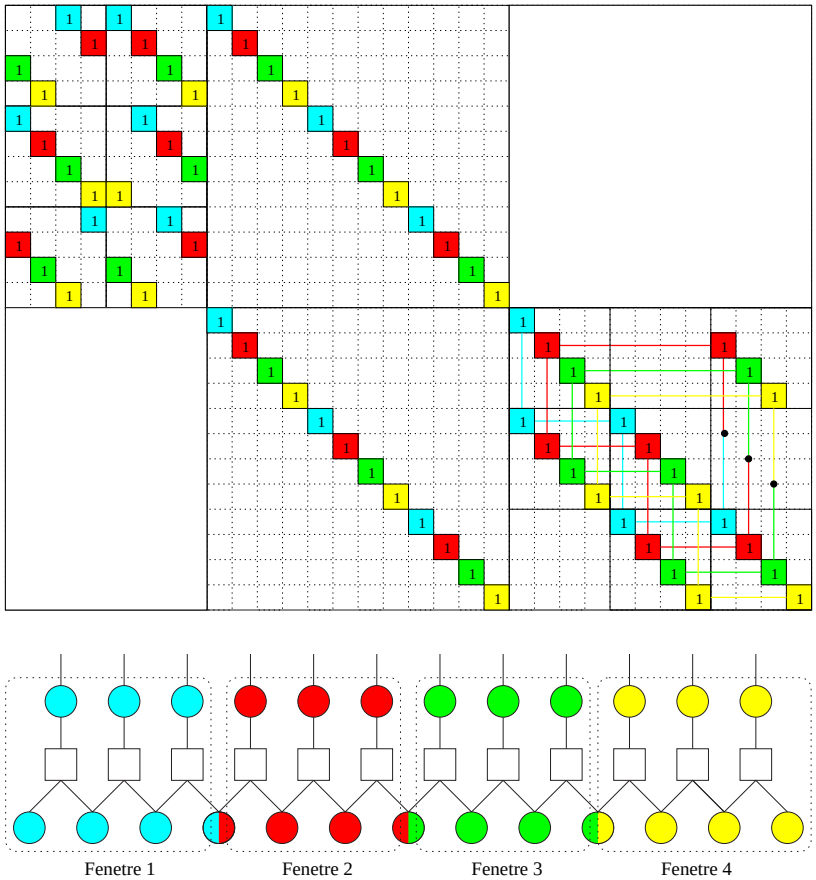


FIG. 2.22 – Illustration du décodage par fenêtres du code. La matrice de contrôle de parité est tout d'abord représentée. Les éléments intervenant dans une même fenêtre ont des couleurs identiques. Le graphe de l'accumulateur et sa décomposition en fenêtres sont également illustrés.

impose que les coefficients de permutation dans une colonne de $\mathbf{S}$ soient distants de l.

□ *Exemple:* Soit la matrice de contrôle de parité suivante ($m = 2$,z) :

$$\mathbf{H} = \left[\begin{array}{cc|cc} \mathbf{I}_{\delta_0} & \mathbf{I}_{\delta_2} & \mathbf{I}_0 & \mathbf{I'}_1 \\ \mathbf{I}_{\delta_1} & \mathbf{I}_{\delta_3} & \mathbf{I}_0 & \mathbf{I}_0 \end{array}\right]$$

la matrice de base associé à $\mathbf{H_s}$ est la matrice $\mathbf{S}$ définie par :

$$\mathbf{S} = \begin{bmatrix} \delta_0 & \delta_2 \\ \delta_1 & \delta_3 \end{bmatrix}$$

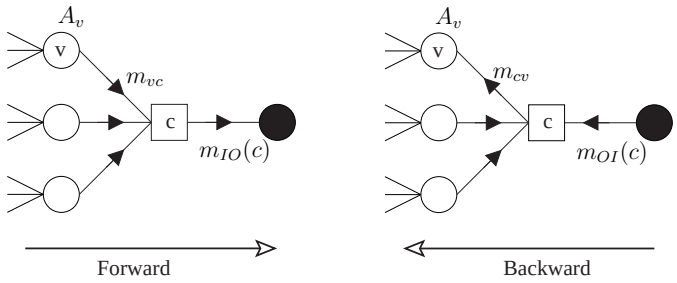


FIG. 2.23 – Illustration de la phase Forward et Backward du calcul des messages du codes LDPC défini par la matrice $[\mathbf{H_s}\ \mathbf{I}]$.

Si l'on souhaite une fenêtre de taille $2m$, les conditions sur les coefficients de permutation sont :

$$\begin{aligned} \delta_1 &\neq (\delta_0 - 2)\text{mod } z; (\delta_0 - 1)\text{mod } z; \delta_0; (\delta_0 + 1)\text{mod } z; (\delta_0 + 2)\text{mod } z \\ \delta_3 &\neq (\delta_2 - 2)\text{mod } z; (\delta_2 - 1)\text{mod } z; \delta_2; (\delta_2 + 1)\text{mod } z; (\delta_2 + 2)\text{mod } z \end{aligned} \qquad \square$$

Nous verrons par la suite qu'un ensemble de règles peut être dérivé suivant l'architecture choisie, notamment dans le cas où du parallélisme est mis en oeuvre au niveau des processeurs de décodage.

Avant d'illustrer les performances d'un tel ordonnancement, nous proposons de décrire l'algorithme de décodage. Cet algorithme peut se décrire en trois phases. Une première phase, dite phase Aller (Forward) consiste à calculer les messages se propageant du code interne vers le code externe (m_{IO}). Ces L messages sont déterminés de la manière suivante (cf figure 2.23 (Forward)) :

$$m_{jc}^{i} = A_j^{u_0} - m_{cj}^{i-1}, \quad \forall j \in V_c' \tag{2.61}$$

$$m_{IO}^{i}(c) = g\left(\bigcup_{j \in V_c'} m_{jc}^{i}\right) \tag{2.62}$$

où V_c' est l'ensemble des noeuds de données correspondant à un bit systématique connecté au noeud de contrôle c. La variable u_0 illustre l'évolution de la mise à jour de l'information *a posteriori*. L'information extrinsèque m_{cv} étant mise à jour une seule fois par itération, son évolution est caractérisée par l'indice i différent de celui utilisé pour l'information *a posteriori* A_v. A la première itération, les messages m_{cv} sont considérés comme nuls.

Dans un deuxième temps, la fenêtre correspondante au niveau de l'accumulateur est décodée. Le décodage du treillis (ou du graphe), réalisé par fenêtre, nécessite une initialisation des extrémités. Plusieurs méthodes sont possibles, dont la méthode dite du *pointeur* [94, 93]. Cette méthode illustrée sur la figure 2.24 consiste à initialiser

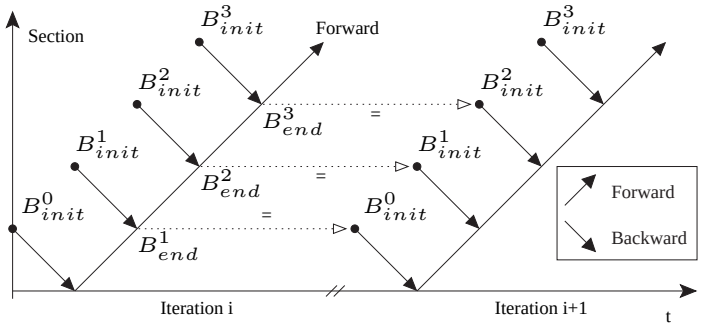


FIG. 2.24 – Illustration de l'initialisation des fenêtres par la méthode du pointeur.

le début d'une fenêtre à l'itération i par les messages de fin de la fenêtre précédente calculés à l'itération $i-1$. A la première itération les initialisations prennent la valeur neutre (en l'occurrence 0 quand on travaille en log-rapport de vraisemblance).

La dernière étape de l'algorithme de décodage (Backward) consiste à mettre à jour l'information *a posteriori* A_v associée à chaque noeud de données. Cette opération peut se décrire de la manière suivante :

$$m^i_{cv} = g\left(m^i_{IO}(c), \bigcup_{j\in V'_c/v} \left(A^{u_0}_j - m^{i-1}_{cj}\right)\right) \tag{2.63}$$

$$A^{u_0+1}_v = A^{u_0}_v - m^{i-1}_{cv} + m^i_{cv} \tag{2.64}$$

Dans le cas où les messages $m^i_{jc},\ j \in V'_c$ ont été mémorisés le temps du traitement d'une fenêtre du code externe, la mise à jour de l'information *a posteriori* peut se réaliser de la manière suivante :

$$m^i_{cv} = g\left(m^i_{IO}(c), \bigcup_{j\in V'_c/v} m^i_{jc}\right) \tag{2.65}$$

$$A^{u_0+1}_v = m^i_{vc} + m^i_{cv} \tag{2.66}$$

L'utilisation de cette formulation permet de réduire le nombre d'opérations nécessaires, ainsi que le nombre d'accès aux mémoires stockant les informations A_v et m_{cv}. En contrepartie, il est nécessaire de mémoriser les L ensembles de messages $m^i_{jc},\ j \in V'_c$ le temps du traitement d'une fenêtre du code externe. Dans la suite de ce manuscrit nous illustrerons les simplifications possibles quand un algorithme sous-optimal est utilisé.

En terme de performance, ce séquencement permet une convergence plus rapide que dans le cas où les codes interne et externe sont décodés séparément. Un exemple de courbes de performances est illustré sur la figure 2.25. Les performances sont comparées dans le cas d'un algorithme de décodage série classique (décodage du

code interne puis du code externe : *Turbo BP* (TBP)), et d'un algorithme avec le séquencement proposé (décodage conjoint des deux codes *Turbo Layered BP* (TLBP)). Les simulations illustrent le gain en vitesse de convergence de l'algorithme TLBP com-

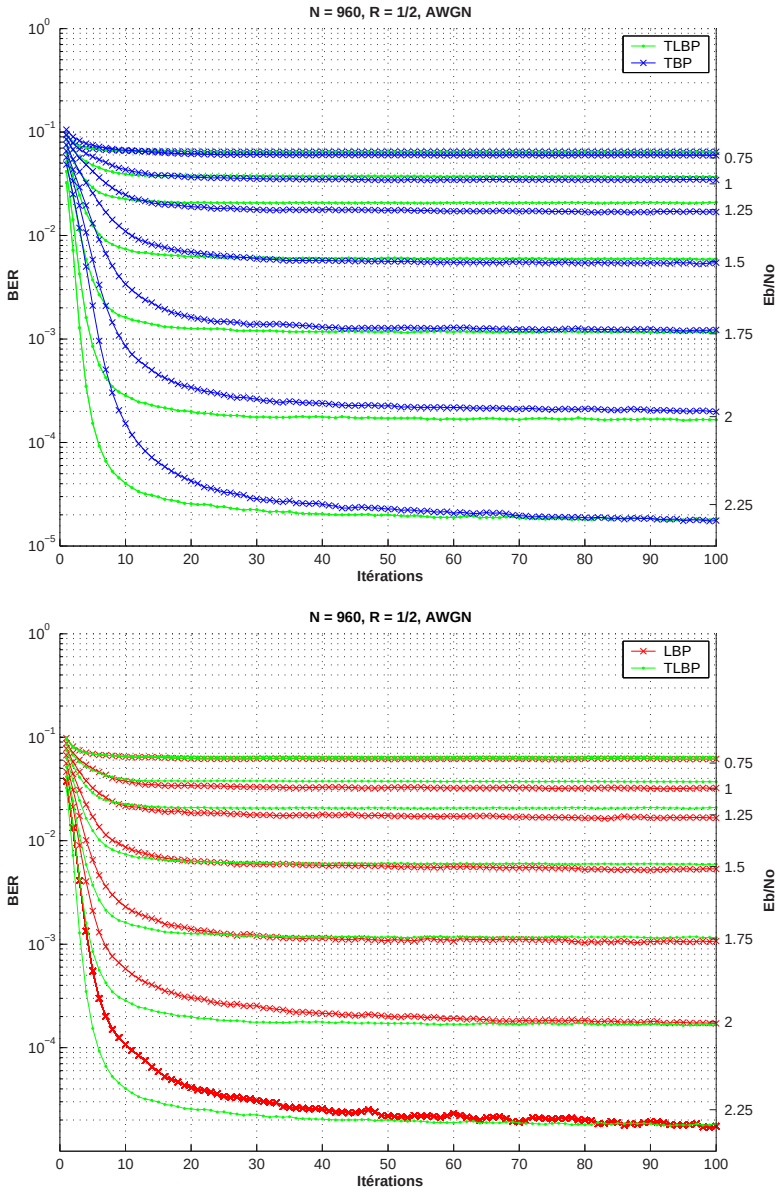


FIG. 2.25 – Comparaison de performances entre les différents algorithmes étudiés : *Layered BP* (LBP), *Turbo BP* (TBP) et *Turbo Layered BP* (TLBP). Le séquencement TLBP qui correspond à un décodage conjoint des codes interne et externe apporte la meilleure convergence.

paré à l'algorithme TBP. On peut aussi noter que le séquencement proposé converge plus rapidement que le séquencement *Layered BP*. Ce phénomène peut s'expliquer à la fois par la mise à jour séquentielle des noeuds de données (systématiques) et

par la propagation des messages à travers le graphe de l'accumulateur. En effet, dans le cas de l'algorithme *Layered BP*, l'information de décodage est une fonction de l'équation de parité en cours de résolution et de l'information de décodage des équations précédentes. Dans le cas du *Turbo Layered BP*, l'information de décodage est aussi une fonction de l'information de décodage des équations précédentes mais également de l'ensemble des équations de la fenêtre en cours de décodage. Cet échange d'information est amélioré par le brassage des messages à travers le treillis.

Du point de vue réalisation matérielle l'ordonnancement proposé est aussi très intéressant. Le fait de décoder le code par fenêtres réduit sensiblement la mémoire requise pour le stockage des messages se propageant sur les branches. En effet dans le cas du séquencement proposé, il n'est plus nécessaire de mémoriser les messages de branches associés au graphe de la partie bi-diagonale de la matrice de contrôle de parité. De plus, la mémoire nécessaire pour le stockage de l'information *a posteriori* est réduite. La mise à jour des noeuds de données intervenant dans le treillis n'étant pas nécessaire[10] pour le décodage, la profondeur de cette mémoire est réduite à K. Cependant, il est nécessaire de mémoriser les observations du canal associées aux M noeuds de données intervenant dans le treillis. Le gain ici intervient donc par la différence de quantification entre l'information *a posteriori* et l'observation du canal. La réalisation matérielle d'un décodeur utilisant le séquencement proposé sera détaillée dans le chapitre 4.

Un nouvelle ordonnancement a été introduit dans ce paragraphe. De la même manière que le séquencement *Layered BP*, la mise en oeuvre de cet ordonnancement exploite la structure et la spécificité de la matrice de contrôle de parité. Le but de cette approche consiste à utiliser au maximum les caractéristiques du code pour définir un ordonnancement de décodage. Une étude d'architecture et de réalisation matérielle sera illustrée dans la suite du document. Les avantages procurés par ce type de séquencement seront soulignés.

2.4 Conclusion

Ce chapitre a présenté les codes LDPC structurés et plus particulièrement les codes de type Repeat Accumulate. Les motivations du choix d'une telle structure ont tout d'abord été illustrées. A partir de règles de construction de codes LDPC communément utilisées, une structure de matrice de contrôle de parité avec une forme bi-diagonale apparaît naturellement. Dans une optique de réalisation matérielle, une caractérisation simple du code est requise. La définition de la matrice de contrôle de parité à partir de matrices identité permutées semble offrir une très bonne solution.

[10] On a donc un comportement qui peut s'apparenter à des noeuds de degré 1

L'analyse de certaines propriétés de la structure de codage choisie a été illustrée. En particulier, l'étude des poids des mots de code issus du codage de mots de poids faibles a permis la définition de règles de construction de codes. De la même manière, l'étude de la détection des cycles dans le graphe du code à partir du graphe de base et des coefficients de permutation, nous a conduit à la définition d'un algorithme de construction incrémentale de codes. L'ensemble de ces études a montré l'importance du choix de certains paramètres du code tel que m, le nombre de sous matrices nulles ou identité par colonne de la matrice de contrôle de parité. Ce paramètre tout comme le profil d'irrégularité du code, intervient dans la borne de la distance minimale et plus généralement dans le comportement du code à très faible probabilité d'erreur.

Dans une dernière partie, ce chapitre s'est intéressé au décodage de la structure de codage étudiée. Le code ayant des propriétés structurelles particulières et connues *a priori*, l'algorithme et le séquencement de décodage peut être conçu à partir de ces hypothèses. L'algorithme *Layered BP*, dérivé de l'ordonnancement *shuffle*, et largement discuté dans la littérature a été dans un premier temps présenté. Par cette illustration, il a été montré comment le séquencement pouvait tirer partie de la structure du code. Dans un second temps, un nouvel ordonnancement a été développé. Celui-ci utilise deux propriétés du code, la construction par bloc et l'existence d'une structure bi-diagonale dans la matrice de contrôle de parité. Par application des concepts utilisés dans l'algorithme *Layered BP*, relatifs à la mise à jour des données , et ceux du décodage d'une concaténation série de deux codes, un algorithme efficace a été dérivé. Le fait de considérer le code externe comme un code convolutif récursif à deux états a facilité la définition de l'algorithme noté *Turbo Layered BP*. La suite de ce manuscrit s'attache donc à la définition conjointe d'architectures et de règles de construction pour des codes LDPC structurés décodés par les algorithmes présentés dans ce chapitre.

Chapitre 3

Étude théorique d'architectures pour les codes LDPC structurés

Résumé

Ce troisième chapitre a pour objectif d'analyser des architectures de décodage relatives à l'algorithme *Layered BP* et *Turbo Layered BP*. Différentes stratégies sont illustrées et analysées. Les calculs théoriques des débits ainsi que les contraintes posées sur la définition du code sont décrits. Dans le cadre de l'algorithme *Turbo Layered BP*, la définition d'une architecture générique pouvant supporter des rendements de codage et des degrés de connexion de noeuds de contrôle différents est proposée. Une dernière partie résume nos travaux sur des architectures associées à des concaténations série de codes hétérogènes.

3.1 Motivations

La chapitre précèdent a illustré le choix d'un type de code. A cette structure, on peut associer plusieurs algorithmes de décodage dont les algorithmes *Layered BP* et *Turbo Layered BP*. Il a été démontré l'intérêt de définir conjointement le code et une stratégie de décodage. Ce concept peut être aussi étendu à la conception d'une architecture matérielle de décodeur. Dans le cas des Turbo-codes parallèles, ces techniques de définition conjointe de l'entrelaceur et de l'architecture sont bien connues et largement utilisées. Les entrelaceurs peuvent, par exemple, être contraints pour permettre une architecture et un décodage efficace du code [13, 63]. Le principe consiste à construire un entrelaceur qui permet la lecture simultanée de p informations extrinsèques dans des bancs mémoires différents, dans l'ordre naturel et l'ordre entrelacé. Ce raisonnement peut être appliqué à d'autres contraintes liées, par exemple, à un ordonnancement donné [36, 63]. Dans le cas de code LDPC, cette méthodologie de conception est aussi régulièrement mise en oeuvre [64, 67].

Ce chapitre a pour objectif de mettre en évidence l'intérêt d'une telle méthode de conception. Nous proposons dans un premier temps une étude d'architecture associée à l'algorithme *Layered BP* dans le cas des codes LDPC structurés présentés. Des règles de construction seront dérivées à partir de considérations matérielles. Dans un deuxième temps, nous analyserons différentes architectures associées à l'algorithme *Turbo Layered BP*. Cette méthodologie de conception illustrera les architectures possibles quand le code LDPC est décomposé en une concatenation de codes hybrides.

3.2 Étude d'architectures associées à l'algorithme Layered BP

Avant de décrire des architectures de décodage, nous proposons de fixer des hypothèses de travail. Tout d'abord nous souhaitons que l'architecture soit suffisamment générique pour permettre le décodage d'un ensemble de codes de tailles, et de rendements de codage différents. Les paramètres m et z doivent donc pouvoir varier en fonction du code. Le processeur de résolution d'équation de parité doit quant à lui être capable de traiter un nombre d'éléments par équations variables. En résumé, l'architecture considérée doit être capable de traiter un ensemble de cas, cet ensemble pouvant être plus ou moins important. A titre d'exemple, le code LDPC de la norme IEEE 802.16e est défini pour 18 tailles différentes [21].

Dans un premier temps nous proposons de nous intéresser à une architecture générique de décodeur. Dans un second temps, nous nous focaliserons sur la définition d'une architecture cherchant à augmenter le débit utile de décodage. Nous illustrerons les conséquences du choix d'une telle architecture sur les critères de sélection des codes.

3.2.1 Architecture générique de décodage

Nous considérons des codes LDPC structurés dont les matrices de contrôle de parité sont construites à partir de matrices nulles et de matrices identité permutées de taille $z \times z$. Par hypothèse le décodage du code se fait par un algorithme de type *Layered BP* suivant les équations de décodage décrites dans le chapitre précédent (section 2.3.1).

Afin de définir un décodeur générique capable de traiter un ensemble de degrés de connexion de noeuds de contrôle différents, nous considérons un processeur CNP série capable de traiter une équation de parité en L_c cycles. Par hypothèse on appelle CNP le processeur de décodage décrit sur la figure 2.17. Le paramètre L_c correspond à l'ensemble des cycles nécessaires pour le chargement, le traitement et le stockage des différentes informations. En considérant un décodeur capable de traiter deux trames consécutives sans latence, on peut déduire le débit utile D d'une architecture générique dans le cas de la résolution séquentielle des équations de parité. Le nombre de cycles

nécessaires pour résoudre les M équations de parité est égale à ML_c. Le débit utile D peut donc s'exprimer par :

$$D = \frac{RN}{ML_c it} f_{clk} \tag{3.1}$$

où f_{clk} est la fréquence d'horloge qui cadence le décodeur et où it est le nombre d'itérations de décodage. On considère ici qu'un cycle correspond à un cycle d'horloge. Comme nous l'avons mentionné précédemment, un niveau de parallélisme égal à z peut être mis en oeuvre dans le cas des codes considérés. Le débit utile s'exprime alors :

$$D = z\frac{RN}{ML_c it} f_{clk} \tag{3.2}$$

Cette expression du débit utile illustre l'importance du paramètre L_c. Outre le facteur de parallélisme p, ce paramètre est le seul paramètre dépendant de l'architecture choisie et plus précisément du processeur de résolution d'équations de parité. Si l'on souhaite une architecture générique capable de traiter un ensemble de codes, le processeur CNP doit être un processeur série avec une mémoire de profondeur $d_c^{\max}$, égale au degré maximum d'un noeud de contrôle. Dans ce cas, le nombre de cycles de traitement d'une équation de parité L_c est proportionnel au degré de connexion du noeud de contrôle considéré. En considérant une architecture série de CNP, le paramètre L_c peut s'exprimer :

$$L_c = 2\sum_i \tilde{\rho}_i i + \epsilon \tag{3.3}$$

où $\sum_i \tilde{\rho}_i i$ représente le degré moyen de connexion des noeuds de contrôle et ϵ une latence liée à l'architecture. Dans la suite de cette section nous utiliserons la notation d_c pour le degré moyen de connexion des noeuds de contrôle. La démonstration de ce calcul sera par la suite mise en évidence. A titre d'exemple, si on considère l'architecture présentée dans [67] où $\epsilon = 5$. Le code considéré est un code de rendement $R = 1/2$, de taille $N = 2304$ définie dans la norme IEEE 802.16e [21] avec $z = 96$ et $\sum_i \tilde{\rho}_i i = 6.33$. Le débit utile pour 20 itérations de décodage est alors égal $0.2717 f_{clk}$ bit/s [(1)].

La définition d'une telle architecture nous amène à discuter de quelques points clefs concernant la réalisation matérielle. Tout d'abord nous avons évoqué la possibilité d'une parallélisation de niveau z (i.e. z processeurs fonctionnant en parallèle). Un tel niveau de parallélisme n'est pas sans poser des problèmes de réalisation. Tout d'abord, l'architecture doit être capable de lire/écrire dans une mémoire z valeurs simultanément. Cette contrainte impose donc l'utilisation de mémoires multi-ports (divisées par exemple en bancs) dont le contrôle et la mise oeuvre peut être assez complexe. D'autre part, la complexité de connexion des mémoires aux organes de décodage par un réseau de permutation n'est pas négligeable. En effet le réseau de permutation doit router z messages simultanément. Même si le fait d'utiliser des

[(1)] Dans l'ensemble du manuscrit le débit sera exprimé en fonction de la fréquence d'horloge, celle-ci étant une fonction de la cible matérielle choisie

matrices identité circulairement permutées permet l'utilisation de *barrel shifter*[(2)], la complexité de mise en oeuvre est assez importante. A titre d'exemple la surface de silicium allouée à cette fonction est de l'ordre de 10% sur une technologie 0.13 μm dans le cas de réalisations sur ASIC (Application-Specific Integrated Circuit) présentées dans [67] et [95].

Les codes dont les matrices de contrôle de parité sont construites à partir de matrices identité permutées et qui sont décodés par un algorithme *Layered BP* permettent le choix entre différents compromis entre débit et coût matériel. Si l'on veut décoder le code avec un débit le plus élevé possible, il est nécessaire d'utiliser pleinement le degré de parallélisme offert par le code pour toutes les tailles envisagées. En d'autres termes, il faut dimensionner une architecture dont le parallélisme peut atteindre le plus grand z caractérisant le code. Cette solution permet d'obtenir un débit augmentant linéairement avec z pour un rendement donné. Ce type d'approche est proposé dans [67] et [95] dans le cadre d'un décodeur générique pour les codes LDPC de la norme IEEE 802.16e et IEEE 802.11n. Cette architecture n'est pas très efficace au vu des ressources matérielles requises. Dimensionner l'architecture pour le pire cas, c'est à dire $z = z_{\max}$ revient à considérer une architecture avec un nombre $n_p = z_{\max}$ processeurs CNP. Quand le paramètre z du code est inférieur à $z_{\max}$, une partie de la ressource matérielle n'est pas utilisée. Pour permettre le traitement de plusieurs rendements, le CNP série doit être dimensionné pour $d_c^{\max}$ (On rappelle que ce paramètre joue sur la profondeur du buffer du processeur). Même si le processeur CNP est très peu complexe quand il est considéré individuellement, les ressources matérielles utilisées peuvent devenir très importante lorsqu'il est dupliqué $n_p = z_{\max}$ fois [96]. A titre d'exemple, la surface de silicium allouée à cette fonction est de l'ordre de 55% sur une technologie 0.13 μm dans le cas de réalisations sur ASIC présentées dans [67].

A l'opposé, une architecture peut être définie pour minimiser la ressource matérielle. Cette stratégie consiste à définir le nombre de processeurs n_p comme le plus petit diviseur commun de l'ensemble des paramètres z caractérisant les codes. Cette architecture n'exploitera pas pleinement le degré de parallélisme offert par la construction du code mais permet une utilisation quasi-optimale de la ressource matérielle. On peut noter que dans le cas des codes LDPC définis dans la norme IEEE 802.11n, le nombre de processeurs en parallèle conduisant à l'utilisation en continu de la ressource est $n_p = z_{min} = 27$. Par contre, dans le cas des codes LDPC définis par la norme IEEE 802.16e, l'ensemble des paramètres z est défini par $z = 4k, k = 6 \ldots 24$, ce qui conduit à un nombre de processeurs permettant une utilisation continue de la ressource de $n_p = 4$. Il est bien entendu possible de choisir un facteur de parallélisme différent. Cependant, du fait de la contrainte imposée par l'algorithme *Layered BP* sur les équations à résoudre en parallèle, l'efficacité et le débit de l'architecture va

[(2)] Un *barrel shifter* est un composant capable de réaliser plusieurs décalages d'un mot binaire en un cycle

varier en fonction des paramètres choisis. A titre d'exemple on peut considérer un cas où $z = 28$ (paramètre IEEE 802.16e) et un nombre de processeurs $n_p = 16$. Dans un premier temps, 16 équations parmi les 28 premières sont décodées en parallèles avec une utilisation de tous les processeurs disponibles. Dans un deuxième temps, il faut résoudre les $28 - 16 = 12$ équations restantes du premier groupe de 28. Celles-ci ne peuvent pas, *a priori*, être résolues en même temps que les $28 - 12 = 16$ équations du deuxième groupe de 28 car, un noeud de données peut apparaître deux fois dans ce groupe. Il est donc nécessaire de décoder les 12 équations du premier groupe, en utilisant seulement 12/16 de la ressource matérielle.

Ces remarques montrent l'importance du choix de la stratégie de décodage à adopter. Une solution intermédiaire consiste à intégrer cette contrainte dans la construction du code où le paramètre z peut être défini comme une fonction linéaire d'un nombre de processeurs n_p, dimensionné pour une complexité visée. Ce type de construction peut mener à une utilisation de la ressource matérielle quasi optimale, tout en garantissant un bon compromis entre le débit et le coût matériel de la réalisation. A titre d'exemple les codes LDPC définis au sein du IEEE 802.11n ont un paramètre $z = 27k,\ k = 1, 2, 3$. Une étude de différentes stratégies de réalisation est illustrée dans [95]. En imposant $n_p = 27$ le débit est trois fois moins important que dans le cas $n_p = 81$ mais avec une complexité de réalisation (Mémoire, Nombre d'éléments logiques (FPGA)) divisée par un facteur légèrement inférieur à 3.

Les paramètres contraignant le débit de l'architecture sont donc le degré de parallélisme p et le paramètre L_c définissant le nombre de cycles nécessaires au décodage d'une équation de parité. Ce paramètre dépend de l'architecture choisie pour le processeur CNP. Si l'on souhaite une architecture générique, capable de traiter n'importe quel code, la solution la plus adaptée reste une architecture série de processeur. Cependant, si l'on autorise quelques contraintes sur la définition du code, le paramètre L_c peut être déterminé de manière à maximiser le débit et l'utilisation de chaque processeur. Cette stratégie de conception conjointe fait l'objet de l'étude présentée dans la section suivante.

3.2.2 Vers une architecture contrainte

Nous avons illustré précédemment une architecture générique de décodeur pour des codes LDPC dont la matrice de contrôle de parité est construite à partir de matrices identité permutées. Dans cette section nous proposons une analyse plus fine de cette architecture en nous autorisant un degré de liberté sur la définition des codes. Le but est ici de définir des contraintes sur les choix des codes pour permettre le décodage suivant une architecture définie préalablement. Dans un premier temps, nous proposons une modélisation des architectures de processeurs CNP. Dans un second temps, des contraintes sur la construction du code seront illustrées dans le but d'améliorer l'efficacité de l'architecture.

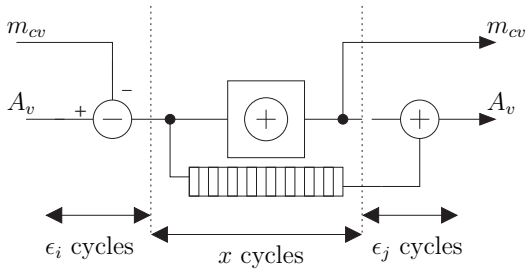


FIG. 3.1 – Modélisation générique d'un processeur CNP adapté pour un algorithme *Layered BP*. Le nombre de cycles nécessaires à chaque traitement est illustré.

3.2.2.1 Étude du processeur de résolution d'équations de parité

Nous rappelons que nous nous focalisons sur une architecture de processeurs CNP dans le cadre d'un décodage par un algorithme de type *Layered BP*. Un modèle d'architecture de processeurs générique est illustré sur la figure 3.1. Le but de ce paragraphe est de proposer des architectures de processeurs CNP avec des spécificités différentes au niveau du temps de traitement, et des ressources matérielles utilisées. Pour modéliser les différentes architectures, nous proposons une synoptique inspirée des travaux de Boutillon *et al.* sur la définition d'architectures pour le décodage de code convolutif avec un algorithme MAP [97]. Nous proposons de détailler trois architectures spécifiques ayant chacune une spécificité. Ces architectures seront notées CNP^{Σ}, CNP^{X} et CNP^{Σ^2} (3).

Architecture CNP^{Σ}

Tout d'abord, nous considérons une architecture série de processeur où le décodage du noeud de contrôle se fait par un algorithme Forward Backward. La synoptique du temps de traitement associée à cette architecture est illustrée sur la figure 3.2. Dans un premier temps, les messages m_{vc} sont calculés (cf figure 3.1). Après ϵ_0 cycles, le message calculé par la soustraction de l'information *a posteriori* par l'information extrinsèque est traité par le processeur forward associé à la résolution de l'équation de parité. Le nombre de cycles nécessaires pour calculer l'ensemble des messages est alors de d_c. Une fois l'ensemble des messages forward calculés et après ϵ_1 cycles, les messages backward peuvent être calculés par le processeur backward associé à la résolution de l'équation de parité. ϵ_2 cycles plus tard, l'information extrinsèque m_{cv} associée à chaque noeud est mise à jour et l'information *a posteriori* A_v est re-estimée après ϵ_3 cycles.

Cette représentation est très utile pour définir et caractériser l'architecture du processeur. Une telle architecture nécessite deux mémoires de type LIFO (Last In First Out). Une première mémoire est utilisée pour stocker les messages m_{vc} et une

(3) Ces notations sont dérivées de celles utilisées dans [97] et [63].

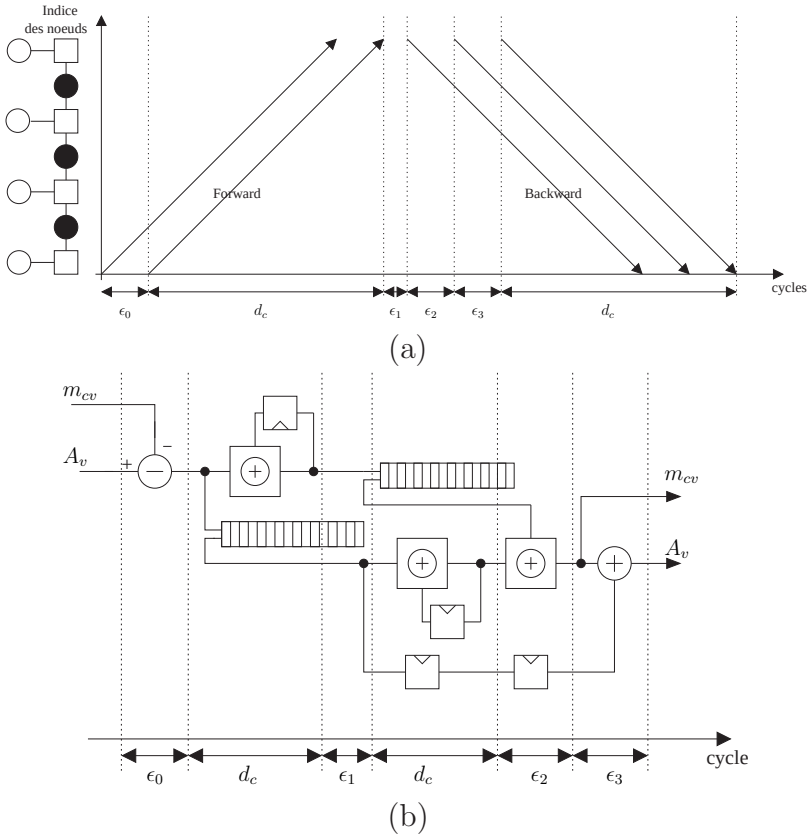


FIG. 3.2 – Synoptique du temps de traitement d'un processeur CNP$^{\Sigma}$ série et exemple d'architecture associée

seconde pour la mémorisation des messages forward. Cette architecture nécessite un seul processeur de calcul qui peut à la fois calculer les messages forward et backward (les équations de décodage sont les mêmes). Concernant le temps de traitement d'une équation de parité L_c, il est égal à $2d_c + \epsilon$, où ϵ est égal à la somme des latences dans le processus de traitement de l'équation de parité. Plus généralement, le paramètre ϵ pourra aussi prendre en compte les cycles nécessaires à la lecture et écriture des informations de décodage, incluant la latence de propagation dans le réseau de permutation. Ce type processeur sera noté CNP$^{\Sigma}$. En utilisant cette architecture, le débit utile peut s'exprimer par :

$$D = p\frac{RN}{M(2d_c + \epsilon)it}f_{clk} \tag{3.4}$$

Si l'on souhaite améliorer le débit, il est nécessaire de réduire le temps de traitement de chaque équation de parité. L'illustration d'une telle solution est décrite dans le paragraphe suivant.

Architecture CNP^{X}

De manière à augmenter le débit utile, une solution peut consister à considérer

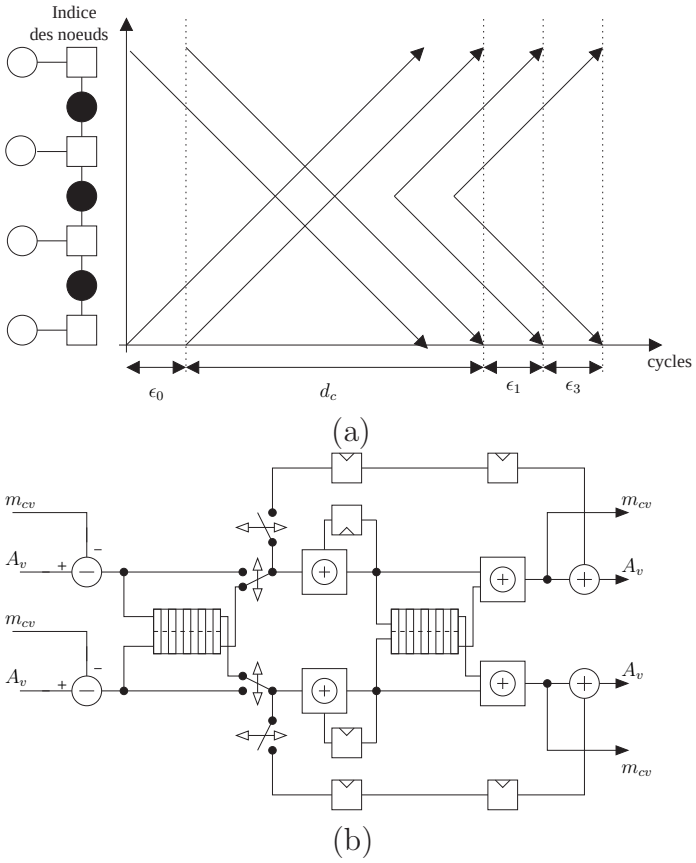


FIG. 3.3 – Synoptique du temps de traitement d'un processeur CNP^X série avec un parallélisme interne de 2 et exemple d'architecture associée

l'architecture CNP décrite par le séquencement illustré sur la figure 3.3. Le processeur CNP de ce type sera noté CNP^X. Ce séquencement a la particularité de faire fonctionner les processeurs forward et backward en même temps, sur deux parties du graphe distinctes. Cette architecture a les mêmes besoins en ressources de mémorisation que dans le cas précédent mais nécessite deux processeurs, deux additionneurs et soustracteurs fonctionnant en parallèle. Ce type de processeur nécessite également que l'architecture globale puisse traiter la lecture, l'écriture et le routage de deux messages en parallèle. Concernant le temps de traitement de l'équation de parité, celui ci est égal à $d_c + \epsilon$, d'où un débit utile égal à :

$$D = p\frac{RN}{M(d_c + \epsilon)it}f_{clk} \tag{3.5}$$

Il faut noter que cette architecture revient à utiliser un processeur avec un parallèlisme interne de 2. Même si théoriquement ce séquencement n'impose aucune contrainte sur le code, en pratique il peut être judicieux de fixer un degré de noeuds de contrôle pair. Dans ce cas, le contrôle logique mis en oeuvre pour ce type de processeur est simplifié. Il faut aussi noter que, en comparaison à l'architecture précédente et pour

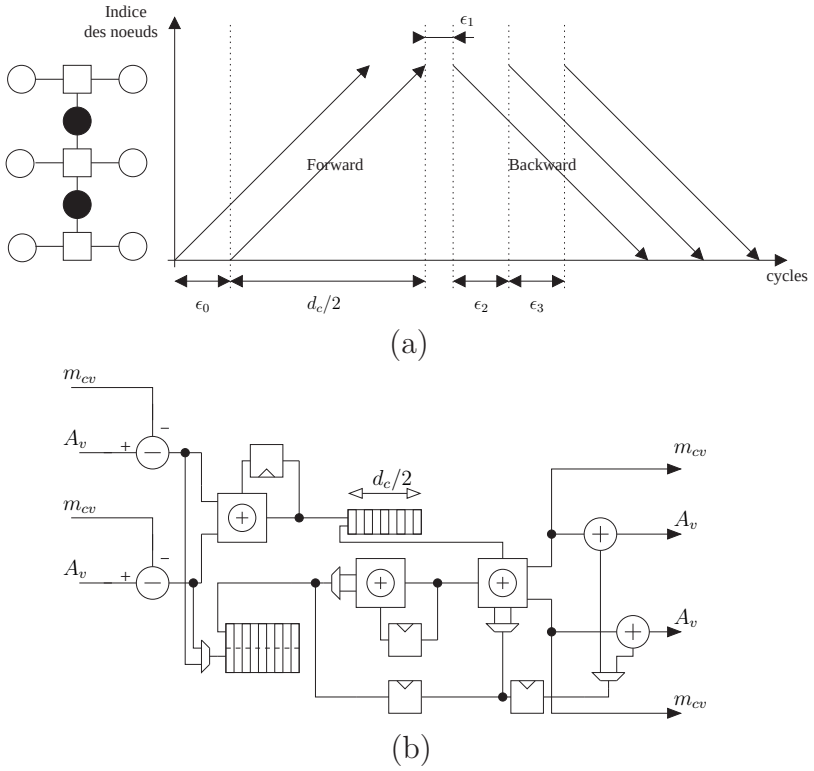


FIG. 3.4 – Synoptique du temps de traitement d'un processeur CNP$^{\Sigma^2}$ série où les noeuds sont traités par couple et exemple d'architecture associée.

un même niveau de parallélisme p, l'architecture globale du décodeur utilisant ce type de processeur devra traiter deux fois plus de données. Cette contrainte impose donc un réseau de permutation plus complexe. Dans le cas où le débit est un paramètre plus important que la complexité de réalisation, ce type d'architecture peut être particulièrement judicieux.

Architecture CNP$^{\Sigma^2}$

Dans le même esprit, une architecture où le traitement des noeuds est réalisé par couple est illustrée sur la figure 3.4. Le processeur CNP de ce type sera noté CNP$^{\Sigma^2}$. Le temps de traitement d'une équation est alors égal à $L_c = d_c + \epsilon$. Tout comme l'architecture précédente, cette stratégie est équivalente à l'utilisation d'un processeur avec un parallélisme interne de 2. Il est donc nécessaire que l'architecture globale du décodeur puisse traiter la lecture, l'écriture et le routage de deux messages en parallèle. L'intérêt de cette architecture est la réduction du nombre de messages aller à mémoriser. Cette réduction d'un facteur deux permet de diminuer les ressources mémoires nécessaires au niveau du buffer (cf figure 3.4). Cette propriété est d'autant plus intéressante que le degré maximum de connexion des noeuds de contrôle est important et que le nombre de processeurs dupliqués n_p est grand. Cette réduction de

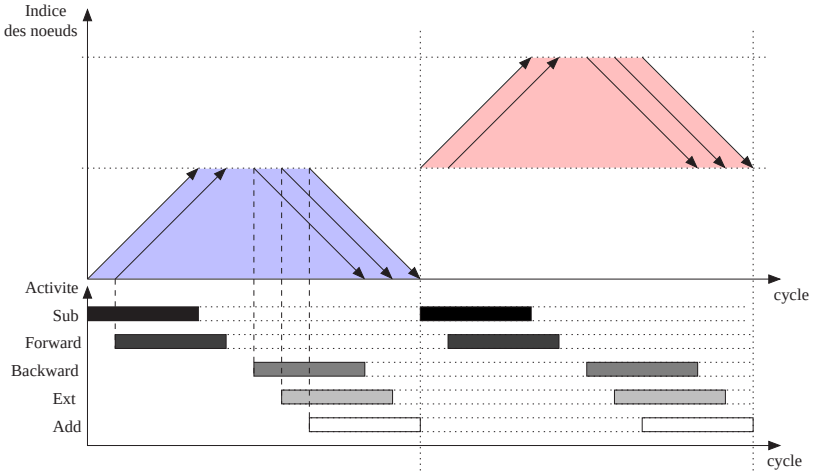


FIG. 3.5 – Mise en évidence d'un ordonnancement avec un processeur CNP$^{\Sigma}$ où la ressource n'est pas utilisée de manière efficace. L'utilisation de la ressource pour chaque processeur est illustrée.

mémoire s'accompagne néanmoins d'une augmentation de la complexité de la fonction de traitement d'un noeud de contrôle.

Conclusions

Le choix entre ces différentes réalisations du processeur de parité doit donc être pris en compte en fonction du débit et de la complexité de la réalisation matérielle choisie. L'analyse de ces différentes architectures permet de mettre en évidence un point important concernant l'architecture. Tout d'abord, le séquencement global du décodage a pour hypothèse une résolution séquentielle d'une équation ou d'un groupe d'équations. Le séquencement oblige donc d'attendre la fin du décodage d'une équation avant de commencer une nouvelle, comme illustré sur la figure 3.5. Ce séquencement ne permet donc pas l'utilisation d'une manière efficace de la ressource puisque les processeurs ne sont pas utilisés de manière continue. Cette propriété impacte aussi le débit utile de l'architecture, limité par le terme de latence ϵMit. A partir de cette observation nous proposons d'étudier un séquencement qui permet une utilisation efficace de la ressource accompagnée d'une amélioration du débit de décodage.

3.2.2.2 Maximisation du débit et notion d'activité

L'utilisation de manière efficace de la ressource est reliée au concept d'activité introduit dans [63]. L'activité est définie comme une mesure de l'efficacité de l'architecture. Celle-ci est une fonction de l'utilisation moyenne des ressources de calculs pendant l'exécution de l'algorithme. Comme nous avons pu le mentionner précédemment, les architectures séries associées à un algorithme *Layered BP* peuvent être optimisées

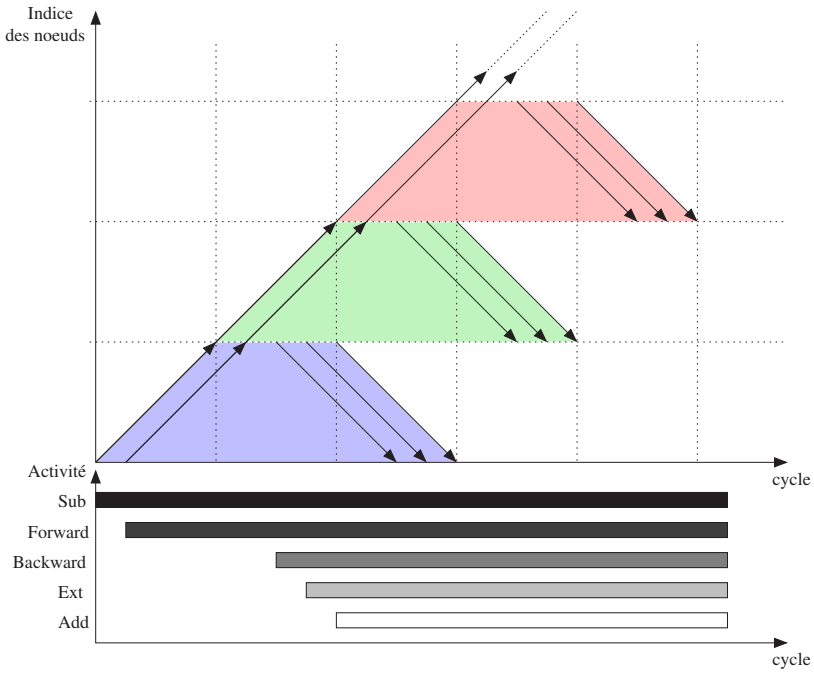


FIG. 3.6 – Séquencement de l'algorithme de décodage utilisant un processeur CNP$^{\sigma}$ permettant l'utilisation en continu des ressources matérielles.

de manière à augmenter l'activité de chaque processeur.

On considère dans un premier temps le séquencement illustré sur la figure 3.6 utilisant un processeur CNP$^{\Sigma}$. Cet ordonnancement permet de maximiser l'activité du processeur de décodage. Les processeurs sont alimentés de manière continue, ce qui permet à la fois une utilisation quasi-optimale de la ressource mais également une augmentation du débit. Le débit peut dans ce cas s'exprimer par :

$$D = \frac{RN}{(d_c + \epsilon) + \frac{Md_c}{p} it} f_{clk} \tag{3.6}$$

En remarquant que $(d_c+\epsilon) \ll itMd_c/p$, le gain en débit apporté par un tel ordonnancement comparé à un ordonnancement équivalent à celui illustré sur la figure 3.5 est de l'ordre de $1+\epsilon/d_c$. Il faut cependant noter que le processeur CNP doit être pipeliné de manière à accepter des données en flux continu. Le processeur illustré sur la figure 3.2 est un processeur acceptant le pipeline. Ce type de séquencement est donc très intéressant du point de vue du débit et de l'efficacité de l'architecture. Cependant, les règles d'activation des noeuds inhérentes à l'algorithme de décodage imposent certaines contraintes sur la définition du code. Ces règles dépendent de l'architecture choisie. Pour illustrer comment l'architecture choisie influence le choix du code, nous proposons d'étudier deux stratégies différentes. La première consiste à considérer une architecture avec un nombre de processeurs égal à $n_p = z_{\max}$. Le but de cette architecture est donc d'utiliser pleinement le parallélisme offert par la construction du

code au détriment de l'efficacité. La seconde architecture étudiée cherche quant à elle à trouver un compromis entre efficacité et débit.

3.2.2.3 Architecture parallèle $n_p = z_{\max}$

Ce type de séquencement est très intéressant du point de vue du débit. Les règles d'activation des noeuds inhérentes à l'algorithme de décodage rendent cependant difficile l'application de ce concept. En effet, si on considère un parallélisme de z, le décodage du groupe de z équations de parité ne peut se faire que si ces noeuds ne sont pas déjà en cours de décodage. Ce problème a été traité dans le cadre de la normalisation du code LDPC du groupe IEEE 802.16e et illustré dans [67]. La solution proposée consiste à définir un code dont la matrice de contrôle de parité ne comporte pas deux matrices identité permutées successives dans une colonne. Le cas de la matrice bi-diagonale est traité par permutation de groupes de colonnes.

□ *Exemple:* Soit la matrice de contrôle de parité suivante :

$$\mathbf{H} = \begin{array}{c} \left[\begin{array}{cccccccccccc} \mathbf{I}_{\delta_0} & \mathbf{I}_{\delta_1} & - & - & - & - & \mathbf{I}_0 & - & - & - & - & \mathbf{I}_1 \\ - & - & \mathbf{I}_{\delta_2} & - & - & \mathbf{I}_{\delta_3} & \mathbf{I}_0 & \mathbf{I}_0 & - & - & - & - \\ - & - & \mathbf{I}_{\delta_4} & \mathbf{I}_{\delta_5} & - & - & - & \mathbf{I}_0 & \mathbf{I}_0 & - & - & - \\ - & - & - & - & \mathbf{I}_{\delta_6} & \mathbf{I}_{\delta_7} & - & - & \mathbf{I}_0 & \mathbf{I}_0 & - & - \\ - & \mathbf{I}_{\delta_8} & - & - & \mathbf{I}_{\delta_9} & - & - & - & - & \mathbf{I}_0 & \mathbf{I}_0 & - \\ \mathbf{I}_{\delta_{10}} & - & - & \mathbf{I}_{\delta_{11}} & - & - & - & - & - & - & \mathbf{I}_0 & \mathbf{I}_0 \end{array}\right] \end{array} \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}$$

Ce code permet un décodage pipline par l'algorithme *Layered BP* si on considère un décodage par groupe de z équations par ordre croissant de la matrice de contrôle de parité permutée suivante :

$$\mathbf{H}^p = \begin{array}{c} \left[\begin{array}{cccccccccccc} \mathbf{I}_{\delta_0} & \mathbf{I}_{\delta_1} & - & - & - & - & \mathbf{I}_0 & - & - & - & - & \mathbf{I}_1 \\ - & - & \mathbf{I}_{\delta_4} & \mathbf{I}_{\delta_5} & - & - & - & \mathbf{I}_0 & \mathbf{I}_0 & - & - & - \\ - & \mathbf{I}_{\delta_8} & - & - & \mathbf{I}_{\delta_9} & - & - & - & - & \mathbf{I}_0 & \mathbf{I}_0 & - \\ - & - & \mathbf{I}_{\delta_2} & - & - & \mathbf{I}_{\delta_3} & \mathbf{I}_0 & \mathbf{I}_0 & - & - & - & - \\ \mathbf{I}_{\delta_{10}} & - & - & \mathbf{I}_{\delta_{11}} & - & - & - & - & - & - & \mathbf{I}_0 & \mathbf{I}_0 \\ - & - & - & - & \mathbf{I}_{\delta_6} & \mathbf{I}_{\delta_7} & - & - & \mathbf{I}_0 & \mathbf{I}_0 & - & - \end{array}\right] \end{array} \begin{array}{c} 0 \\ 2 \\ 4 \\ 1 \\ 5 \\ 3 \end{array}$$

La matrice $\mathbf{H}^p$ ne comporte pas deux matrices identité permutées successives dans une colonne. Il est important de noter que la matrice $\mathbf{H}^p$ représente le même code que celui défini par $\mathbf{H}$. □

Une telle solution permet le décodage de type *Layered BP* tout en maximisant l'activité de chaque processeur, ce qui se traduit par une augmentation du débit comparé à un séquencement traditionnel. Cependant, la mise en oeuvre de cette règle nécessite une définition particulière du code. Cette règle de construction peut être difficilement mise en oeuvre quand la matrice des coefficients de permutation est dense ou que

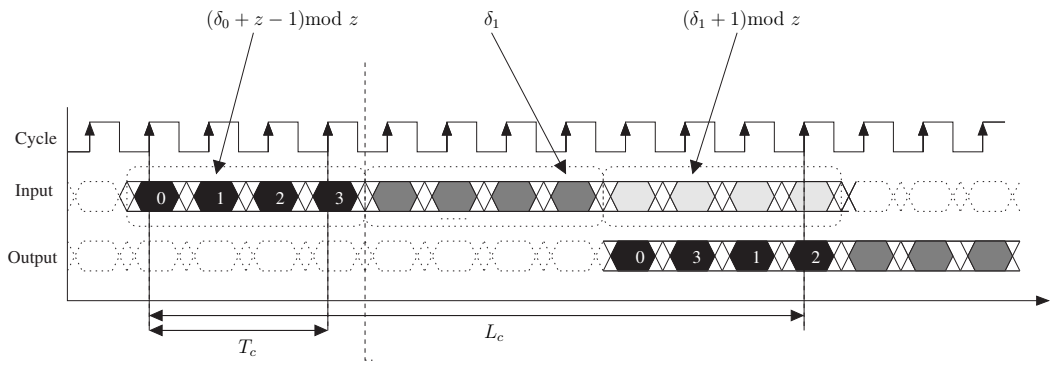


FIG. 3.7 – Exemple de séquencement pipeliné à l'interface entre deux matrices identité permutées $\mathbf{I}_{\delta_0}$ et $\mathbf{I}_{\delta_1}$. Pour respecter la règle énoncée, il est nécessaire que $(\delta_0 + z - 1)\text{mod } z \neq \delta_1$, $(\delta_0 + z - 1)\text{mod } z \neq (\delta_1 + 1)\text{mod } z$. Dans ce cas le rapport $\lceil L/T_c \rceil = 3$

le paramètre m est petit[(4)]. Par exemple, dans le cadre du code LDPC de la norme IEEE 802.16e, seuls deux ensembles de codes ($R = 1/2$ et $2/3$) respectent cette règle. Cette propriété amène donc à considérer des décodeurs capables à la fois de traiter un algorithme de type *Layered BP* autorisant le pipeline et un algorithme BP par inondation dont l'architecture permet elle aussi le pipeline. Ce type d'architecture est donc relativement complexe à mettre oeuvre mais possède l'avantage de permettre des débits élevés. Ce type d'architecture a été proposé dans [67].

3.2.2.4 Architecture semi-parallèle où $n_p < z_{\min}$

Nous proposons d'étudier une architecture qui optimise conjointement le débit et l'activité, tout en minimisant la ressource matérielle requise. Le nombre de processeurs dupliqués n_p est donc inférieur à la plus petite valeur du paramètre z. Soit T_c le nombre de cycles nécessaires au chargement d'une équation de parité dans le processeur (cf figure 3.7). La règle relative au décodage *Layered BP* peut s'exprimer de la manière suivante :

> *Un ensemble de noeuds de données connectés à un ensemble de noeuds de contrôle décodés à l'instant i ne doit pas être connecté à l'ensemble de noeuds de contrôle décodés aux instants $i + k, k = 1 \ldots \lceil \frac{L}{T_c} \rceil - 1$*

On considère la valeur $\lceil \frac{L}{T_c} \rceil - 1$ n'ayant pas connaissance *a priori* de l'ordre d'entrée et de sortie des données. Cet ordre est dépendant de l'architecture choisie pour le processeur CNP. Il faut noter que cette règle permet aussi d'éviter un conflit de lecture et écriture simultanée dans une même position de la mémoire.

Les configurations pouvant résulter en un non-respect de la règle de décodage interviennent dans les cas où la matrice de contrôle de parité comporte deux matrices

(4) On peut montrer très facilement que le plus petit m autorisant une construction permettant le séquencement pipeline est $m = 5$ dans le cas où la matrice est de type bi diagonale

identité permutées successives dans une colonne[5] :

$$\mathbf{H} = \begin{bmatrix} & \vdots & \\ \cdots & \mathbf{I}_{\delta_0} & \cdots \\ & \mathbf{I}_{\delta_1} & \\ & \vdots & \end{bmatrix}$$

Remarque : Si l'on veut éviter une latence entre deux itérations, il est aussi nécessaire de considérer les configurations où il existe des matrices identité permutées en début et fin d'une même colonne.

Un moyen de s'affranchir de cette contrainte est de définir des règles sur les coefficients δ_0 et δ_1 en fonction du parallélisme souhaité et des paramètres L_c et T_c. Dans un premier temps, pour faciliter la compréhension, nous proposons d'étudier le cas d'une architecture comprenant un seul processeur ($n_p = 1$) de décodage autorisant le pipeline. Sous ces hypothèses, et en connaissant le coefficient δ_0, on peut définir une règle sur la détermination du coefficient δ_1 :

$$\delta_1 \neq (\delta_0 - k) \bmod z, \ \forall k = 1 \cdots \left\lceil \frac{L_c}{T_c} \right\rceil - 1 \tag{3.7}$$

Cette équation est la formulation mathématique de la règle énoncée précédemment et est illustrée sur la figure 3.7. On peut noter que cette règle élimine $\lceil L_c/T_c \rceil - 1$ coefficients parmi les z possibles. Cette règle peut être généralisée dans le cas d'un parallélisme de p diviseur de z :

$$\delta_1 \neq (\delta_0 - k') \bmod z, \ \forall k' = 1 \cdots p \left\lceil \frac{L_c}{T_c} \right\rceil - 1 \tag{3.8}$$

Le nombre de coefficients interdits est égal à $p(\lceil L_c/T_c \rceil) - 1)$ ce qui permet de borner le degré de parallélisme de la manière suivante :

$$p < \frac{z+1}{\left\lceil \frac{L_c}{T_c} \right\rceil} \tag{3.9}$$

L'ensemble de ces relations fait apparaître l'importance du paramètre $\lceil L_c/T_c \rceil$. Suivant l'architecture de processeurs CNP choisie, ce paramètre varie. En effet, si on considère le séquencement illustré sur la figure 3.2 utilisant un processeur CNP$^{\Sigma}$, le terme $\lceil L_c/T_c \rceil$ est égal à :

$$\left\lceil \frac{L_c}{T_c} \right\rceil = \left\lceil \frac{2d_c + \epsilon}{d_c} \right\rceil = 2 + \left\lceil \frac{\epsilon}{d_c} \right\rceil \tag{3.10}$$

Connaissant donc le paramètre de l'architecture, des règles de construction peuvent être appliquées au code pour permettre un décodage pipeline de type *Layered BP*. Si

[5] On considère le traitement de la matrice de contrôle de parité de haut en bas.

l'on veut un débit donné et que l'on fixe un parallélisme p, on peut estimer le rapport $\lceil \epsilon/d_c \rceil$ maximal admissible :

$$\left\lceil \frac{\epsilon}{d_c} \right\rceil < \frac{z+1}{p} - 2 \tag{3.11}$$

S il utilise une architecture de processeurs de type CNP^X ou CNP$^{\Sigma^2}$ alors le paramètre $\lceil L_c/T_c \rceil$ s'exprime :

$$\left\lceil \frac{L_c}{T_c} \right\rceil = \left\lceil \frac{d_c + \epsilon}{d_c/2} \right\rceil = 2 + 2 \left\lceil \frac{\epsilon}{d_c} \right\rceil \tag{3.12}$$

ce qui conduit à la contrainte suivante pour une parallélisme p donné :

$$\left\lceil \frac{\epsilon}{d_c} \right\rceil < \frac{z+1}{2p} - 1 \tag{3.13}$$

On peut noter que le débit associé à ces deux architectures CNP^X et CNP$^{\Sigma^2}$ est environ deux fois plus élevé que dans le cas où un processeur CNP$^\Sigma$ est utilisé. A débit constant, l'utilisation de ces deux architectures de processeurs est donc moins contraignante pour la définition du code. Il faut bien garder à l'esprit que ce type d'architecture permet certes de diminuer la latence, mais cela au prix d'un traitement par couple des données. Ce traitement peut être plus complexe notamment dans la phase de routage des messages.

Une dernière remarque concerne l'activité des processeurs CNP^X et CNP$^{\Sigma^2}$ dans le cadre de processeurs pipelinés. Si l'on souhaite maximiser l'activité il est préférable d'utiliser le processeur CNP$^{\Sigma^2}$. En effet, comme illustré sur la figure 3.8, les ressources utilisées pour le calcul des informations de sortie à partir des messages forward et backward ne sont pas utilisées de manière continue dans le cas d'un processeur CNP^X. Cette perte d'efficacité sera d'autant plus importante que d_c est petit. Ce résultat est semblable à celui présenté dans [63] dans le cas des Turbo-codes.

A travers un exemple de description conjointe d'une architecture et d'un code, nous avons pu dériver des règles de construction dans le cas d'un algorithme de type *Layered BP* utilisant un processeur CNP pipeliné. Les règles décrites permettent l'utilisation des n_p processeurs avec une activité quasi optimale. Ce type de séquencement est particulièrement adapté quand on cherche à minimiser la complexité du décodeur tout en garantissant un certain débit de fonctionnement. On rappelle que cette architecture permet à la fois d'utiliser les propriétés de réduction du nombre d'itérations de l'algorithme de décodage *Layered BP* et une utilisation continue des processeurs de décodage. Ces deux concepts seront par la suite mis en oeuvre dans le cas du séquencement associé à l'algorithme de décodage *Turbo Layered BP*.

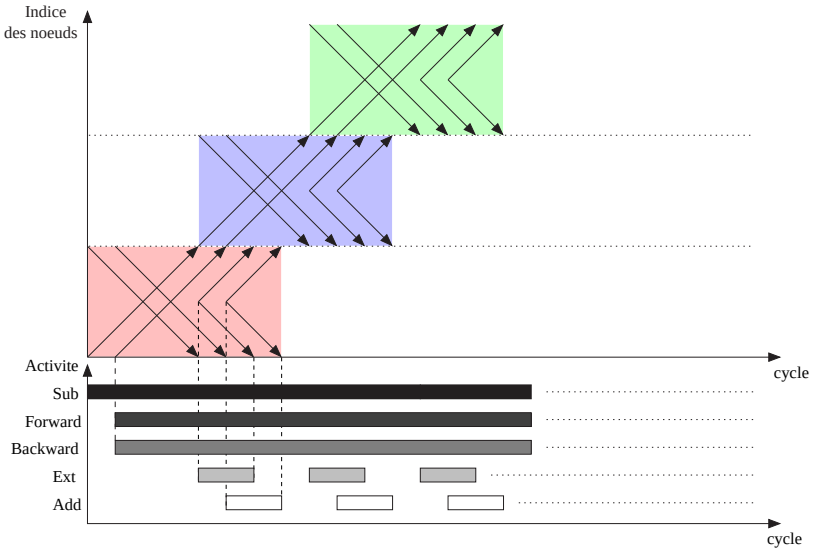


FIG. 3.8 – Séquencement de l'algorithme de décodage utilisant un processeur CNP^X permettant de maximiser l'utilisation des ressources matérielles. Comme le montre la figure, ce séquencement ne permet pas l'utilisation optimale des ressources.

3.3 Étude d'architectures associées à l'algorithme Turbo Layered BP

Dans le chapitre précédent, nous avons montré l'intérêt de l'utilisation de l'algorithme *Turbo Layered BP*. Cet algorithme de décodage consiste en un échange d'information entre deux décodeurs concaténés en série. L'intérêt de cet algorithme est d'autant plus important que le décodage est réalisé par fenêtres. Dans le chapitre précédent, nous avons illustré des règles de construction appliquées au code pour permettre le décodage suivant le séquencement *Turbo Layered BP*. Cette section a pour but d'illustrer une architecture de décodage associée à cet algorithme. Dans un premier temps, des règles de construction et des architectures de décodeurs seront illustrées pour des séquencements mettant en oeuvre du pipeline et du parallélisme. Dans une seconde partie, la généricité de l'architecture sera discutée.

3.3.1 Architecture de décodage

Le décodage par fenêtre du code impose la vérification des équations de parité dans un certain ordre. Nous proposons dans cette section d'illustrer des règles de construction de code pour permettre différents séquencements. Tout d'abord un ordonnancement série sera illustré. Dans un second temps les contraintes liées à l'utilisation de processeurs pipelinés seront décrites. Enfin deux architectures parallèles seront étudiées.

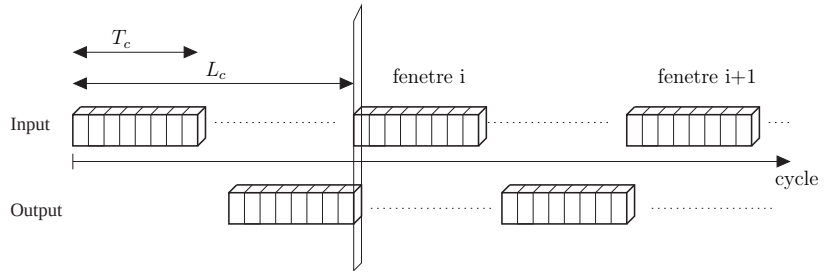


FIG. 3.9 – Illustration du séquencement série. Une fenêtre de décodage peut commencer à être traitée quand le résultat du décodage de la précédente est disponible.

3.3.1.1 Séquencement série

Le séquencement série de l'algorithme consiste à décoder chaque fenêtre de taille L séquentiellement, comme illustré sur le chronograme de la figure 3.9. Le décodage d'une fenêtre comporte une première étape de résolution de L équations de parité dans le sens forward. La résolution des équations de parité permet le décodage de L sections de treillis de l'accumulateur (on considère que les L équations décodées ont été correctement choisies). Une fois le treillis décodé, les L équations de parité peuvent être résolues dans le sens backward. Par hypothèse nous fixons la taille de la fenêtre de décodage L comme un multiple de m, $L = lm$.

Pour permettre l'utilisation de l'algorithme de décodage *Layered Turbo BP*, il est nécessaire que les équations de parité intervenant dans une fenêtre de taille lm ne fassent apparaître un noeud de données qu'une seule fois. Soit $\delta_i,\ i = 0 \ldots q-1$ les q coefficients de permutations associés à q matrices identité permutées d'une colonne de $\mathbf{H_s}$. En utilisant la relation 2.14 et sans perte de généralité en considérant la première fenêtre de décodage de taille lm, les noeuds de données intervenant dans les différentes équations de parité sont les noeuds d'indices :

$$\underbrace{(\delta_0)\text{mod } z \ \cdots \ (\delta_{q-1})\text{mod } z}_{1} \cdots \underbrace{(\delta_0 + l)\text{mod } z \ \cdots \ (\delta_{q-1} + l)\text{mod } z}_{l} \tag{3.14}$$

Pour permettre l'utilisation de l'algorithme de décodage *Layered Turbo BP*, il est nécessaire que l'ensemble de ces indices soient différents. Cette règle impose que les coefficients de permutation caractérisant les matrices identité permutées dans une colonne de $\mathbf{H_s}$ soient distants de l :

$$\delta_i \neq (\delta_j \pm k) \bmod z,\ i \neq j,\ k = 0 \ldots l-1 \tag{3.15}$$

Si on considère que le nombre maximum de matrices identité permutées par colonne de $\mathbf{H_s}$ est égal à $d_v^{\max}$, alors le nombre de coefficients de permutations différents nécessaires est de $ld_v^{\max}$. On a donc la relation suivante sur la valeur du paramètre l :

$$ld_v^{\max} < z \Rightarrow l < \frac{z}{d_v^{\max}} \tag{3.16}$$

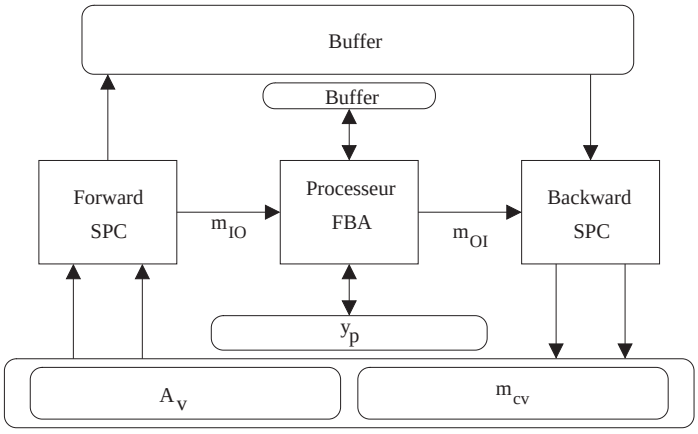


FIG. 3.10 – Schéma de architecture d'un décodeur *Turbo Layered BP* basée sur trois processeurs de décodage. Le processeur annoté *Forward SPC* (Single Parity Check) calcule les messages du code interne vers le code externe m_{IO}. Le processeur traitant les messages de sortie de l'accumulateur est quant à lui annoté *Backward SPC*

Cette relation permet de borner la taille de la fenêtre de décodage en fonction de la distribution d'irrégularité du code et des paramètres m et z choisis pour une taille donnée.

Avant de discuter sur le choix de la taille de la fenêtre de décodage et donc des paramètres l et m, nous proposons d'illustrer une architecture de décodage. Cette architecture est présentée sur la figure 3.10. Le décodeur se décompose en trois processeurs de calcul. Le processeur annoté *Forward SPC* calcule à partir des informations *a posteriori* A_v et des messages m_{cv}, les messages m_{vc} se propageant des noeuds de données vers les noeuds de contrôle. Ces lm messages sont mémorisés et utilisés pour déterminer l'information à propager à l'accumulateur m_{IO}. Le processeur annoté FBA décode une fenêtre de l'accumulateur qui correspond à lm sections de treillis. Ce processeur est associé à une mémoire stockant les métriques de branches nécessaires à la détermination de l'information extrinsèque. Suivant l'architecture de processeurs choisie, cette mémoire est d'une profondeur proportionnelle à lm. Le dernier processeur, annoté *Backward SPC*, réalise une mise à jour de l'information *a posteriori* A_v et des messages m_{cv} à partir de l'information extrinsèque extraite du décodage de l'accumulateur et des messages m_{vc} mémorisés précédemment. La taille de la fenêtre de décodage va influer sur la profondeur des buffers de l'architecture et donc sur la complexité du système de décodage.

En résumé, plus l sera grand, plus la règle de construction du code sera contraignante et les ressources mémoires nécessaires importantes[(6)]. D'un autre coté, une fenêtre de petite taille peut pénaliser les performances de décodage de l'accumulateur

[(6)]La seule exception est le nombre de valeurs d'initialisations qui diminue avec le nombre de fenêtres de décodage de l'accumulateur

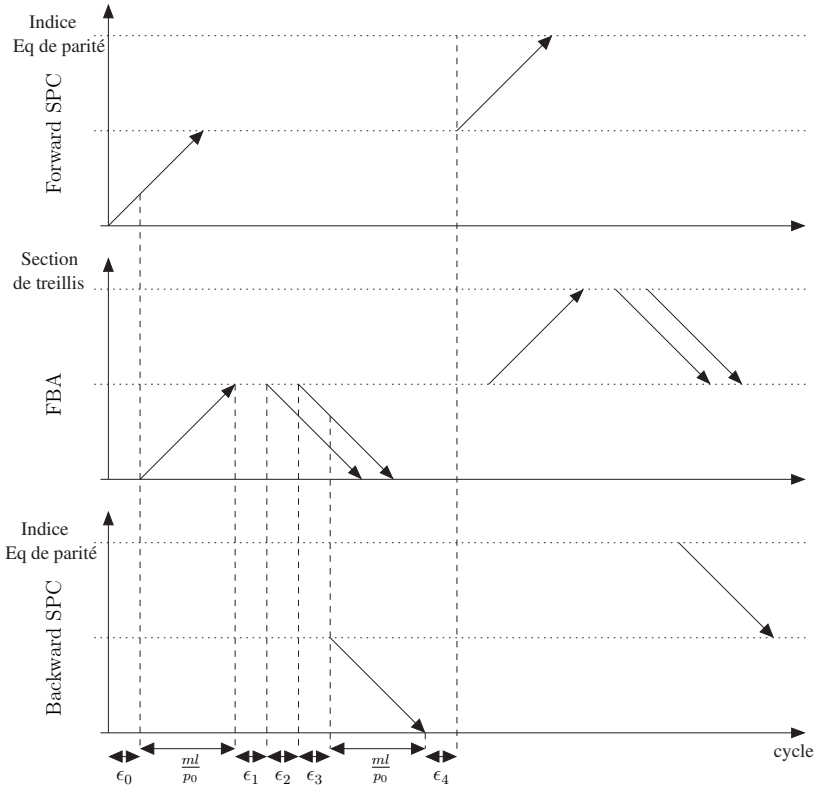


FIG. 3.11 – Séquencement série de l'algorithme *Turbo Layered BP*, où les différentes étapes du décodage sont représentées.

(il est généralement préconisé une taille de fenêtre minimum égale à 5-6 fois la longueur de contrainte du code pour code convolutif systématique de rendement 1/2). Comme nous l'avons mentionné dans le chapitre précèdent, le choix du paramètre m influe également sur la borne de la distance minimale du code.

Pour définir le fonctionnement de chaque processeur et calculer le débit de décodage, nous proposons de raisonner sur le processeur FBA. On considère que ce processeur est capable de traiter p_0 sections de treillis en un cycle, avec un ordonnancement comme celui illustré sur la figure 3.11. Le processeur FBA étant un processeur traitant des données en série, la résolution de p_0 équations de parité par le processeur *Forward SPC* doit se faire en un cycle. Ce comportement suggère donc l'utilisation d'un processeur *Forward SPC* avec une architecture parallèle capable de traiter p_0 équations de parité en un cycle. Cette solution a pour conséquence une réduction de la flexibilité du décodeur. Ce point sera abordé par la suite. A partir de la représentation de la figure 3.11, le débit utile de décodage peut être déduit :

$$D = \frac{RN}{(\epsilon + 2\frac{lm}{p_0})\frac{z}{l}it} f_{clk} \tag{3.17}$$

où ϵ représente la somme des latences du décodage de la fenêtre, $\epsilon = \sum_i \epsilon_i$. On rappelle que le paramètre ϵ est un paramètre global intégrant l'ensemble des cycles nécessaires pour la lecture, le traitement local et l'écriture d'une donnée. Les paramètres ϵ_i sont illustrés et placés dans le chronogramme à titre indicatif et ne reflètent pas localement le comportement de l'architecture.

Cette expression du débit suggère quelques remarques sur le choix de la taille de la fenêtre, et plus globalement, des paramètres du code. Tout d'abord, le paramètre ϵ pénalise lourdement le débit du décodeur et notamment quand le paramètre $2lm/p_0$ est petit. Une des solutions pour s'affranchir de cet effet consiste à décoder des fenêtres de grandes tailles, ce qui en contrepartie contraint très fortement le code. Une deuxième solution est l'utilisation d'un décodeur pipeliné. Cette solution peut permettre de s'affranchir du paramètre ϵ tout en maximisant l'activité de chaque processeur de calcul. Ce séquencement et les conséquences sur la détermination du code sont discutées dans la section suivante.

3.3.1.2 Séquencement série pipeliné

On considère le séquencement pipeliné dérivé du séquencement série dont la synoptique est illustrée sur la figure 3.12. On rappelle que le but de ce séquencement et de s'affranchir du terme ϵ et de maximiser l'activité des processeurs de calculs. Ce séquencement est illustré sur la figure 3.13. Le débit utile peut s'exprimer à l'aide de la représentation de la figure 3.13 par :

$$D = \frac{RN}{\left(\epsilon + \frac{ml}{p_0} + \frac{m}{p_0} z\right) it} f_{clk} \tag{3.18}$$

Dans le cas où le pipeline est aussi possible entre deux itérations, l'expression du débit est :

$$D = \frac{RN}{\epsilon + \frac{ml}{p_0} + \left(\frac{m}{p_0} z\right) it} f_{clk} \tag{3.19}$$

Plus le paramètre l sera petit, plus le débit sera important. Il faut cependant remarquer que ce paramètre a très peu d'influence si la taille du code et le nombre

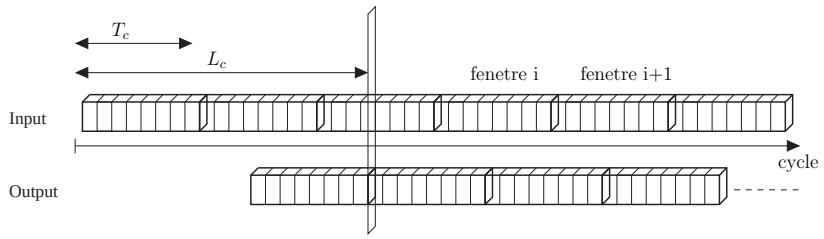


FIG. 3.12 – Illustration du séquencement série pipeliné. Le décodage des fenêtres se fait sans interruption.

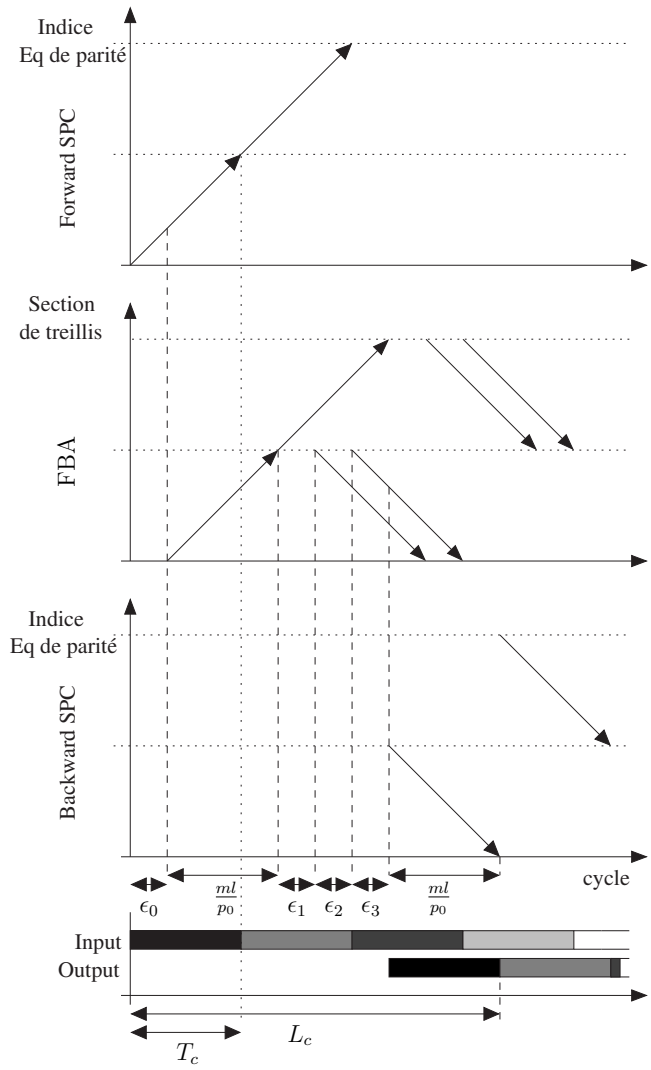


FIG. 3.13 – Illustration du séquencement série pipeliné.

d'itérations sont grands.

L'utilisation d'un tel séquencement impose des règles de construction sur le code afin de garantir, les propriétés de l'algorithme *Turbo Layered BP* et, d'éviter les problèmes d'accès simultanés à un même espace mémoire. Un noeud de données intervenant dans une équation de parité dont la mise à jour n'a pas été réalisée ne doit pas intervenir dans une équation en cours de décodage. Plus particulièrement, si un noeud de données intervient dans une équation de parité de la fenêtre k, il ne doit pas intervenir dans les $k-j,\ j=0\ldots\lceil L_c/T_c\rceil-1$ fenêtres précédentes et les $k+j,\ j=0\ldots\lceil L_c/T_c\rceil-1$ fenêtres suivantes. Cette contrainte appliquée aux coefficients de permutation définissant la nature des matrices identité permutées dans une

colonne de $\mathbf{H_s}$ permet de déduire la règle suivante :

$$\begin{array}{c} \delta_i \neq (\delta_j \pm k'l \pm k) \bmod z,\ i \neq j \\ k' = 0 \ldots \left\lceil \frac{L_c}{T_c} \right\rceil - 1,\ k = 0 \ldots l-1 \end{array} \tag{3.20}$$

Dans le cas du séquencement illustré sur la figure 3.13, le paramètre $\lceil L_c/T_c \rceil$ peut s'exprimer par :

$$\left\lceil \frac{L_c}{T_c} \right\rceil = 2 + \left\lceil \frac{p_0 \epsilon}{ml} \right\rceil \tag{3.21}$$

Il faut noter que les propriétés de l'opérateur modulo permettent que le pipeline se réalise entre deux itérations si la règle énoncée ci dessus est bien respectée.

L'utilisation d'un tel ordonnancement est accompagné d'une légère augmentation de la complexité. La mémoire nécessaire pour les buffers et plus particulièrement pour celui mémorisant les informations m_{vc} doit permettre le stockage d'un nombre plus important de données. Dans le cas d'une fenêtre de taille lm, et d'une latence de décodage de L_c cycles, ce buffer doit être de profondeur au moins égale à $lm(\lceil L_c/T_c \rceil - 1)$.

En résumé, cet ordonnancement sera d'autant plus intéressant que le paramètre ϵ, directement lié aux paramètres de l'architecture et de la technologie visée (latence de lecture/écriture dans une mémoire...) est important. Ce séquencement autorise donc l'utilisation de petites fenêtres de décodage. Dans ce cas, la profondeur du buffer sera réduite et le débit amélioré. Cependant dans le cas de codes de petites tailles avec des degrés de connexion des noeuds de données élevés, la contrainte peut rendre la construction du code très difficile. Ce séquencement sera donc adapté dans le cas où le paramètre m sera petit devant le paramètre z. L'augmentation du débit de décodage peut aussi se réaliser en augmentant le nombre de fenêtres décodées simultanément. Ce séquencement parallèle est illustré dans la section suivante.

3.3.1.3 Séquencement parallèle

Soit le séquencement illustré sur la figure 3.14 caractérisant le décodage de p fenêtres simultanées. Cet ordonnancement permet de multiplier le débit de décodage de l'ordonnancement série par un facteur p :

$$D = p \frac{RN}{(\epsilon + 2\frac{lm}{p_0})\frac{z}{l} it} f_{clk} \tag{3.22}$$

Le fait de décoder simultanément plusieurs fenêtres implique une architecture avec une complexité plus importante. En effet, l'ensemble des processeurs et buffers doit être dupliqué p fois. Il faut aussi noter que le réseau de permutations doit être dimensionné de manière à permettre le transfert de $p_0 p$ données entre les organes de mémorisation et les processeurs de calcul. L'introduction de ce nouveau niveau de parallélisme nécessite la définition de nouvelles règles de construction de codes.

La nouvelle contrainte sur l'activation des noeuds de données est la suivante : un noeud de données ne doit intervenir qu'une seule fois dans le décodage de p fenêtres de taille lm. Avant de définir les règles sur les coefficients de permutations, quelques remarques doivent être illustrées. Tout d'abord, pour que l'effet de ce niveau de parallélisme sur les performances soit négligeable, nous considérons la division du treillis de l'accumulateur en p sections de tailles égales. Cette condition impose la résolution des équations de parité dans un ordre particulier. En utilisant la relation 2.14 et le même raisonnement que dans les cas précédent, on montre que la règle à imposer sur les coefficients de permutation définissant les matrices identité sur une colonne de $\mathbf{H_s}$ est :

$$\begin{gathered}\left(\delta_i + k\frac{z}{p}\right) \bmod z \neq \left(\delta_j + k'\frac{z}{p} \pm k''\right) \bmod z, \\ \forall i \neq j,\ \forall k = 0 \ldots p-1,\ \forall k' = 0 \ldots p-1,\ \forall k'' = 0 \ldots l-1\end{gathered} \tag{3.23}$$

Cette règle de construction introduit des contraintes très fortes sur le code notamment quand le niveau de parallélisme et le degré de connexion des noeuds sont grands. Comme l'ordonnancement série, le paramètre ϵ a un effet très important sur le débit et l'activité de chaque processeur. Une manière de s'affranchir de cet effet est de considérer un séquencement parallèle pipeliné. Celui ci est illustré dans la section suivante.

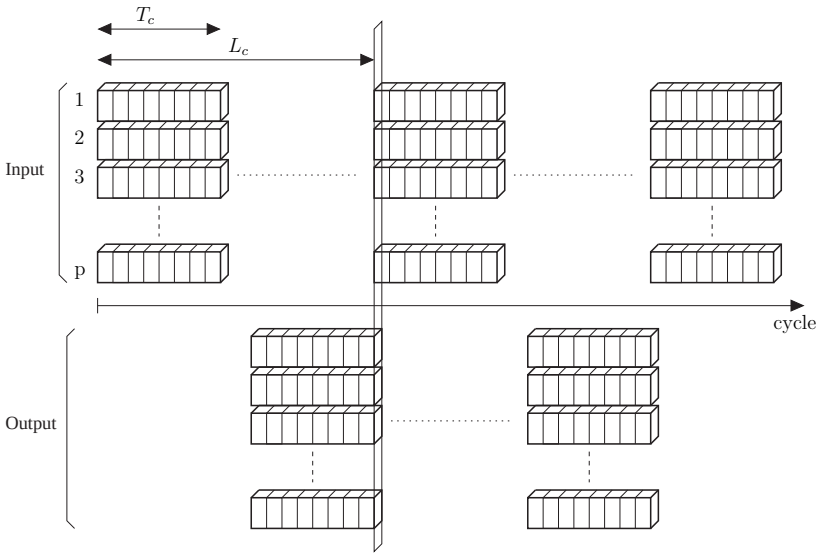


FIG. 3.14 – Illustration du séquencement parallèle. Le décodage de p fenêtres se fait simultanément. Un nouveau groupe de fenêtres est décodé quand le décodage du groupe précédent est terminé.

3.3.1.4 Séquencement parallèle pipeliné

L'ordonnancement parallèle pipeliné consiste à considérer le décodage de p fenêtres en parallèle et cela d'une manière continue. Le synoptique de ce séquencement est illustré sur la figure 3.15. Le décodage de manière continue des fenêtres permet de

multiplier par p le débit du séquencement série pipeliné. La nouvelle expression du débit de décodage est donc :

$$D = p\frac{RN}{\epsilon + \frac{ml}{p_0} + \left(\frac{m}{p_0}z\right)it} f_{clk} \tag{3.24}$$

Comme nous l'avons déjà mentionné précédemment, plus la taille de la fenêtre sera petite, meilleur sera le débit de l'architecture. On peut aussi noter que dans ce séquencement, les processeurs de calcul sont utilisés d'une manière quasi-optimale. Cette augmentation du débit s'accompagne néanmoins de contraintes très fortes sur le code. Ainsi un noeud de données ne doit intervenir qu'une seule fois dans les p fenêtres décodées en parallèle ainsi que dans les $\lceil L_c/T_c \rceil - 1$ groupes de p fenêtres suivant et précédent. Cette règle transposée aux q coefficients de permutation définissant la permutation des matrices identité permutées dans une colonne de $\mathbf{H_s}$ peut s'écrire :

$$\left(\delta_i + k\frac{z}{p}\right) \bmod z \neq \left(\delta_j + k'\frac{z}{p} \pm k'' \pm k'''l\right) \bmod z,$$
$$\forall i \neq j,\ \forall k = 0 \ldots p-1,\ \forall k' = 0 \ldots p-1,\ \forall k'' = 0 \ldots l-1,\ \forall k''' = 0 \ldots \left\lceil \frac{L_c}{T_c} \right\rceil - 1 \tag{3.25}$$

Il faut noter que la complexité de l'architecture est augmentée par le traitement simultané de groupes de p données et, par la duplication par un facteur p des processeurs de calcul. Cette solution sera donc intéressante quand un débit élevé est requis et que le paramètre ϵ est grand. Le taille du code devra quant à elle être suffisamment grande pour permettre sa construction suivant les règles décrites.

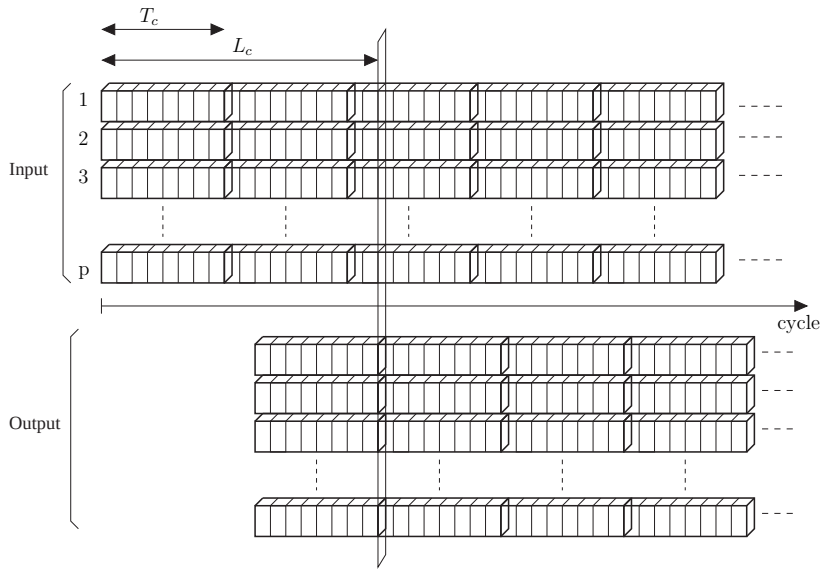


FIG. 3.15 – Illustration du séquencement parallèle pipeliné.

3.3.1.5 Conclusions

Dans cette section, nous avons exploré différentes stratégies de décodage. A chaque ordonnancement est associé une estimation du débit, des paramètres relatifs à l'architecture, et les contraintes posées sur le code. Une interprétation graphique de ces contraintes est illustrée dans l'annexe A.

Comme illustré sur la figure 3.16, plus le débit sera important plus les contraintes sur le code seront fortes. Quand le paramètre ϵ est de l'ordre de m, le meilleur compromis entre complexité, contraintes sur le code et débit, sont les architectures autorisant le pipeline des opérations. Le choix de l'ordonnancement devra cependant être réalisé une fois que toutes les données relatives à l'application et à la cible matérielle sont connues.

On peut aussi noter que les contraintes sur le code issues des considérations d'architecture, sont redondantes avec celles décrites dans l'analyse des distances de Hamming. Ainsi un code ayant une bonne distance w_1 aura une forte probabilité de permettre l'un des séquencements étudiés.

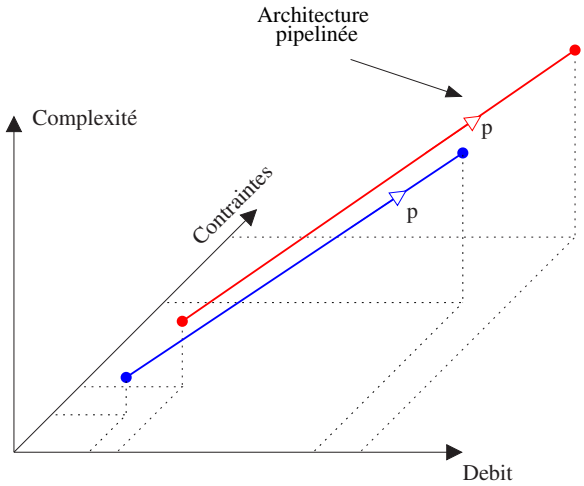


FIG. 3.16 – Illustration dans un repère Débit/Complexité/Contraintes des architectures étudiées en fonction du parallélisme p.

3.3.2 Mise en oeuvre de la généricité

L'étude des architectures associées à l'algorithme *Turbo Layered BP* a mis en évidence que pour obtenir une activité optimale du processeur FBA, les équations de parité décrites par la matrice $\mathbf{H_s}$ devaient être traitées en un cycle. Cette contrainte impose donc l'utilisation de processeurs traitant en parallèle tous les éléments d'une équation de parité. Cette stratégie est peu propice à une réalisation flexible quand

le nombre d'éléments par équation de parité varie avec le rendement ou pour un ensemble de codes. Dans le premier chapitre, nous avons décrit des solutions possibles qui consistent à considérer le pire cas. Cette solution est très complexe à mettre à oeuvre notamment quand les processeurs sont dupliqués ou quand le nombre d'éléments à traiter est grand.

Le but de cette section est d'illustrer une nouvelle méthode pour permettre, dans le cas de l'algorithme étudié associé à un code dont la matrice de contrôle de parité est de forme bi-diagonale, une plus grande flexibilité. Dans un premier temps nous introduirons le concept de matrice équivalente. L'application de cette méthode sera dans un deuxième temps illustrée dans le cas de notre problématique. Dans une dernière partie, des nouvelles contraintes sur la determination des codes seront discutées.

3.3.2.1 Notion de matrice de contrôle de parité équivalente

Soit une matrice de contrôle de parité $\mathbf{H}$ divisible en deux sous matrices $\mathbf{H_s}$ et $\mathbf{H_p}$ où la matrice $\mathbf{H_p}$ est une matrice bi-diagonale de taille $M \times M$. Les équations de parité décrites par la matrice de contrôle de parité sont de la forme :

$$\sum_{i=0}^{nJ-1} x(\mathbb{P}(i,k)) + p_{k-1} = p_k \tag{3.26}$$

où l'on considère que le nombre d'éléments non nul par ligne de la matrice $\mathbf{H}$ est égal $nJ+2$ (on verra par la suite que ce nombre peut varier). $\mathbb{P}(i,k)$ correspond à la position du i^{eme} élément non nul de la ligne k de $\mathbf{H}$. Pour alléger les équations, l'opérateur modulo a volontairement été oublié. Cette équation de parité peut être ré-écrite en faisant intervenir des sommes de J bits de la manière suivante :

$$\sum_{j=0}^{n-1} \sum_{i=jn}^{(j+1)J-1} x(\mathbb{P}(i,k)) + p_{k-1} = p_k \tag{3.27}$$

Cette relation peut de nouveau être exprimée de la manière suivante :

$$\begin{array}{rcccl} e_k^0 & = & p_{k-1} & + & \displaystyle\sum_{i=0}^{J-1} x(\mathbb{P}(i,k)) \\ e_k^1 & = & e_k^0 & + & \displaystyle\sum_{i=J}^{2J-1} x(\mathbb{P}(i,k)) \\ & & \vdots & & \\ p_k & = & e_k^{n-2} & + & \displaystyle\sum_{i=(n-1)J}^{nJ-1} x(\mathbb{P}(i,k)) \end{array} \tag{3.28}$$

où les éléments e_k^i sont des variables intermédiaires. Ces relations font apparaître une relation d'accumulation entre les variables e_j^i et p_j. Chaque équation de parité peut

être divisée en n. On peut donc définir une nouvelle matrice de contrôle de parité équivalente $\mathbf{H}_{eq}$ de taille $nM \times (K + nM)$ à laquelle il est associé un mot de code équivalent $\underline{x}_{eq}$ de dimension $K + nM$.

□ *Exemple:* Soit la matrice de contrôle de parité ligne suivante :

$$\mathbf{H} = [1\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 1]$$

à laquelle on associe le mot de code $\underline{x}$:

$$\underline{x} = [c_0\ c_1\ c_2\ c_3\ c_4\ c_5\ c_6\ c_7\ c_8\ c_9\ c_{10}\ p_0\ p_1]$$

L'équation de parité décrivant le code peut s'écrire :

$$p_1 = p_0 + c_0 + c_1 + c_3 + c_5 + c_8 + c_{10}$$

Si l'on considère le paramètre $J = 2$ alors cette équation peut s'exprimer par :

$$p_1 = p_0 + (c_0 + c_1) + (c_3 + c_5) + (c_8 + c_{10})$$

ou encore, en faisant intervenir des variables intermédiaires :

$$\begin{aligned} e_0^0 &= p_0 + (c_0 + c_1) \\ e_1^0 &= e_0^0 + (c_3 + c_5) \\ p_1 &= e_1^0 + (c_8 + c_{10}) \end{aligned}$$

On peut alors définir la matrice de contrôle de parité équivalente $\mathbf{H}_{eq}$ par :

$$\mathbf{H}_{eq} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}$$

à laquelle il est associé le mot de code équivalent :

$$\underline{x}_{eq} = \left[c_0\ c_1\ c_2\ c_3\ c_4\ c_5\ c_6\ c_7\ c_8\ c_9\ c_{10}\ p_0\ e_0^0\ e_1^0\ p_1\right]$$ □

L'introduction de valeurs intermédiaires peut se réaliser de manière à faire apparaître une matrice équivalente de type bi-diagonale. Les variables intermédiaires peuvent être vues comme des noeuds de données cachés ou encore comme des noeuds de données associés à des bits poinçonnés. Il faut noter que le décodage de la matrice de contrôle de parité équivalente apporte les mêmes performances que le décodage de la matrice de contrôle de parité mère quand le nombre d'itérations est suffisamment grand. C'est cette propriété qui va être utilisée pour rendre flexible l'architecture proposée.

3.3.2.2 Matrice de contrôle de parité équivalente et flexibilité du décodeur

Nous proposons dans cette section d'illustrer comment l'utilisation de la matrice de contrôle de parité équivalente peut permettre de rendre générique le décodeur. On considère dans un premier temps une matrice de contrôle de parité de forme bi-diagonale ayant $nJ + 2$ éléments non nuls par ligne, caractérisant un code de rendement R. Une généralisation sera par la suite illustrée. Par hypothèse on dispose d'un décodeur *Turbo Layered BP* dont les processeurs SPC sont capables de traiter un noeud de contrôle à J entrées et une sortie. Une augmentation du rendement de codage s'accompagne généralement d'une augmentation du taux de connexion des noeuds de contrôle. Le concept de matrice équivalente permet d'adresser cette contrainte de flexibilité.

En utilisant le concept de matrice équivalente, on peut transformer la matrice de contrôle de parité en une matrice de contrôle de parité de forme bi-diagonale ayant $J+2$ éléments non nuls par ligne. Le rendement de codage du code équivalent associé à la matrice de contrôle de parité équivalente est donc de :

$$R_{eq} = \frac{R}{n(1-R)} \tag{3.29}$$

□ *Exemple:* Soit la matrice de contrôle de parité suivante définissant un code de rendement $R = 2/3$ et de taille $N = 12$:

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

et le mot de code correspondant suivant :

$$\underline{x} = \left[\begin{array}{cccccccc|cccc} c_0 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & p_0 & p_1 & p_2 & p_3 \end{array}\right]$$

Soit un décodeur *Turbo Layered BP* dont les processeurs SPC sont capables de traiter un noeud de contrôle à 2 entrées et une sortie. Pour permettre à ce décodeur de fonctionner, la matrice doit être traitée de manière à ne faire apparaître que 2 éléments non nuls par ligne de la partie non bi-diagonale. Une matrice équivalente respectant cette règle peut s'écrire de la manière suivante :

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

Le mot de code équivalent est alors :

$$\underline{x}_{eq} = \left[\begin{array}{cccccccc|cccccccc} c_0 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & \mathbf{e_0} & p_0 & \mathbf{e_1} & p_1 & \mathbf{e_2} & p_2 & \mathbf{e_3} & p_3 \end{array}\right]$$

Le rendement du code équivalent est égal à 1/2. □

Le décodage par l'algorithme et l'architecture proposés ne rajoute pas de complexité au décodeur. En effet l'insertion de variables intermédiaires dans la matrice équivalente se traduit par une association de valeurs neutres à certaines entrées du décodeur FBA. Autrement dit, le décodeur voit le code comme un code poinçonné dont il connaît la position des éléments qui n'ont pas été transmis. Si la fenêtre de décodage de l'accumulateur a été bien dimensionnée, alors le décodage se fait sans augmentation de complexité. La particularité de ce décodeur, comparé à la majorité des décodeurs décrits dans la littérature, est son débit de fonctionnement. Celui-ci devient invariant en fonction du rendement du code et est fixé par le rendement de la matrice équivalente. Par cette technique, on se rapproche du comportement d'un décodeur de Turbo-codes.

Ce concept de matrice équivalente peut aussi être applicable à un décodage de type BP où *Layered BP*. Le principe serait de construire une matrice équivalente dont le nombre d'éléments non nuls par lignes est en adéquation avec ce que peut traiter un processeur CNP. Cependant, le fait de décoder des noeuds cachés dans le graphe augmente sensiblement les calculs et les ressources mémoires nécessaires. L'introduction de noeuds cachés se traduit par une augmentation du nombre de branches dans le graphe, ce qui entraîne une augmentation non négligeable des ressources mémoires nécessaires dans le cas d'un algorithme classique. Dans notre cas, l'utilisation d'une telle technique est rendue possible du fait du décodage de la partie bi-diagonale de la matrice comme un treillis associé avec un décodage par fenêtres[(7)]. Contrairement à l'algorithme BP et *Layered BP*, il n'est pas nécessaire de mémoriser des messages de branches supplémentaires. Toute la flexibilité requise par le décodeur est alors reportée sur la modification du treillis de l'accumulateur. Nous avons illustré précédemment la détermination de matrices équivalentes dans un cas où la matrice $\mathbf{H_p}$ est strictement bi-diagonale. Un exemple d'une structure où la matrice $\mathbf{H_p}$ n'est pas strictement bi-diagonale est illustré annexe B.

Pour permettre à un décodeur de traiter une large gamme de rendement de codage avec un processeur donné, il est nécessaire de fixer des contraintes sur les codes. Ces contraintes sont décrites dans la section suivante.

3.3.2.3 Contraintes sur la construction du code

Pour permettre une réalisation simple, ainsi qu'une caractérisation simplifiée de l'insertion des valeurs neutres dans le treillis, des règles de construction doivent être

[(7)] Le décodage d'un treillis plus grand n'augmente pas la complexité du décodeur pour une taille de fenêtre constante.

mises en oeuvre.

Tout d'abord pour que l'insertion des valeurs neutres se fassent d'une manière la plus régulière possible, il est souhaitable que le nombre d'éléments non nuls par ligne de la matrice soit constant. Cette contrainte de concentration de degré de connexion des noeuds de contrôle est en adéquation avec les règles de construction de codes énoncées au chapitre 2.

Une seconde règle de construction concerne directement le choix du degré de connexion des noeuds de contrôle. Ce choix doit être un compromis entre les différents paramètres relatifs à l'architecture, au débit de décodage et aux performances du code. Le degré de connexion des noeuds de contrôle est de la forme $nJ + 2$ où J est le nombre d'entrées du processeur associé à un noeud de contrôle. Plus le paramètre J sera grand, plus la complexité du processeur sera importante, celui-ci devant traiter J entrées en un cycle. Le débit de décodage dépend aussi de ce paramètre. Ainsi plus J sera grand plus le débit sera élevé. En considérant les performances du code et le rendement de codage, le taux de connexions des noeuds de contrôle peut être optimisé. A titre d'exemple, la figure 3.17 illustre le degré de connexion moyen des noeuds de contrôle pour des ensembles de codes de rendement de codage différents. Cette figure montre qu'il existe un important degré de liberté dans le choix du degré de connexion moyen des noeuds de contrôle. Il faut aussi noter que pour un même rendement de codage, plus le taux moyen de connexion des noeuds de contrôle est important, plus la densité de la matrice de contrôle de parité sera importante. Cette propriété implique un nombre de calculs plus important pour le codage et le décodage du code.

Il est clair qu'il n'existe pas un paramètre optimal, capable à la fois de garantir une bonne granularité des degrés de connexion, un bon débit et de bonnes performances quelque soit la taille et le rendement du code. Dans le cadre de nos études, deux stratégies différentes nous semblent pertinentes. Dans un cas où un système nécessite une très large gamme de rendements de codage, nous suggérons de fixer le paramètre J à 2. La granularité offerte dans le degré de connexion des noeuds de contrôle est alors de $2n + 2$. Le processeur de résolution des équations de parité est alors très peu complexe, mais le débit de décodage devient aussi assez faible. Le cas où J est fixé à 4 est une deuxième solution intéressante. La granularité offerte est de $4n + 4$ ce qui permet de déterminer de bonnes distributions pour une gamme de rendement à partir d'un rendement de codage de 1/2 (cf figure 3.17). Dans ce cas, la relation entre rendement de codage distribution est la suivante :

$$\rho(x) = x^{4n+2} \rightarrow R = \frac{n}{n+1} \tag{3.30}$$

Cette solution offre une bonne distribution pour les rendements considérés tout en offrant un bon compromis sur la complexité et le débit de décodage. Ces différents points seront abordés dans le dernier chapitre. Le choix d'une telle relation entre rendement de codage et distribution des degrés semble limiter la granularité en terme de

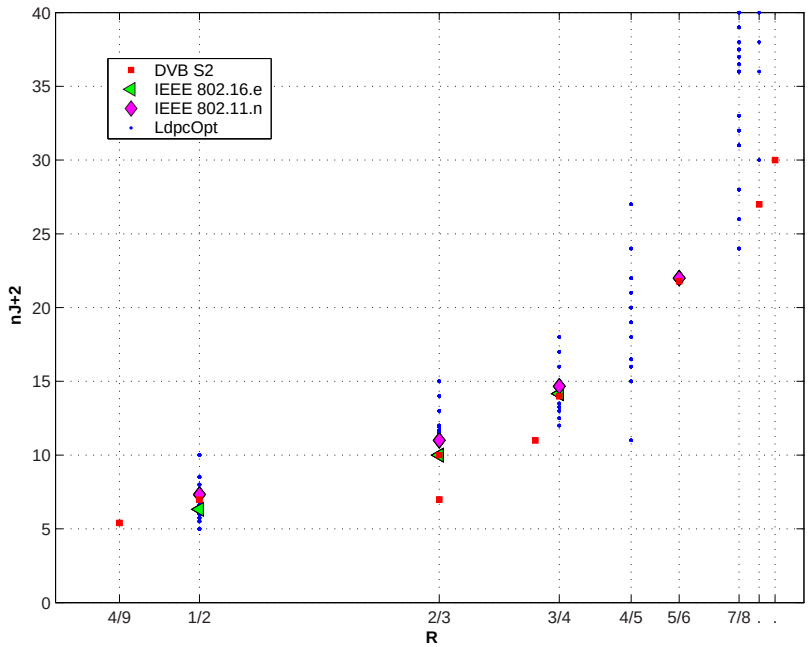


FIG. 3.17 – Variation du degré de connexion moyen des noeuds de contrôle en fonction du rendement de codage. Les moyennes ont été calculées à partir des codes LDPC normalisés dans le cadre des standards DVB-S2, IEEE 802.16e, IEEE 802.11n et de la base de données de codes LDPC optimisés (sous la contrainte d'un degré de connexion des noeuds de données maximum de 20, et une part de noeuds de données de degré 2 compatible avec la forme bi-diagonale) de l'EPFL [98]. Cet ensemble de code est noté LdpcOpt.

rendements de codage de notre décodeur. Cette technique et ce choix de paramètres permettent cependant de couvrir une grande partie des rendements considérés communément dans les standards, $R = 1/2,\ 2/3,\ 3/4,\ 4/5,\ 5/6,\ 6/7,\ 7/8$.... Comme il a été démontré dans [86, 58] il est aussi possible de définir des bonnes séquences de poinçonnage pour cette famille de codes. Ces techniques utilisées en complément de la solution proposée permettent d'obtenir un schéma de codage avec une très large gamme de rendements. Ce choix sur la valeur du paramètre J revient à discuter du choix du niveau de parallélisme du processeur de résolution de parité. Comme nous l'avions déjà mentionné, plus le parallélisme sera élevé, plus le débit de décodage sera important et moins le décodeur sera flexible.

3.3.2.4 Remarques et conclusions

Le problème de la flexibilité des décodeurs de codes LDPC a été adressé dans cette section. L'idée principale de la solution proposée est de transférer les contraintes liées à la variation du degré de connexion des noeuds de contrôle au treillis de l'accumulateur. Du fait du décodage par fenêtre et de la simplicité de mise en oeuvre du décodage d'un treillis de longueur variable, la solution proposée est efficace et peu

complexe. Comme nous l'avons mentionné, cette solution n'est efficace que dans le cas de l'algorithme de décodage proposé. Cette technique directement appliquée à un algorithme de type BP entraînerait une augmentation très importante des ressources de mémorisation nécessaires.

L'application au niveau du décodeur de cette technique, qui implique un dépoinçonnage implicite, ouvre de nouvelles applications à ce type de codes. En effet, il est souvent reproché aux codes LDPC de manquer de flexibilité en terme de rendements de codage. D'une manière générale cette constatation est vraie, notamment pour des sytèmes de type *Hybrid Automatic Repeat Request* (H-ARQ) utilisant la technique d'*Incremental redundancy*. Les techniques d'H-ARQ combinent l'efficacité et la robustesse des codes correcteurs d'erreurs avec les techniques ARQ afin d'optimiser les débits. Les techniques H-ARQ de type II, également connues sous l'appellation *Incremental Redundancy* (IR) sont particulièrement intéressantes de ce point de vue et sont employées avec succès dans des systèmes tel que HSDPA (High Speed Downlink Packet Access) [99]. Elles consistent à envoyer lors d'une retransmission, des informations de parité supplémentaires, c'est-à-dire qui n'avaient pas été envoyées lors des transmissions précédentes. Pour ces différentes raisons, il est préférable d'utiliser des codes correcteurs d'erreurs dit *Rate-Compatible* (RC), qui sont des codes correspondant à plusieurs rendements différents, obtenus à partir d'un même code d'origine. Ils peuvent par exemple être construits par poinçonnage d'un code de faible rendement. Cette technique est classiquement utilisée dans le cas des Turbo-codes parallèles mais est plus délicate dans le cas des codes LDPC. En effet, les motifs de poinçonnage doivent être appliqués avec précautions aux codes LDPC de manière à éviter des pertes d'information trop importantes qui pourraient empêcher le décodage de converger vers le mot de code. De nombreuses études théoriques ont été présentées dans le cas des codes LDPC, comme dans [100, 101]. L'inconvénient majeur de ces techniques est l'absence de mise en oeuvre simple d'un décodeur générique.

L'utilisation de la famille de codes proposée, associée au concept très simple de matrice équivalente, permet de répondre à ces problématiques. Si le codeur a connaissance de la matrice équivalente, il peut coder l'information et n'envoyer au travers du canal qu'une partie de la redondance. Le décodeur traitant la matrice équivalente insère simplement les éléments neutres dans les sections de treillis appropriées. La retransmission de parité est réalisée en remplaçant certaines valeurs neutres par l'observation du canal de la redondance supplémentaire. Cet aspect n'a pas été exploré au cours de la thèse ni la manière de construire de bons codes répondant à cette problématique. Cependant, l'architecture proposée associée à l'algorithme de décodage présenté et à la construction de code suggérée permet en pratique l'utilisation de ce type de codes dans des systèmes H-ARQ de type II. L'ensemble des résultats illustrés dans cette section peut apporter des solutions à ces problèmes imputés aux codes LDPC.

3.4 Étude d'une architecture hybride

Nous avons mentionné dans le chapitre précédent que l'algorithme de propagation de croyance est exact dans le cas d'un code dont le graphe ne contient pas de cycles. Les auteurs de [89] et [55] ont suggéré qu'un bon ordonnancement peut être défini en isolant des parties suffisamment grandes du graphe qui forment des chaînes et, en les décodant en utilisant un algorithme de propagation de croyance sur un treillis. Nous avons vu que les codes LDPC construits à partir d'une matrice bi-diagonale permettent très simplement l'application de ce concept. L'extension de ce principe à la partie non bi-diagonale de la matrice de contrôle de parité peut être explorée.

Cette méthode consiste à extraire de la partie non bi-diagonale de la matrice un cycle ou une chaîne de longueur suffisamment importante, et d'utiliser cette propriété au décodage. En particulier, une solution intéressante consiste à décomposer la matrice $\mathbf{H_s}$ en un produit de matrices de type bi-diagonale de la manière suivante :

$$\mathbf{H_s} = \mathbf{V} \prod_{i=0}^{n-1} \mathbf{G_1} \mathbf{\Pi}_i \tag{3.31}$$

où $\mathbf{\Pi}_i$ est une matrice d'entrelacement de taille $K \times K$, $\mathbf{G_1}$ une matrice de taille $K \times K$ bi-diagonale, et $\mathbf{V}$ une matrice de faible densité de taille $M \times K$. Cette décomposition revient à considérer le code global comme une concaténation série de $n+2$ codes, dont au moins $n+1$ sont définis par un treillis à deux états.

Cette stratégie de décomposition n'est pas simple à mettre en oeuvre. Dans une première partie de cette section, nous verrons comment mettre en oeuvre ce principe à travers la définition de famille de codes particulières. Dans un second temps, une rapide présentation d'architecture pour ce type de stratégie sera illustrée.

3.4.1 Structure des codes

La décomposition de la sous matrice $\mathbf{H_s}$ en un produit de matrices bi-diagonales n'est pas très simple à mettre en oeuvre. Nous ne chercherons donc pas à trouver la décomposition à partir de la matrice $\mathbf{H_s}$, mais nous proposons de considérer une famille de codes où l'on concatene en série plusieurs treillis à peu d'états qui pourront se décoder simplement. Parmi les familles de codes ayant ces propriétés on trouve les codes de type *Acumulate Repeat Accumulate* (ARA) codes introduits par Divsalar *et al.* dans [59] où encore les codes S-SCP (Systematic with Serially Concatenated Parity) [60]. La structure de ces codes est présentée sur la figure 3.18. Les codes S-SCP tels que définis dans [60] sont une concaténation d'un code externe et d'un code *Inner Parity Generator*(IPG) comme illustré sur la figure 3.18(a). En particulier il a été démontré l'intérêt de la structure quand le code est construit à partir d'une concaténation série d'un code convolutif externe, d'un ensemble d'équations de parité (SPC) et d'un code convolutif interne [60] (cf figure 3.18(b)). La solution particulièrement intéressante est

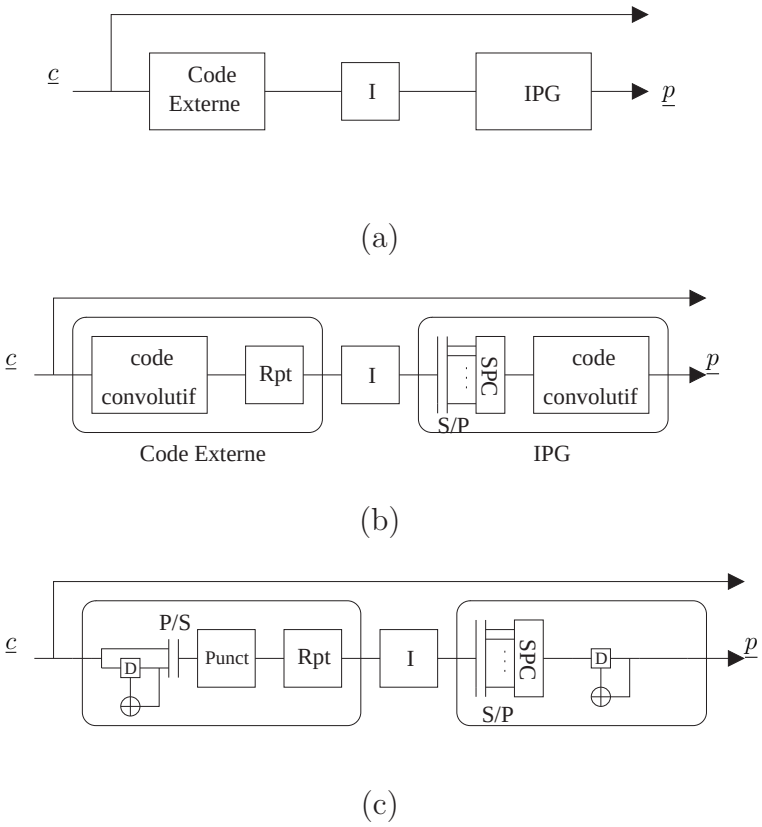


FIG. 3.18 – Illustration des codes de type S-SCP ((a) et (b)) et ARA (c). I correspond à une fonction d'entrelacement, Rpt a une répétition et Punct a une fonction de poinçonnage.

celle où les codes convolutifs sont des codes dont le treillis est à deux états. La matrice de contrôle de parité de ces codes peut alors s'exprimer de la manière suivante :

$$\mathbf{H} = \begin{bmatrix} \mathbf{G_1} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{V} & \mathbf{H_p} \end{bmatrix} \tag{3.32}$$

où $\mathbf{H_p}$ est une matrice de taille $M \times M$ bi-diagonale, $\mathbf{V}$ une matrice[(8)] de taille $M \times K$ de faible densité et $\mathbf{G_1}$ une matrice de taille $K \times K$ caractérisant le code externe. Cette matrice définissant le code de précodage est bi-diagonale quand le code externe est un code convolutif de fonction de transfert $1 + D$. Ces codes, étudiés dans [60] et équivalents à des codes ARA (cf figure 3.18(c)), ne sont certes pas les meilleurs codes d'un point de vue performances théoriques mais semblent bénéficier d'un compromis performance/flexibilité/complexité très avantageux [60].

Les codes S-SCP tels que définis dans [60] peuvent aussi s'interpréter comme des codes de type *Repeat Accumulate*. En effet, si l'on considère les équations d'encodage

(8) l'expression mathématique de cette matrice inclut la fonction d'entrelacement

suivante :

$$\mathbf{H}\underline{x}^t = \underline{0}^t$$
$$\begin{bmatrix} \mathbf{G_1} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{V} & \mathbf{H_p} \end{bmatrix} \begin{bmatrix} \underline{c}^t \\ \underline{h}^t \\ \underline{p}^t \end{bmatrix} = \underline{0}^t \tag{3.33}$$

où $\underline{h}^t$ est un vecteur intermédiaire, on peut déduire le système d'équation suivante :

$$\begin{cases} \mathbf{G_1}\underline{c}^t = \underline{h}^t \\ \mathbf{V}\underline{h}^t = \mathbf{H_p}\underline{p}^t \end{cases} \tag{3.34}$$

qui peut aussi s'écrire de la manière suivante :

$$\mathbf{VG_1}\underline{c}^t = \mathbf{H_p}\underline{p}^t \tag{3.35}$$

Cette équation d'encodage est similaire à celle d'un code de type *Repeat Accumulate* dont la matrice de contrôle de parité est :

$$\mathbf{H} = [\mathbf{VG_1}\ \mathbf{H_p}] \tag{3.36}$$

Les codes S-SCP sont donc bien des codes de type *Repeat Accumulate* dont la partie non bi-diagonale de la matrice a été décomposée suivant la relation 3.31. Il faut aussi noter que ces codes peuvent être interprétés comme des codes LDPC *multi-edge* [102]. Une représentation de type *multi-edge* du graphe du code est illustrée sur la figure 3.19.

Chronologiquement, nos premières études se sont focalisées sur cette famille de codes, et plus particulièrement sur les algorithmes de décodage et le séquencement. La sous-section suivante présente succinctement quelques résultats sur l'étude de ces codes hydrides.

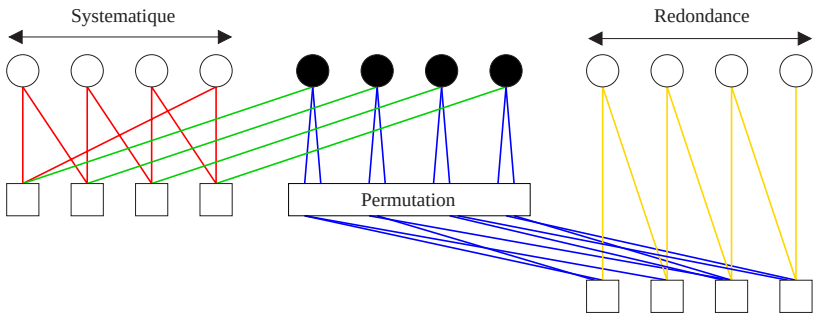


FIG. 3.19 – Représentation simplifiée du graphe du code S-SCP étudié dans [60]. Le code interne et externe sont des codes dont le treillis est à deux états. La représentation reprend le formalisme introduit par Richardson *et al.* dans [102] où chaque type de branche est représenté par une couleur différente.

3.4.2 Étude d'architecture de décodage

La diversité des séquencements possibles pour le décodage de ces codes font de cette famille un très bon cas d'étude. En effet, les codes S-SCP étudiés peuvent être interprétés indifféremment comme :

- des codes LDPC avec des noeuds de données cachés. La matrice de contrôle de parité est alors de la forme :

$$\mathbf{H} = \begin{bmatrix} \mathbf{G_1} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{V} & \mathbf{H_p} \end{bmatrix}$$

  Le code ainsi défini peut donc être décodé en utilisant un algorithme de type BP ou l'un de ses dérivés.

- des codes LDPC de type Repeat Accumulate dont la matrice de contrôle parité s'écrit :

$$\mathbf{H} = [\mathbf{VG_1}\ \mathbf{H_p}]$$

  Le décodage peut donc se réaliser par un algorithme de type BP ou Turbo BP tel que illustré précédement.

- des codes LDPC de type Repeat Accumulate précodé. Dans cette vision, le décodeur se décompose en deux sous-décodeurs. Le premier, le décodeur externe, décode le code convolutif externe dont la matrice de contrôle de parité est la suivante :

$$\mathbf{H}^{ext} = [\mathbf{G_1}\ \mathbf{I}]$$

  Le second code, le code interne, est un code LDPC de type Repeat Accumulate dont la matrice de contrôle de parité s'exprime :

$$\mathbf{H}^{int} = [\mathbf{V}\ \mathbf{H_p}]$$

  Comme nous l'avons vu précédemment, le décodage de ce type de code peut être réalisé suivant différentes stratégies exposées dans les sections précédentes.

- une concaténation hybride de codes simples [103]. Le code est alors vu comme la concaténation de trois codes. Le premier est un code convolutif de fonction de transfert $1 + D$. Le second est un code LDGM dont la matrice de contrôle de parité s'exprime de la manière suivante :

$$\mathbf{H}^{LDGM} = [\mathbf{V}\ \mathbf{I}] \tag{3.37}$$

  Le troisième code est quant à lui un accumulateur.

Les travaux présentés précédemment dans ce chapitre sont issus de réflexions concernant cette structure de codage. En particulier, nous avons défini une sous-famille de codes S-SCP, dont la matrice de contrôle de parité est construite à partir

de matrices identité permutées [104]. Nous avons ensuite étudié le séquencement de décodage et les règles à appliquer sur le code pour pouvoir réaliser un décodage suivant l'algorithme *Turbo Layered BP*. Ces résultats ont été présentés dans les sections précédentes.

En plus des résultats déjà présentés, nous nous sommes intéressés à la définition conjointe de codes et d'architectures dans un cas particulier. L'intérêt principal de cette structure de codage étant sa simplicité de décodage, nous nous sommes intéressés à la définition d'une architecture bas coût. L'algorithme initialement proposé dans [60] consiste au décodage séquentiel de deux sous-codes, le code convolutif externe et le code de type Repeat Accumulate. Les débits atteints dans le cas d'un code de rendement 1/2, sous l'hypothèse d'un séquencement série et du traitement en un cycle d'une section de treillis, sont de l'ordre de $1/(3it)f_{clk}$ ($\simeq K$ cycles pour le treillis du code externe et $\simeq 2K$ pour le code interne, cf séquencement série section 3.3.1.1). Pour la même complexité de décodage il est possible d'améliorer le débit si on considère que la définition du code permet de commencer le décodage du code interne avant la fin du décodage du code externe. De la même manière il est possible de determiner des contraintes pour permettre le décodage du code externe avant la fin du décodage du code interne. Ce type de considérations n'est pertinent que dans le cas d'un degré de parallélisme assez faible. Cette hypothèse est à l'opposé de l'architecture parallèle proposée dans [60] où un niveau de parallélisme de 90 est mis en oeuvre pour obtenir les débits annoncés.

Dans le cas de codes S-SCP définis à partir de matrices de contrôle de parité basées sur des matrices identité permutées, il est possible de calculer le paramètre de pipeline définissant la latence entre le début du décodage du code externe et le code interne à partir des coefficients de permutation définissant $\mathbf{V}$. Nos études sur le séquencement du décodage ont permis de montrer l'intérêt de définir le code externe comme un code circulaire. Le degré de liberté dans le choix de la premiere section de treillis à traiter permet d'optimiser le paramètre de pipeline. Une architecture matérielle a été développée en VHDL. La réalisation matérielle du décodeur a montré qu'il est assez difficile de définir une architecture utilisant pleinement ces propriétés théoriques pour une large gamme de tailles de rendement de codage. En effet, alors que pour des grandes tailles et des rendements faibles il est possible de définir de bons codes avec un paramètre de pipeline donné, il est beaucoup plus difficile de concevoir des codes ayant de bonnes propriétés pour des petites tailles ou des rendements forts. Il est aussi possible de définir un décodeur avec un séquencement des opérations dépendant du paramètre de pipeline du code. Ce type de décodeur est tout à fait possible à réaliser mais engendre une complexité de contrôle additionnelle.

3.4.3 Synthèse

L'étude du cas des codes S-SCP nous a permis de réaliser un travail de réflexion sur la définition d'un bon compromis entre les différents ordonnancements. Les résultats

spécifiques aux code S-SCP ont été très brièvement présentés dans cette sous-section. Ces résultats ont montré comment relier la définition de la matrice de contrôle de parité avec le paramètre de pipeline entre les différents décodages des codes internes et externes. Les études d'architectures et de séquencements de décodage présentés succinctement dans cette section sont à l'origine de l'ensemble des résultats illustrés dans ce chapitre, dont notamment l'étude du séquencement *Turbo Layered BP*.

3.5 Conclusion

Ce chapitre a présenté des études d'architectures relatives à des algorithmes et séquencement différents. Dans un premier temps, une étude d'architecture associée à l'algorithme *Layered BP* a été présentée dans le cas de codes dont les matrices de contrôle de parité sont construites à partir de matrices identité permutées. Cette étude a identifié et illustré des règles de construction de codes pour différentes architectures de processeurs CNP. Ces règles ont aussi été dérivées dans le cas d'un séquencement pipeliné, où le but est de trouver un compromis entre débit et parallélisme.

Dans une seconde partie, la définition et l'étude d'architectures pour des algorithmes de type *Turbo Layered BP* ont été discutées. Plusieurs séquencements ont été présentés et analysés. A partir des ordonnancements présentés, des règles de construction de codes ont été dérivées. Chaque séquencement peut répondre à une contrainte de débit et/ou de complexité. La mise en oeuvre d'un décodeur générique au travers du décodage d'une matrice équivalente a été présentée. La solution proposée est une réponse à ce qui est considéré encore aujourd'hui comme un des points faibles imputés aux codes LDPC. Les résultats de cette section seront mis en oeuvre dans le cadre de la réalisation matérielle d'un décodeur. Des éléments de réalisation ainsi que des résultats liés à ces solutions seront illustrés dans le chapitre suivant.

La dernière partie de ce chapitre a consisté à illustrer le cas d'architectures hybrides de décodage. Comme nous avons pu le mentionner, les résultats présentés dans ce chapitre sont issus de l'étude de ces architectures hybrides. C'est l'analyse de ces structures hybrides qui nous a amené à considérer des architectures de décodage dérivées des considérations habituellement liées aux codes convolutifs et aux Turbo-codes.

Chapitre 4

Éléments de réalisation d'un décodeur LDPC

Résumé

Ce dernier chapitre a pour but de démontrer comment les résultats illustrés dans les chapitres précédents peuvent être mis en oeuvre dans la conception matérielle d'un décodeur. Dans un premier temps, nous explicitons les choix retenus pour la réalisation matérielle. Une fois le séquencement et l'algorithme de décodage choisis, les différentes structures des processeurs de calculs sont discutées et détaillées. Les éléments relatifs à l'intégration d'un décodeur dans un composant programmable de type FPGA sont alors illustrés (quantification et organisation des mémoires). Enfin, des résultats de complexité et de performance sont présentés pour différents contextes. En particulier, une comparaison avec un turbo décodeur duo-binaire 8 états est détaillée.

4.1 Préambule

Les précédents chapitres se sont intéressés à la construction des codes et à la définition d'architectures de décodage. Le chapitre 2 a présenté des outils pour concevoir des "bons"codes. Le chapitre 3 a décrit des règles de construction associées à des architectures pour différentes stratégies de décodage. Le but de ce dernier chapitre est donc de montrer comment les résultats présentés peuvent être mis en oeuvre dans la réalisation matérielle d'un décodeur.

Nous proposons d'illustrer dans ce chapitre la mise en place d'un décodeur fonctionnant suivant l'algorithme *Turbo Layered BP*. Cet algorithme de décodage converge plus vite que ses concurrents par un séquencement particulier utilisant les propriétés du code. Cette solution n'a pas, à notre connaissance, été explorée dans l'état de l'art, contrairement aux architectures utilisant un algorithme de type *Layered BP*. Dans une première partie nous illustrerons quelques points clefs relatifs à la réalisation de

ce décodeur. Nous présenterons l'architecture et les simplifications utilisées pour la réalisation des processeurs de calcul. Nous aborderons également la problématique de la quantification des données internes du décodeur et l'agencement des organes de mémorisation. Dans une seconde partie, le choix des paramètres du décodeur sera discuté ainsi que la chaîne de test associée. Enfin, une dernière partie synthétisera différents résultats relatifs aux performances et à la complexité du décodeur. Une comparaison sera proposée avec un turbo décodeur duo-binaire 8 états.

4.2 Étude détaillée de la réalisation du décodeur

Dans cette section, nous proposons d'illustrer nos choix de réalisation. Dans un premier temps, la sélection d'un ordonnancement *Turbo Layered BP* pour le décodage sera expliquée. A partir des choix effectués, nous nous intéresserons à la réalisation des processeurs de décodage et à l'organisation des mémoires. La stratégie de quantification et la conséquence sur les performances seront également discutées.

4.2.1 Algorithme de décodage

Avant de définir la structure des processeurs de calcul, il est nécessaire de s'attarder sur le choix de l'algorithme de base utilisé pour la vérification des équations de parité. Comme nous l'avons illustré au cours du premier chapitre, il s'agit de trouver le meilleur compromis entre performance et complexité.

La résolution des équations de parité sans utiliser d'approximation permet d'obtenir les meilleures performances. Cette stratégie nécessite cependant le calcul quasi-exact des log-rapports de vraisemblance à l'entrée du décodeur. Il faut donc être capable d'estimer l'ensemble des paramètres du canal quasi-parfaitement (dont la variance du bruit). Quand une approximation de type minimum est mise en oeuvre, l'algorithme est indépendant, à un facteur multiplicatif près, de la dynamique des données d'entrées. L'utilisation de cette approximation permet aussi de réduire le nombre de messages de branches, m_{cv}, à mémoriser (cf chapitres précédents). Cependant, comme tout algorithme sous-optimal, cette approximation s'accompagne de pertes de performances.

De manière à minimiser la complexité et garder la propriété de l'opérateur minimum[(1)], nous proposons d'utiliser un algorithme du type *normalized Min-Sum*. Les messages en sortie du traitement d'un noeud de contrôle sont multipliés par un facteur multiplicatif inférieur à 1. La pondération des messages peut se réaliser de différentes manières. Une première solution consiste à effectuer la pondération directement dans le noyau de calcul de la fonction $g(.)$. Cette fonction peut alors se définir par :

$$g(x, y) = \alpha \ \mathrm{sign}(x) \ \mathrm{sign}(y) \ \min\left(|x|, |y|\right) \tag{4.1}$$

[(1)] qui permet la réduction du nombre de messages à stocker, cf chapitres précédents

où α est un coefficient de pondération inférieur à 1. Cette solution produit de bons résultats quand le coefficient α est proche de 0.9 [26]. Le processeur de résolution d'équations de parité étant défini pour être un processeur à J entrées, le calcul du message de sortie peut s'exprimer de la manière suivante :

$$\begin{aligned} g(\{x_i,\ i=0,J-1\}) &= g(x_0, g(x_1, g(x_2, ...))) \\ &= \alpha^J \prod_{i=0}^{J-1} \text{sign}(x_i) \min_{i=0,J-1} (|x_i|) \end{aligned} \tag{4.2}$$

Une seconde méthode consiste donc à appliquer un coefficient de pondération égal à α^J aux messages en sortie du processeur associé à un noeud de contrôle. Cette méthode est moins complexe que la pondération des messages dans le noyau, le nombre d'opérations étant réduit. On notera β_F le coefficient de pondération en sortie du processeur SPC forward, β_B le coefficient de pondération en sortie du processeur SPC backward et β_g le coefficient de pondération intervenant dans la fonction $g(.)$ utilisée par le décodeur traitant le treillis de l'accumulateur. Pour faciliter le processus de pondération, nous chercherons à définir des coefficients qui peuvent se développer en puissance de 2[(2)]. La fonction de pondération se réalisera alors par des décalages et des additions. Des performances de différentes configurations sont présentées sur la figure 4.1 dans le cas d'un code de rendement 1/2 et de taille $N = 3600$ avec $J = 4$. Les différentes configurations simulées sont les suivantes :

- *Sans Approximation.* On utilise la formulation exacte de la fonction $g(.)$ en considérant les paramètres du canal parfaitement estimés. Cette configuration apporte les meilleures performances.
- *Approximation du* min. La fonction $g(.)$ est approchée par la fonction minimum. A un taux d'erreur binaire de 10^{-5} cette approximation entraîne une perte d'environ 0.35dB.
- *Approximation du* min *avec pondération dans le noyau.* La fonction $g(.)$ est approchée par la fonction minimum pondérée par 0.9375 qui est aussi égal à $2^{-1}+2^{-2}+2^{-3}+2^{-4}$. La perte de performance est inférieure à 0.05dB à un taux d'erreur binaire de 10^{-5}.
- *Approximation du* min *avec* $\beta_f = 0.75$, $\beta_b = 0.75$, $\beta_g = 1$. Les messages en sortie du noeud de contrôle au niveau du processeur SPC forward et SPC backward sont pondérés par 0.75. Le noyau de la fonction $g(.)$ utilisé pour décoder le treillis de l'accumulateur est approximé par une simple fonction minimum. Il faut noter que les coefficients de pondération n'ont pas été choisis au hasard puisque $0.75 = 2^{-1} + 2^{-2} \simeq \sqrt[4]{0.9375}$. La perte de performance est inférieure à 0.1dB à un taux d'erreur binaire de 10^{-5} par rapport à un algorithme sans approximation.
- *Approximation du* min *avec* $\beta_f = 0.75$, $\beta_b = 1$, $\beta_g = 1$. Dans ce cas une seule pondération est appliquée à la sortie du processeur SCP forward. La perte de performance est alors inférieure à 0.15dB à un taux d'erreur binaire de 10^{-5} par rapport à un algorithme sans approximation.

(2) Cette pondération peut aussi se réaliser à l'aide d'une table

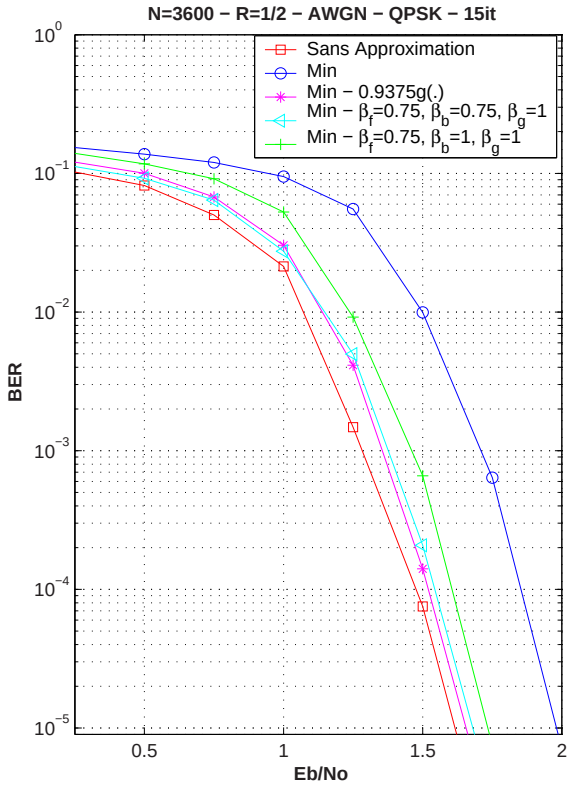


FIG. 4.1 – Exemple de performances, en terme de taux d'erreur binaire (BER), de différentes approximations.

Ces résultats illustrent la manière de compenser les pertes dues à l'utilisation de l'approximation du minimum dans l'algorithme de décodage. Ces pertes peuvent donc être diminuées en utilisant des stratégies plus ou moins complexes. Il s'agit encore une fois de trouver le compromis entre performance et complexité. Les solutions consistant à pondérer les messages de branches par β_f et β_b semblent les techniques les plus intéressantes. Les simulations ayant été réalisées en virgule flottante, celles-ci ne reflètent pas forcément le comportement du décodeur tel qu'il sera intégré sur un composant. Par exemple, une multiplication d'un entier par un facteur inférieur à 1 peut entraîner une perte d'information sur le signe et introduire une erreur (un effacement). Une fois que les phénomènes liés à la représentation des données sur un nombre fini de bits auront été présentés, le choix de la stratégie de pondération sera illustré. Avant de rentrer dans l'analyse de la quantification du décodeur, nous proposons d'illustrer l'architecture qui a été choisie pour la réalisation des différents processeurs de calcul.

4.2.2 Processeur de calculs

Comme nous avons pu le mentionner précédemment, l'architecture se compose de trois processeurs. Afin de bien comprendre le déroulement du décodage, la ma-

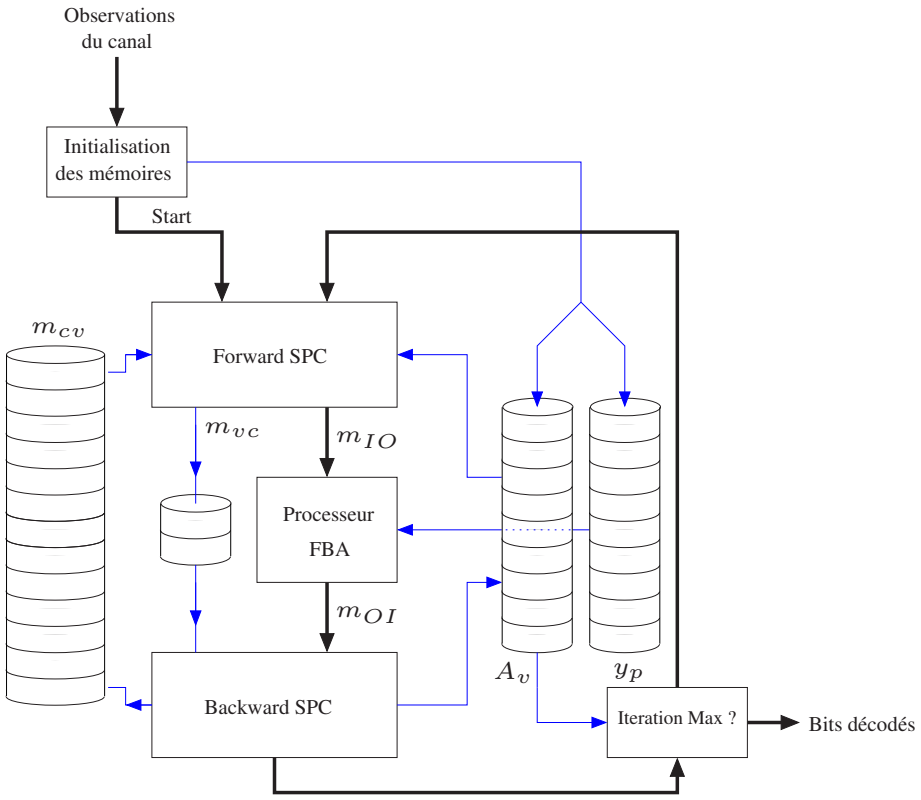


FIG. 4.2 – Illustration du fonctionnement du décodeur. Une fois que les mémoires sont initialisées (représentées par des cylindres), le décodage de la trame peut se réaliser suivant le séquencement décrit. La déclaration de fin de décodage dépend du nombre d'itérations maximum choisi.

chine d'état associée au décodeur est illustrée sur la figure 4.2. Un premier processeur calcule à la fois les messages se propageant d'un noeud de données vers le noeud de contrôle, et les messages à transmettre au décodeur externe (SPC forward). Un second processeur traite une fenêtre de décodage du treillis de l'accumulateur. Le dernier processeur (SPC backward) calcule les messages se propageant du noeud de contrôle vers les noeuds des données et ré-estime l'information *a posteriori*. Nous proposons dans cette section d'étudier précisément ces processeurs. Des simplifications algorithmiques seront discutées dans le cas où un algorithme utilisant l'approximation du minimum est utilisé pour la résolution des équations de parité.

4.2.2.1 Processeurs SPC

Dans le cas d'une approximation de type minimum, les équations de décodage au niveau du processeur SPC forward peuvent s'écrire de la manière suivante :

$$\begin{aligned} m_{jc}^{i} &= A_{j}^{u_0} - m_{cj}^{i-1}, \quad \forall j \in V_c' \\ m_{IO}^{i}(c) &= \beta_F \left(\prod_{j \in V_c'} \operatorname{sign}\left(m_{jc}^{i}\right) \right) \min_{j \in V_c'} \left| m_{jc}^{i} \right| \end{aligned}$$

Dans le cas du processeur SPC backward, les équations de mises à jour de l'information *a posteriori* et du calcul des messages se propageant du noeud de contrôle vers les noeuds de données s'expriment de la manière suivante :

$$\begin{aligned} m_{cv}^{i} &= \beta_B \left(\operatorname{sign}\left(m_{OI}^{i}(c)\right) \prod_{j \in V_c'/v} \operatorname{sign}\left(m_{jc}^{i}\right) \right) \min \left(\left| m_{OI}^{i}(c) \right|, \min_{j \in V_c'/v} \left| m_{jc}^{i} \right| \right) \\ A_{v}^{u_0+1} &= m_{vc}^{i} + m_{cv}^{i} \end{aligned}$$

Nous avons mentionné au chapitre 3 que, pour réduire le nombre de calculs et d'accès à la mémoire, les messages m_{jc}^{i} $\forall j \in V_c'$ sont mémorisés dans un buffer pour être réutilisés par le processeur SPC backward. Dans la suite, les indices u_0 et i seront volontairement omis pour ne pas surcharger les notations.

L'équation de calcul des messages m_{cv} au niveau du processeurs SPC backward fait apparaître une fonction minimum entre une partie des messages m_{vc} et le résultat du décodage du treillis de l'accumulateur $m_{OI}(c)$. Ce calcul a déjà été réalisé en partie lors du traitement des données par le décodeur SPC forward. En effet, pour calculer le message à transmettre au processeur traitant une fenêtre du treillis de l'accumulateur, les messages m_{vc} ont été comparés entre eux. Une partie de cette information peut donc être utilisée par le processeur SPC backward. Le traitement du calcul des valeurs absolues des messages m_{cv} peut donc être défini par de simples comparaisons. Le calcul des messages se propageant du noeud de contrôle vers les noeuds de données peut donc s'écrire :

$$\begin{array}{l} \textbf{if } |m_{OI}(c)| < |m_1| \\ \textbf{then} \\ \quad |m_{cj}| = |m_{OI}(c)|, \quad \forall j \in V_c \\ \textbf{else if } |m_{OI}(c)| > |m_2| \\ \textbf{then} \\ \quad |m_{cj}| = |m_1|, \quad \forall j \in V_c/i_1 \\ \quad |m_{ci_1}| = |m_2| \\ \textbf{else} \\ \quad |m_{cj}| = |m_1|, \quad \forall j \in V_c/i_1 \\ \quad |m_{ci_1}| = |m_{OI}(c)| \\ \textbf{end} \end{array} \tag{4.3}$$

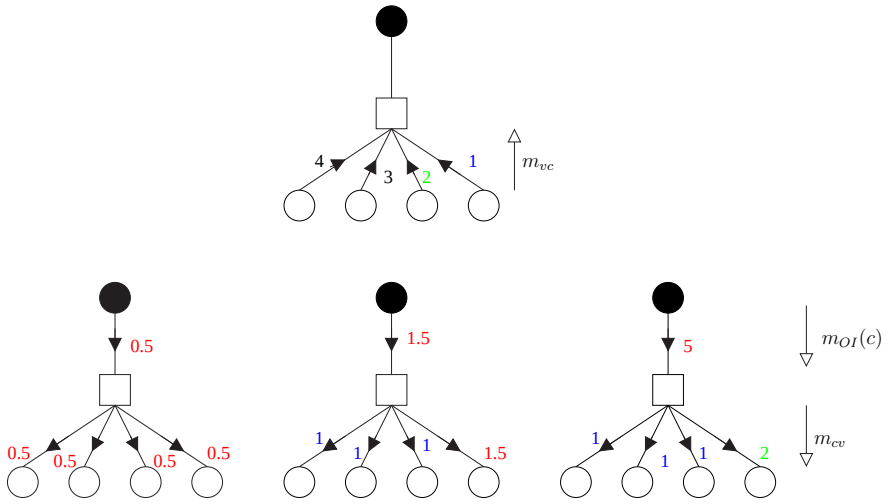


FIG. 4.3 – Exemple de calcul de messages m_{cv} en fonction des messages m_{vc} et m_{OI}

où $|m_1|$ et $|m_2|$ sont définis par :

$$|m_1| = \min_{j \in V_c'} |m_{jc}| \tag{4.4}$$

$$i_1 = \arg\min_{j \in V_c'} |m_{jc}| \tag{4.5}$$

$$|m_2| = \min_{j \in V_c'/i_1} |m_{jc}| \tag{4.6}$$

En d'autres termes, $|m_1|$ représente la plus petite valeur parmi les messages $|m_{vc}|$, i_1 l'indice de position de cette valeur, et $|m_2|$ la deuxième plus petite valeur. Un exemple simple de calcul des différents messages est illustré sur la figure 4.3. Cette simplification permet de s'affranchir du calcul de J fonctions minimum à J entrées au niveau du processeur SPC backward. Il faut cependant garder à l'esprit que le processeur SPC forward doit donc estimer les deux plus petites valeurs et la position de l'une des deux valeurs, ce qui augmente la complexité de la structure du processeur par rapport à une simple fonction minimum. Pour permettre l'utilisation du traitement proposé, il est aussi nécessaire de mémoriser deux indices codés sur $\lceil \log_2 J \rceil$ bits, qui seront stockés dans le buffer à l'interface des processeurs SPC forward et backward. Par exemple, dans le cas où $J = 4$, l'utilisation de ce traitement s'accompagne d'une augmentation de la taille du buffer d'un facteur inversement proportionnel à la quantification des messages m_{vc}. Cette solution permet donc de réduire le nombre de calculs au prix d'une augmentation de la taille du buffer.

A partir de ces observations, nous pouvons définir les structures des processeurs de calcul. La figure 4.4 présente une architecture pipelinée pour le processeur SPC forward où $J = 4$. Après la soustraction de l'information *a posteriori* A_v par l'information extrinsèque m_{cv}, les messages résultants sont mémorisés dans un buffer. Le plus petit de ces messages en valeur absolue est pondéré puis propagé au processeur associé au treillis de l'accumulateur. Les indices des deux plus petits messages sont

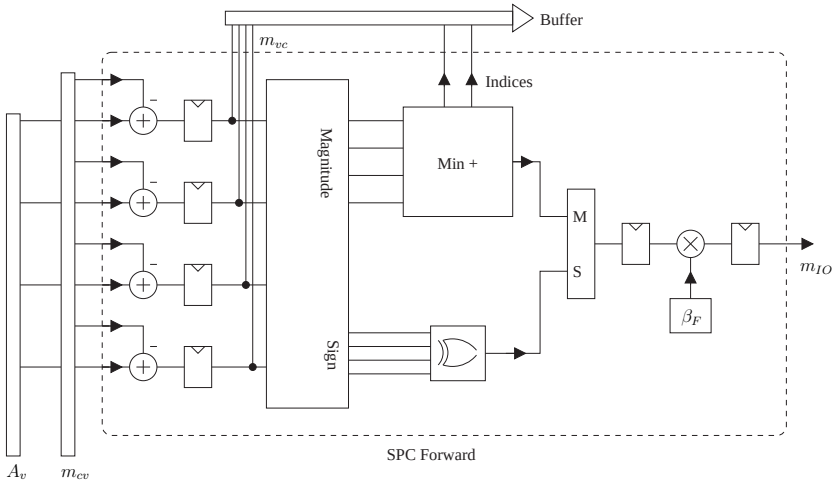


FIG. 4.4 – Architecture pipelinée du processeur SPC forward. Les étages de pipeline sont modélisés par des bascules. Les accès aux mémoires sont représentés par des bus et sont modélisés par des rectangles.

également mémorisés dans le buffer.

Le processeur SPC backward est quant à lui illustré sur la figure 4.5. Il se compose d'une fonction logique caractérisant l'équation 4.3 suivi de J additionneurs fonctionnant en parallèle. Nous verrons dans la suite que cette architecture sera légèrement modifiée de manière à prendre en compte la quantification et les propriétés relatives à la nature des messages m_{vc} à mémoriser. En effet, comme nous avons pu le montrer, il n'existe que deux messages différents en valeur absolue à la sortie d'un noeud de contrôle (cf figure 4.3).

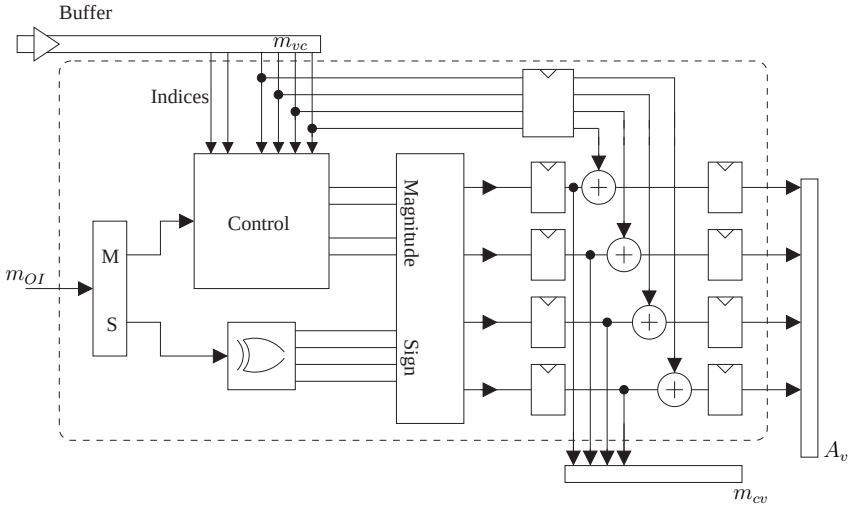


FIG. 4.5 – Architecture pipelinée du processeur SPC backward. Les étages de pipeline sont modélisés par des bascules.

4.2.2.2 Processeur FBA

Le processeur traitant le treillis de l'accumulateur est composé de deux sous-processeurs, un processeur forward et un processeur backward. Ce processeur est décrit sur la figure 4.6. Dans sa version pipelinée, la réalisation matérielle du décodeur nous amène à considérer deux processeurs. Dans une version non pipelinée, un seul processeur peut être mis oeuvre. Une attention particulière doit être prise dans la définition de la profondeur des mémoires de type LIFO (Last In First Out). Dans le cas pipeliné cette mémoire doit pouvoir compenser toute la latence introduite par le processeur forward. Comme nous l'avons déjà mentionné, l'initialisation des fenêtres est réalisée en utilisant la méthode du pointeur. Les initialisations sont gérées par des simples FIFO. Dans le cas de la mise en oeuvre de parallélisme, i.e. que le treillis est découpé en plusieurs sections décodées en parallèle, il est possible d'initialiser les dernières fenêtres de l'itération i par une métrique calculée elle aussi à l'iteration i. Un exemple est illustré sur la figure 4.7.

Dans cette section nous avons détaillé l'architecture des processeurs de calcul. L'utilisation d'un algorithme utilisant l'approximation du minimum permet de simplifier les traitements au niveau du processeur SPC backward. Cette simplification s'accompagne cependant d'une légère augmentation de la profondeur du buffer qui interface le processeur SPC forward et SPC backward. Avant de décrire l'organisation des mémoires, nous proposons de discuter les choix des paramètres de quantification du décodeur.

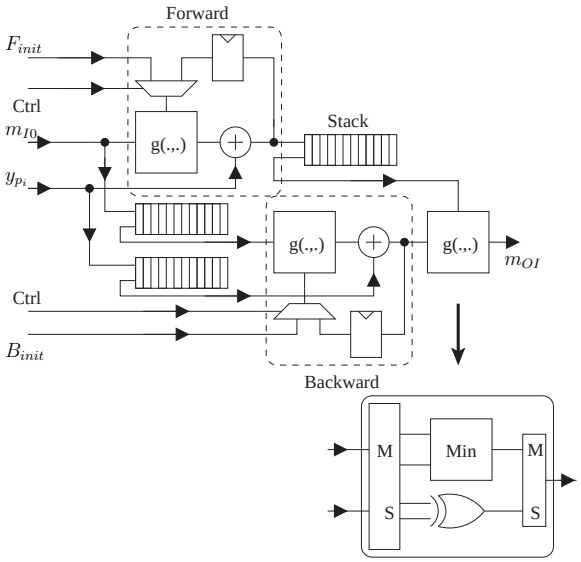


FIG. 4.6 – Architecture pipelinée du processeur FBA décodant le treillis de l'accumulateur.

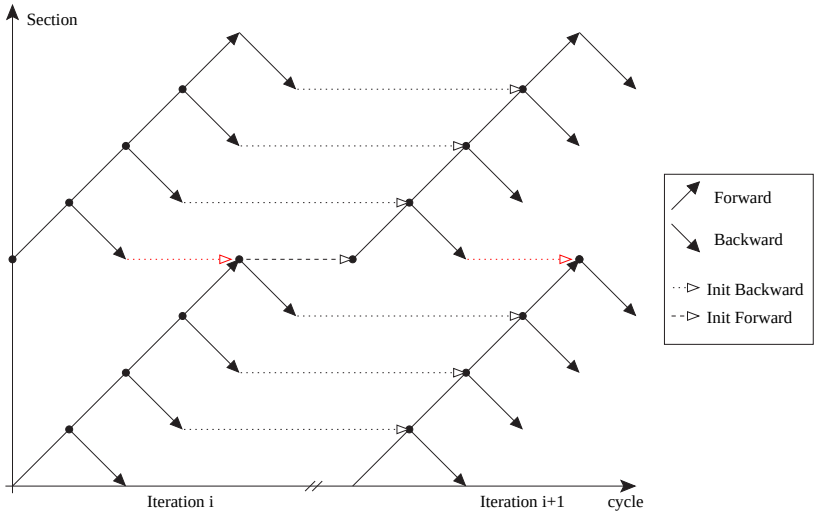


FIG. 4.7 – Exemple d'initialisation de fenêtre dans le cas d'un parallélisme de 2. Il est possible d'initialiser les dernières fenêtres de l'itération $i+1$ par une métrique calculée elle aussi à l'iteration $i+1$.

4.2.3 Quantifications

La quantification des chemins de données dans le décodeur est un point très important dans la réalisation matérielle. Plus la quantification sera large, meilleures seront les performances, mais plus complexe sera le décodeur. Il s'agit donc de trouver le meilleur compromis entre performance et complexité. Dans un premier temps, la méthode d'analyse de la quantification sera décrite. Dans un second temps, la quantification des entrées du décodeur sera discutée. Enfin, l'analyse théorique de la quantification des chemins de données sera proposée. Différentes stratégies de quantification seront alors étudiées.

4.2.3.1 Méthode d'analyse

L'analyse de la quantification des différents chemins de données du décodeur peut être réalisée très simplement. Le fait d'utiliser pour l'algorithme de décodage des opérateurs d'additions, de soustractions et de comparaisons, permet une étude très simple de la quantification des chemins de données. La méthode d'analyse de la quantification se décompose donc en trois phases. Une première phase consiste à déterminer la quantification des signaux d'entrée, le décodeur fonctionnant avec une précision suffisamment grande. La quantification des observations va influer directement sur le choix de la quantification interne du décodeur et donc sur la complexité. L'objectif de cette première phase est donc d'optimiser les performances sous une contrainte de minimisation de la complexité.

Une fois la quantification des données d'entrée choisie, une analyse de la quantification des chemins de données peut être réalisée. Le but de cette analyse est d'identifier

la quantification du chemin de données qui va permettre de déterminer les autres quantifications. Une fois cette quantification identifiée, des simulations de performances peuvent être menées de manière à optimiser les performances sous une contrainte de minimisation de la complexité [(3)].

Une fois cette étude menée, il peut être intéressant d'étudier les effets de seuillage sur les quantifications de manière à minimiser les ressources nécessaires. Cette approche pragmatique permet de déterminer une configuration des quantifications amenant à un décodeur avec une complexité minimisée pour des performances données. Cette méthode d'analyse, assez classique, sera utilisée pour la détermination des quantifications de l'architecture de décodeur étudiée.

4.2.3.2 Quantification du canal

La quantification des données provenant du canal est un paramètre très important pour la définition de la quantification des chemins de données du décodeur. La phase de quantification s'accompagne nécessairement d'une perte d'information qui peut avoir plus ou moins de conséquences sur le fonctionnement du décodeur. Nous proposons d'étudier ces effets dans ce paragraphe.

Tout d'abord, les observations issues du canal sont considérées comme suivant une loi de probabilité gaussienne de moyenne m et de variance σ^2. Par hypothèse une modulation de type QPSK (Quadrature Phase Shift Keying) est utilisée. Ces données sont quantifiées sur Q_c bits. Les données en entrée du décodeur sont donc comprises entre $2^{Q_c-1}-1$ et $-2^{Q_c-1}+1$[(4)]. Dans une première étape, le signal reçu sera écrêté. Nous considérons un seuil égal à $\pm(m+2\sigma)$. Les données sont alors multipliées par un facteur égal à $(2^{Q_c-1}-1)/(m+2\sigma)$ puis arrondi, à l'entier le plus proche dont la valeur absolue est strictement inférieure à 2^{Q_c-1}. Ces deux étapes caractérisent notre quantification des observations du canal.

Pour déterminer la meilleur stratégie de quantification, des simulations ont été réalisées avec différentes valeurs de Q_c. Dans un premier temps, les chemins de données internes du décodeur sont quantifiés avec une précision de 32 bits. On utilise un algorithme de décodage où les coefficients de pondération sont définis par $\beta_f = 0.75$, $\beta_b = 1$, $\beta_g = 1$. Les performances obtenues sont illustrées sur la figure 4.8. Il apparaît qu'une quantification inférieure à 5 bits s'accompagne de pertes de performances. L'utilisation de données représentées sur 4 bits entraîne ainsi une dégradation de moins de 0.15 dB. Dans le cas où Q_c est égal à 3, les pertes sont d'environ 1dB. Ces résultats sont comparables à ceux communément admis dans le cas des Turbo-codes (cf [105] par exemple). Notre but étant de trouver le meilleur compromis entre performances et complexité, nous choisirons une quantification des entrées sur 4 bits. Ce

[(3)] Comme il a été montré dans [25] par exemple, la quantification peut influer sur le phénomène de plancher d'erreur

[(4)] On considère une quantification symétrique et un codage en complément à 2

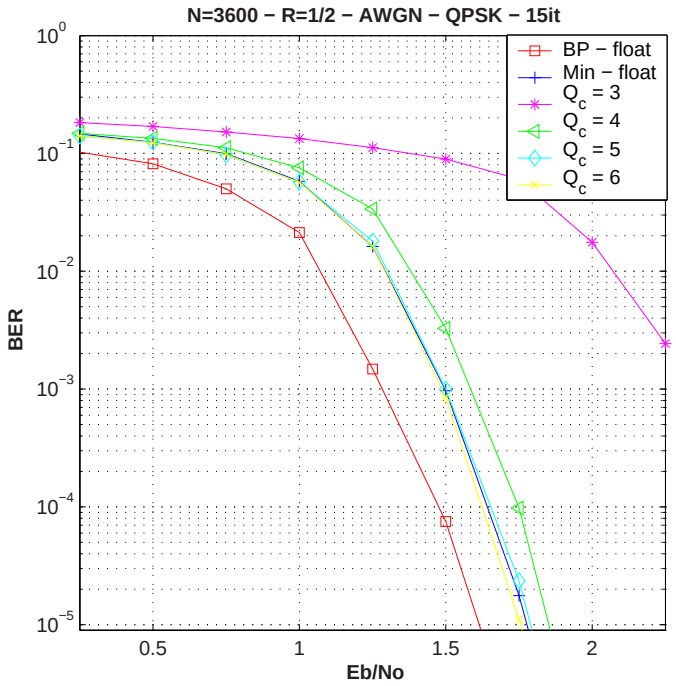
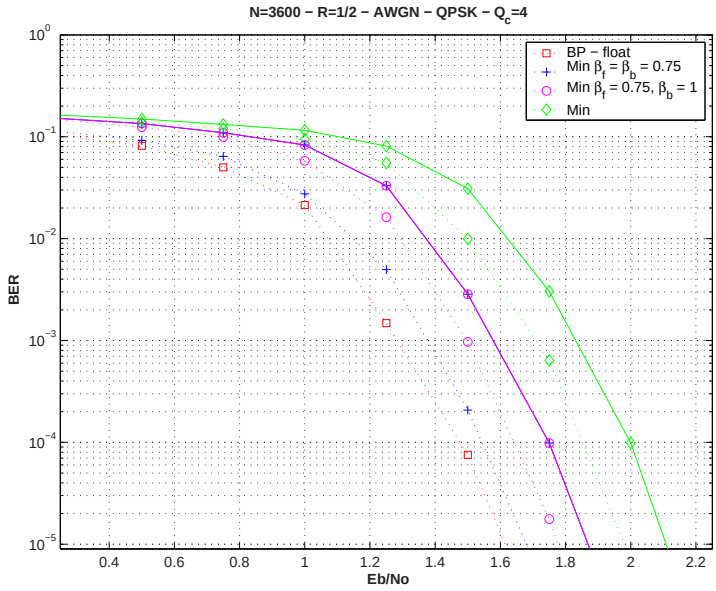


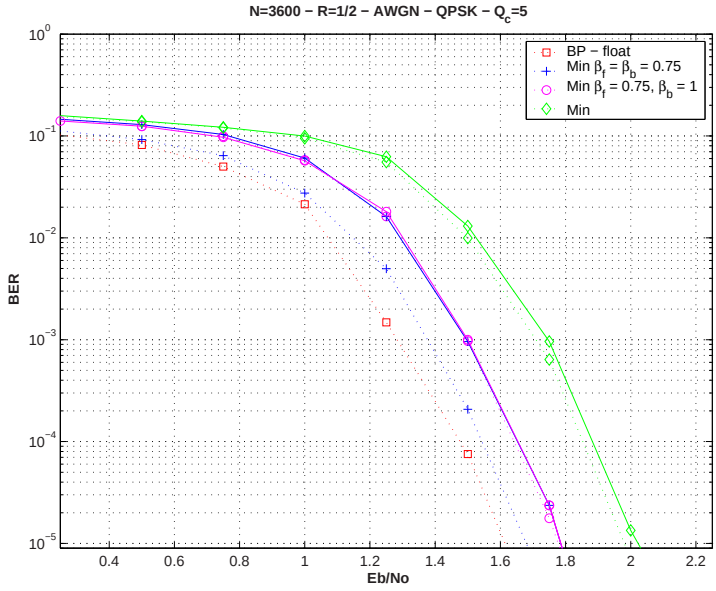
FIG. 4.8 – Illustration de l'effet de la quantification des données d'entrée sur les performances du décodeur. BP correspond à un décodeur en virgule flottante n'utilisant aucune approximation. Min correspond à un décodeur en virgule flottante utilisant l'approximation du minimum avec $\beta_f = 0.75$, $\beta_b = 1$, $\beta_g = 1$.

choix s'accompagne donc d'une dégradation de l'ordre de 0.15 dB mais permet une réduction de la mémoire requise par le décodeur. Les données seront donc comprises entre ± 7.

Le choix de l'utilisation de la séquence de pondération définie par $\beta_f = 0.75$, $\beta_b = 1$, $\beta_g = 1$ apporte le meilleur compromis entre performances et complexité. Les figures 4.9 illustrent les performances des différentes stratégies de pondération dans le cas où la quantification des données d'entrées est sur 4 et 5 bits. Dans le cas d'une approximation de type minimum pondérée ou non, la quantification sur 5 bits n'introduit que très peu de pertes en comparaison à un décodeur en virgule flottante. Dans le cas d'un décodeur en virgule fixe, la pondération de type $\beta_f = 0.75$, $\beta_b = 0.75$ et $\beta_g = 1$ est équivalente en terme de performance, à la stratégie de pondération $\beta_f = 0.75$, $\beta_b = 1$ et $\beta_g = 1$. Ce comportement a donc motivé notre choix de l'utilisation d'une stratégie de pondération où seuls les messages transmis au décodeur du treillis de l'accumulateur sont pondérés.



(a)



(b)

FIG. 4.9 – Illustrations des performances de différentes stratégies de pondération dans le cas d'un décodeur en virgule flottante (courbes continues) et dans le cas d'un décodeur en virgule fixe avec une représentation des données d'entrées sur 4 (a) et 5 bits (b) (courbes pointillées).

4.2.3.3 Analyse de la quantification des chemins de données

Une fois la quantification des données à l'entrée du décodeur fixée, nous pouvons nous intéresser à la quantification des chemins de données internes.

SPC Forward

Dans un premier temps, nous proposons de considérer le processeur SPC forward. Soit Q_A le nombre de bits quantifiant l'information *a posteriori* et $Q_{m_{cv}}$ le nombre de bits quantifiant les messages se propageant d'un noeud de contrôle vers un noeud de données. La première étape de l'algorithme de décodage consiste à calculer les messages se propageant d'un noeud de données vers un noeud de contrôle de la manière suivante :

$$m_{vc} = A_v - m_{cv}$$

Pour éviter tout débordement, le message m_{vc} doit être représenté sur un nombre de bits $Q_{m_{vc}} = \max(Q_A, Q_{m_{cv}}) + 1$. L'étape suivante consiste en une comparaison de ces messages de manière à determiner la plus petite métrique en valeur absolue. Avant pondération, le message en sortie de ce processeur peut donc se représenter sur $Q_{m_{vc}}$ bits. La pondération étant inférieure à 1, le message à l'entrée du processeur traitant le treillis de l'accumulateur sera également représenté sur $Q_{m_{vc}}$ bits.

Processeur FBA

L'analyse de la quantification des chemins de données dans le processeur traitant le treillis de l'accumulateur est aussi très simple à réaliser. Le calcul des métriques forward (resp. backward) est caractérisé par la relation suivante :

$$F_{i+1} = y_p + g(F_i, m_{IO})$$

Le message m_{IO} est codé sur $Q_{m_{vc}}$ bits et la fonction $g(.)$ se caractérise par un fonction minimum s'appliquant sur la valeur absolue des données. Il est alors très simple de montrer que les chemins de données correspondant aux métriques forward et backward doivent être codés sur $Q_{m_{vc}} + 1$ bits sous l'hypothèse que $Q_c < Q_{m_{vc}}$.

Cette quantification peut être réduite si on analyse plus finement la représentation du message m_{IO} après pondération. En effet, la valeur maximum des métriques forward et backward est égale à $2^{Q_c-1} - 1 + \beta_f(2^{Q_{m_{vc}}-1} - 1)$. On peut donc limiter le nombre de bits de quantification des métriques forward et backward à $Q_{m_{vc}}$ (au lieu de $Q_{m_{vc}} + 1$) si le coefficient de pondération β_f suit la relation suivante :

$$\beta_f < \frac{\left(2^{Q_{m_{vc}}-1} - 1\right) - \left(2^{Q_c-1} - 1\right)}{2^{Q_{m_{vc}}-1} - 1} \tag{4.7}$$

Si on considère le cas où $Q_c = 4$ et $Q_{m_{vc}} = 8$, on peut réduire la quantification interne du processeur traitant le treillis de l'accumulateur si β_f est inférieur à 0.94. On rappelle que en pratique nous considérons un facteur $\beta_f = 0.75$ pour $J = 4$. Ce gain de 1 bit n'est pas négligeable puisqu'il influe à la fois sur les chemins de données et sur la taille des buffers présents dans l'architecture du décodeur traitant le treillis de l'accumulateur (cf figure 4.6).

SPC Backward

La dernière phase de l'algorithme consiste, au niveau du processeur SPC backward, à comparer des données quantifiées sur $Q_{m_{vc}}$ bits avec les données issues du décodage de l'accumulateur. En considérant que ces données sont codées sur $Q_{m_{OI}}$ bits, les messages se propageant des noeuds de contrôle vers les noeuds de données doivent être codés sur $\min(Q_{m_{OI}}, Q_{m_{vc}})$ bits. Cette propriété vient du fait que les messages en sortie d'un noeud de contrôle sont une fonction des deux plus petites valeurs entrantes. Dans le cas où la relation 4.7 est vérifiée, ces messages sont donc quantifiés sur $Q_{m_{vc}}$ bits.

Dans une dernière étape, l'information *a posteriori* est mise à jour par le processeur backward SPC. Il faut noter que cette information est la somme de la contribution du canal et de l'ensemble des informations extrinsèques provenant des noeuds de contrôle. Pour éviter la perte d'information, la quantification de l'information *a posteriori* doit donc suivre la relation :

$$Q_A = 1 + \left\lceil \log_2 \left((2^{Q_c - 1} - 1) + d_v (2^{Q_{m_{cv}} - 1} - 1) \right) \right\rceil \tag{4.8}$$

où d_v est le degré de connexion maximum des noeuds de données. A titre d'exemple, dans le cas où les observations du canal sont quantifiées sur $Q_c = 4$ bits, l'information extrinsèque sur $Q_{m_{cv}} = 5$ bits, et un degré de connexion maximum des noeuds de données de $d_v = 13$ (paramètre du LDPC de DVB-S2), alors la quantification de l'information *a posteriori* doit être de $Q_A = 9$ bits contre $Q_A = 7$ dans le cas où le degré de connexion maximum serait de 6. Dans le cas de codes avec des forts degrés de connexion, cette dynamique importante peut être un point critique. Il est donc important de trouver de bons profils d'irrégularité avec des degrés maximum relativement faibles. A même densité de matrice de contrôle de parité, le décodeur sera moins complexe. Ce paramètre doit donc être aussi pris en compte au moment du choix de la distribution des degrés des noeuds de données.

En résumé, si les quantifications Q_A et $Q_{m_{cv}}$ sont définies, il est possible, connaissant le facteur de pondération β_f, de déterminer l'ensemble des quantifications des chemins de données pour éviter tout dépassement. Afin de définir un décodeur le moins complexe possible, nous proposons dans le paragraphe suivant d'étudier des solutions pour réduire la quantification de certaines données. Différentes stratégies seront illustrées et discutées.

4.2.3.4 Impact de la réduction de la dynamique des données internes

Dans le but de réduire la complexité du décodeur, nous proposons d'étudier les effets de réduction de la dynamique sur les performances des codes. Dans un premier temps nous proposons d'étudier le choix du paramètre de quantification de l'information extrinsèque, $Q_{m_{cv}}$. L'étude du nombre de bits de quantification nécessaires pour ces données est très importante d'un point de vue performance et réalisation. Les ressources mémoires nécessaires au décodeur sont fortement dépendantes de la

quantification choisie pour ces messages. En effet, leur nombre est proportionnel au nombre d'éléments non nuls dans la partie non bi-diagonale de la matrice de contrôle de parité. Même si dans le cadre d'une approximation par une fonction minimum il ne faut stocker que les deux plus petits messages en valeur absolue, l'ensemble des J signes, et la position de la plus petite contribution (ou de la deuxième plus petite) pour les M équations de parité, la portion de mémoire allouée à ces métriques est importante[(5)]. Il est donc important de réduire au maximum la quantification de l'information extrinsèque.

Effet du seuillage sur l'information extrinsèque

Les effets de seuillage de l'information extrinsèque pouvant intervenir à des taux d'erreur très bas, nous avons intégré un décodeur sur un composant programmable dont la quantification interne est de 16 bits. Les données d'entrée sont quantifiées sur $Q_c = 4$ bits. La figure 4.10 illustre l'effet du seuillage de la quantification de l'information extrinsèque pour 4 et 15 itérations de décodage. Quand cette quantification est inférieure à celle de l'observation du canal, l'importance est donnée à l'observation plutôt qu'à l'information de décodage. Les performances sont alors relativement mauvaises et un plancher d'erreur apparaît très tôt. Quand les quantifications sont équivalentes, l'observation du canal et l'information extrinsèque ont la même importance. Les performances sont relativement bonnes pour une probabilité d'erreur supérieure à 10^{-6}. Cette stratégie de quantification entraîne par la suite l'apparition d'un plancher d'erreur. Enfin, quand $Q_c < Q_{m_{cv}}$, il est donné plus d'importance à l'information de décodage qu'à l'observation du canal. Quand la quantification de l'information extrinsèque est sur 5 bits, il n'y a pas de plancher d'erreur visible par nos simulations. L'augmentation à 6 bits de quantification n'apporte pas d'amélioration dans la plage de taux d'erreur mesurée.

Suivant le point de fonctionnement du système, on peut donc choisir différentes stratégies. Dans un cas où un taux d'erreur binaire de 10^{-4} est requis, nous choisirons une stratégie de quantification à 4 bits. Dans le cas où des taux d'erreur binaire plus bas sont nécessaires, notre choix se portera sur 5 bits de quantification.

Discussion sur la réduction de la quantification de l'information *a posteriori*

Une fois la quantification de l'information extrinsèque choisie, nous proposons de nous intéresser à celle de l'information *a posteriori*. Dans le cas de forts degrés de connexion des noeuds de données, le nombre de bits nécessaires pour la mémorisation de l'information *a posteriori* peut être très important. Plusieurs stratégies peuvent êtres mises en oeuvre pour réduire cette quantification. Nous proposons dans un premier temps d'illustrer quelques méthodes. Dans une seconde partie, nous discuterons

[(5)] Si cette propriété n'est pas utilisée, il faut $MJQ_{m_{cv}}$ bits de mémoire.

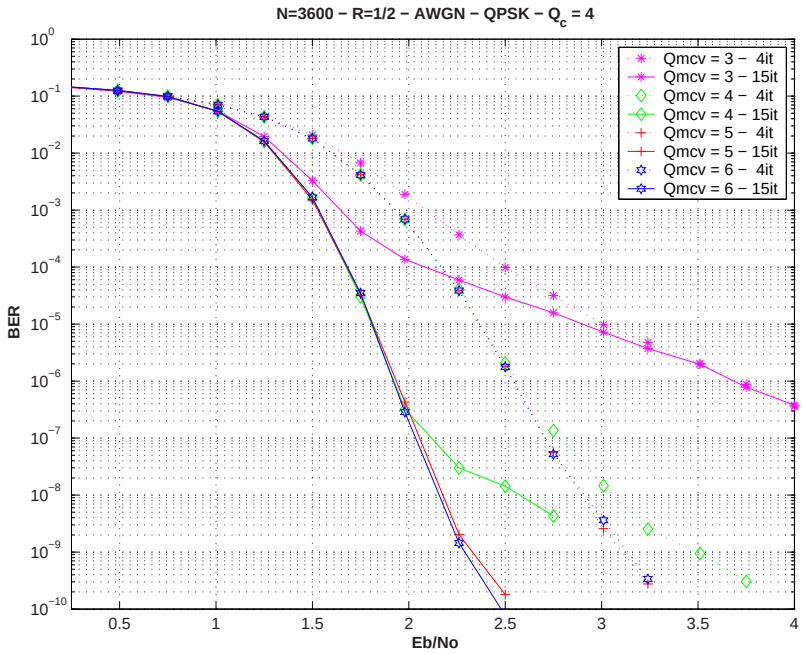


FIG. 4.10 – Illustration de l'effet de la quantification de l'information extrinsèque.

sur la mise en oeuvre de ces techniques.

Une première méthode consiste à compresser l'information à mémoriser. A chaque lecture et écriture, les données sont compressées et dé-compressées. L'utilisation d'une telle méthode s'accompagne d'une augmentation de la complexité de calcul. Cette technique peut être particulièrement intéressante si les ressources mémoires disponibles dans un composant sont très limitées. Cette solution n'a pas été explorée dans nos études mais semble cependant très intéressante.

Les solutions étudiées sont basées sur le seuillage de l'information *a posteriori*. Les différentes méthodes envisagées sont illustrées sur la figure 4.11. La première solution consiste en un simple écrêtage de l'information *a posteriori* sans post-traitement sur l'information extrinsèque. L'un des inconvénients majeurs de cette technique est l'instabilité que peut entraîner la soustraction de l'information *a posteriori* écrêtrée par l'information extrinsèque non post traitée. Cette soustraction peut engendrer dans certain cas l'introduction d'erreurs.

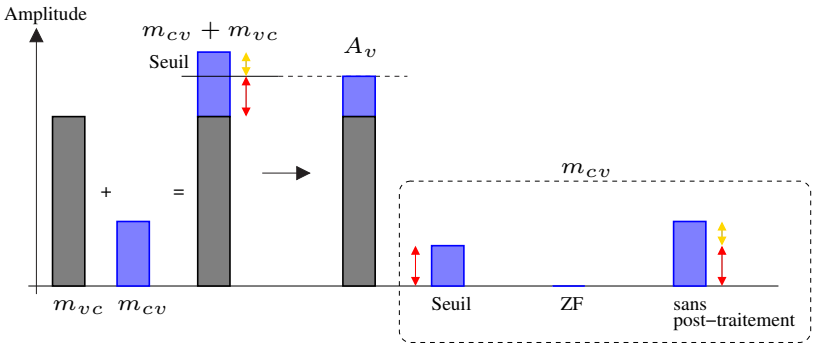


FIG. 4.11 – Illustration des différentes méthodes pour le calcul de l'information extrinsèque après seuillage de l'information *a posteriori*.

Une deuxième solution consiste à écrêter l'information *a posteriori* et à modifier la valeur de l'information extrinsèque de la manière suivante :

$$\begin{aligned}
&\textbf{if } |m_{vc} + m_{cv}| > 2^{Q_A - 1} - 1 \\
&\textbf{then} \\
&\quad A_v = \text{sign}\,(m_{vc} + m_{cv})\left(2^{Q_A-1} - 1\right) \\
&\quad m_{cv} = \text{sign}\,(m_{vc} + m_{cv})\left(2^{Q_A-1} - 1\right) - m_{vc} \\
&\textbf{else} \\
&\quad A_v = m_{vc} + m_{cv} \\
&\textbf{end if}
\end{aligned} \tag{4.9}$$

Cette stratégie est notée "seuil" sur la figure 4.11. Cette méthode permet, au moment du calcul des messages m_{vc}, de ne pas retrancher un message plus important que celui qui a amené à la saturation. En théorie, cette méthode peut éviter l'introduction d'erreurs. Cette méthode a cependant un très gros inconvénient : l'utilisation de cette technique ne permet pas d'utiliser la propriété de réduction du nombre de messages différents, en valeur absolue, en sortie d'un noeud de contrôle. Il peut en effet exister J messages différents (contre 2 si on utilise l'approximation du minimum) une fois l'information extrinsèque post traitée. Dans le cas d'un algorithme utilisant l'approximation du minimum cette méthode est donc très pénalisante.

Dans le même esprit, la valeur du message m_{vc} peut être post traitée et forcée à 0 si l'information *a posteriori* doit être écrêtée[(6)] (notée "ZF" (Zero Forcing) sur la figure 4.11). Cette solution permet de s'affranchir d'un traitement supplémentaire sur l'information extrinsèque et est très simple à mettre en oeuvre. Contrairement à la solution proposée précédemment, il est possible de s'affranchir de la mémorisation de J données par équations de parité. Les messages se propageant d'un noeud de contrôle vers un noeud de données peuvent prendre trois valeurs différentes en valeur absolue. On peut donc reconstruire l'ensemble des messages par la mémorisation des J signes,

[(6)] On se rapproche alors de l'approximation de l'algorithme APP (cf chapitre 1)

des deux plus petites valeurs non nulles, l'indice de la plus petite valeur non nulle et un mot de J bits caractérisant la mise à zéro du message. Malgré cette compression d'information, cette solution est plus complexe en terme de mémorisation par rapport à une solution sans post-traitement.

Une dernière solution, inspirée de la méthode ZF, consiste à ne pas retrancher d'information extrinsèque, au niveau d'un noeud de données, si l'information *a posteriori* a été écrêtée. Contrairement à la technique ZF, qui force localement à zero une branche, cette technique force à zero tous les messages se propageant des noeuds de contrôle vers le noeud de données dont l'information *a posteriori* a été écrêtée. En d'autres termes, on utilise un algorithme de type APP variable (cf chapitre 1) dès lors que l'information *a posteriori* associée à un noeud de données à été écrêtée. Pour mettre en oeuvre cette technique, il faut détecter si la valeur de l'information *a posteriori* est égale à la valeur maximum autorisée par la quantification. Cette étape entraîne un surplus de complexité au niveau du processeur SPC forward mais n'augmente pas la mémoire requise par le décodeur : la valeur de l'information *a posteriori* est testée à chaque calcul faisant intervenir cette valeur. Cette méthode sera appelée méthode APP variable en référence à l'algorithme décrit dans le chapitre 1.

Dans le but de comparer les différentes méthodes présentées, nous considérons un décodeur avec les paramètres de quantification $Q_c = 4$ et $Q_{m_{cv}} = 5$. Le code simulé est un code de rendement 1/2 et de taille $N = 3600$. Ce code est un code irrégulier dont la distribution des noeuds de données est :

$$\tilde{\lambda}(x) = \frac{1}{2}x + \frac{1}{3}x^2 + \frac{1}{6}x^5$$

Le degré maximum de connexion d'un noeud de données étant de 6, la quantification de l'information *a posteriori* doit être de 8 bits pour éviter tout dépassement. Avant de simuler les différentes stratégies, il peut être intéressant d'analyser l'impact des différentes méthodes sur la mémoire totale requise par le décodeur. La figure 4.12 illustre la variation de la mémoire totale du décodeur en fonction de la quantification de l'information *a posteriori*. Les paramètres du décodeur sont $K = 1800$, $R = 1/2$, $Q_{m_{cv}} = 5$, $J = 4$ et $l_p m = 48$. Le paramètre l_p est en relation avec la latence du traitement d'une fenêtre dans un cas où du pipeline est mis en oeuvre. Cette figure illustre aussi le degré maximal des noeuds de données pour lequel la quantification Q_A serait suffisamment élevée pour éviter les dépassements. La courbe notée post-traitement correspond à la technique qui amène à une modification de l'information extrinsèque. Si on considère par exemple des codes dont le degré de connexion des noeuds de données est inférieur à 15, il sera plus intéressant de ne pas seuiller l'information extrinsèque ($Q_A = 9$) ou d'utiliser la méthode APP variable avec $Q_A = 8$, plutôt que d'utiliser un post-traitement amenant à une modification de l'information extrinsèque (sauf si $Q_A = Q_{m_{cv}}$). Dans le cas de notre algorithme, qui utilise la propriété du minimum, la technique de post-traitement de l'information extrinsèque par modification de l'information extrinsèque (technique du seuil) n'est pas adaptée

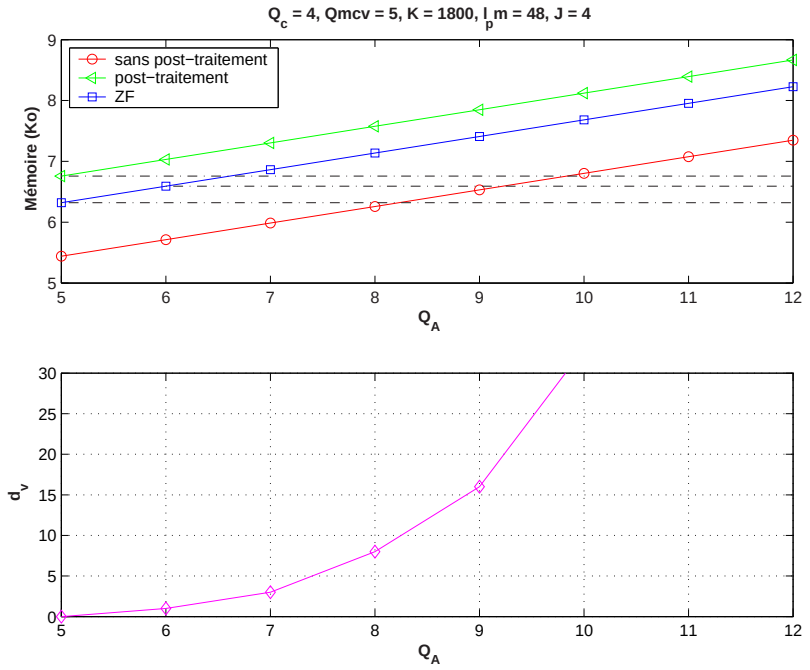


FIG. 4.12 – Variation de la mémoire totale du décodeur en fonction de la quantification de l'information *a posteriori*. ml_p représente la profondeur des buffers (l_p est un facteur qui dépend de la latence des processeurs).

au vue des conséquences sur l'augmentation de la mémoire. La technique du forçage à zéro et la méthode dite APP variable peuvent cependant avoir des bonnes propriétés.

Pour étudier l'effet du seuillage de l'information extrinsèque, nous considérons un décodeur où la quantification Q_A est fixée à 7, 6 et 5 bits. Les résultats sont illustrés sur la figure 4.13. Le degré de connexion maximum des noeuds de données du code étant de 6, le cas $Q_A = 8$ correspond à un cas où l'information *a posteriori* n'est pas seuillée. La technique qui consiste au seuillage de l'information *a posteriori* sans post traitement de l'information extrinsèque amène un comportement instable du décodeur quand $Q_A = 6$ et $Q_A = 5$. Dans le cas où $Q_A = 7$, ce seuillage fait apparaître un plancher d'erreur à un taux d'erreur binaire de 10^{-7}. Ce bon comportement peut s'expliquer par le fait que les noeuds de données majoritaires sont de degrés 3 et que, pour ces noeuds, il n'y a pas de seuillage de l'information *a posteriori* $(7 + 3 \times 15 = 52 < 2^6 - 1)$. A haut rapport signal à bruit, le fait de perdre de l'information sur les degrés de connexion forts ne permet pas de corriger certaines erreurs. Quand la technique du forçage à zéro est mise en oeuvre, les performances sont très bonnes pour $Q_A = 7$ et $Q_A = 6$. Dans le cas $Q_A = 5$ il y a apparition d'un plancher d'erreur à un taux d'erreur binaire d'environ 10^{-7}. Ce comportement peut paraître assez surprenant au vu des petites quantifications utilisées. Dans certain contexte cette solution sera bien adapté et apportera une complexité globale du décodeur inférieure à celle d'un décodeur utilisant la technique de seuillage sans post-

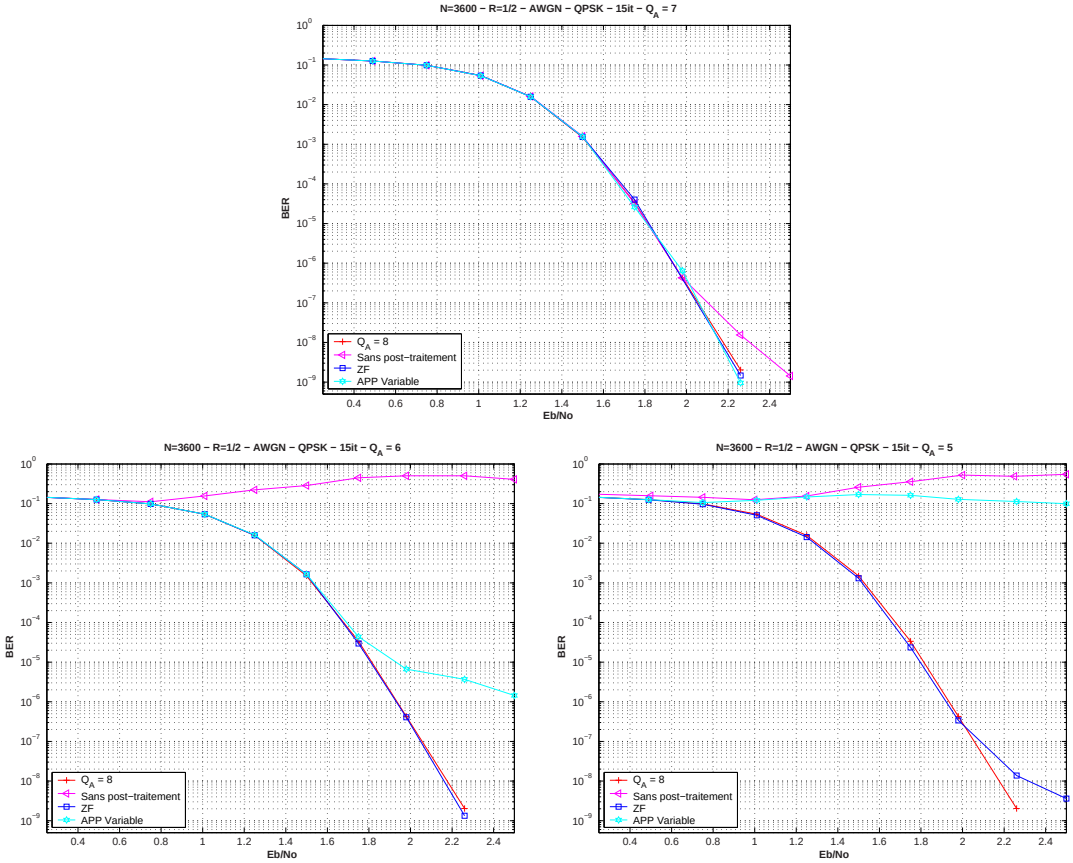


FIG. 4.13 – Illustration des performances d'un code irrégulier de taille $N = 3600$ pour différentes stratégie de quantification.

traitement. La solution de type APP variable sera très intéressante à condition que le seuillage ne soit pas trop fort. Cette solution permet par contre de réduire la mémoire nécessaire au décodeur.

Remarques :
Les comportements des différentes techniques à des taux d'erreur binaire très bas n'ont pas été vérifiés. Les techniques de seuillage s'accompagnant de pertes d'information, on peut présager un changement de comportement à fort rapport signal à bruit. L'analyse de ses propriétés est très difficile car les performances du code peuvent être limitées par sa distance minimale. Il faut aussi noter que l'analyse de quantification a été proposée sous l'hypothèse de données d'entrée suivant une loi gaussienne. Dans le cas de canaux radio-mobiles, certaines quantifications devront être réajustées pour permettre une meilleure prise en compte de la plage de données d'entrée, et ainsi éviter une forte concentration de données souples à zéro, qui correspondent pour le décodeur à des effacements.

Mémoire	A_v	m_{cv} (sans post-traitement ou APP variable)	m_{cv} (ZF)	Buffer	Processeur FBA	y_p
Profondeur	K	M	M	ml_p	ml_p	M
Taille des mots	Q_A	$J + \lceil \log_2 J \rceil + 2\,(Q_{m_{cv}} - 1)$	$2J + \lceil \log_2 J \rceil + 2\,(Q_{m_{cv}} - 1)$	$JQ_{m_{vc}} + 2\lceil \log_2 J \rceil$	$2Q_{m_{vc}} + Q_c$	Q_c
Type	Bancs	FIFO	FIFO	LIFO	LIFO	FIFO

TAB. 4.1 – Recapitulation de la taille des mémoires utilisées dans le décodeur proposé.

Dans cette sous section nous avons étudié les effets de la limitation de la dynamique des données. L'utilisation de la technique APP variable associée a une quantification des données d'entrée sur 4 bits semble être un compromis intéressant dès lors que le seuillage de l'information *a posteriori* n'est pas trop important. Cette sous-section a aussi montré l'importance du choix du degré maximum de connexion d'un noeud de données sur la complexité du décodeur.

4.2.4 Gestion de la mémoire

Cette sous-section a pour objectif de décrire la gestion des différentes mémoires. Dans un premier temps nous proposons de récapituler les profondeurs et les différents types de mémoires utilisées par le décodeur. Dans un second temps, l'architecture et le générateur d'adresses de certaines mémoires seront discutés.

Un tableau récapitulatif des différentes mémoires, du nombre et de la taille de mot utilisés est illustré à la table 4.1. Le buffer stockant les informations provenant du processeur SPC forward a un comportement qui peut se modéliser comme une LIFO. Dans le cas d'un facteur de parallélisme de p, ce buffer est dupliqué p fois. Les buffers utilisés dans le processeur traitant le treillis ont les mêmes comportements. Ils seront également dupliqués p fois dans le cas d'un facteur de parallélisme de p. La mémoire stockant les observations du canal de la partie redondante du mot de code a le comportement d'une FIFO. Dans le cas d'un facteur de parallélisme de p, cette mémoire est divisée en p[7]. Ces propriétés sont également valables pour la mémoire stockant l'information extrinsèque (les messages m_{cv}).

Le cas de la mémoire stockant l'information est plus particulier. Les processeurs de décodage traitent $p \times J$ données simultanément. Il faut donc être capable de lire l'ensemble de ces données en un nombre de cycles limité. L'utilisation de la structure de la matrice de contrôle de parité peut aider à résoudre ce problème. Dans un premier temps on considère un facteur de parallélisme p égal à 1. Le processeur SPC forward doit traiter J données provenant de la mémoire stockant l'information *a posteriori* (A_v). Par construction du code, ces J données n'appartiennent pas à un même groupe de z données comme illustré sur la figure 4.14.

Cette propriété nous permet donc de diviser la mémoire stockant l'information *a posteriori* en k mémoires de profondeur z. Cette organisation permet la lecture/écriture de J données simultanément. La génération des adresses de lecture et d'écriture est dans ce cas très simple. La position d'une matrice identité permutée

[7] la profondeur peut aussi être divisée par p et la longueur d'un mot multipliée par p

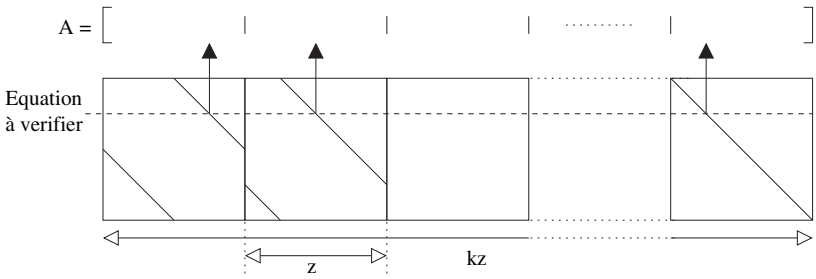


FIG. 4.14 – Illustration de l'organisation de la mémoire stockant l'information *a posteriori*.

dans une ligne correspond à l'indice du banc mémoire. La valeur de la permutation est utilisée pour la génération de l'adresse dans le banc considéré. Cette adresse est incrémentée modulo z à chaque lecture/ecriture. Dans le cas d'un parallélisme avec un facteur p, on peut utiliser les propriétés de la fonction modulo pour obtenir le même comportement.

Remarque : Le fait d'utiliser des matrices identité permutées permet la découpe de la mémoire stockant l'information *a posteriori* en k bancs de profondeur z. Ces bancs mémoires ont aussi la propriété de stocker l'information *a posteriori* associée à des noeuds de données ayant les mêmes degrés de connexion. La dynamique de ces valeurs varie en fonction du degré de connexion. Il est possible de concevoir un décodeur où les bancs mémoires auraient une taille différente dépendant de la caractéristique de l'information stockée. Si on considère une stratégie où le dimensionnement de la quantification Q_A est tel que les dépassements sont impossibles, les mémoires stockant l'information *a posteriori* de noeuds de données de faibles degrés peuvent être de taille inférieure à celles stockant les noeuds de forts degrés. A titre d'exemple les codes LDPC normalisés au sein de l'IEEE 802.11n ont au moins $10z$ noeuds de données, hors de la bi-diagonale, ayant un degré inférieur ou égal à 4, et cela quelque soit la taille et le rendement considéré. Ils ont également au maximum $3z$ noeuds de données de degré supérieur ou égal à 12 par code. Dans un tel cas, l'organisation de la mémoire peut prendre en compte ces paramètres. La mémoire peut donc non plus être dimensionnée pour le pire cas, mais en fonction des caractéristiques communes des codes.

4.2.5 Conclusion

Dans cette section nous avons décrit l'architecture et les paramètres de réalisation qui ont été choisis pour la réalisation d'un décodeur. Le choix de l'approximation utilisée a été dans un premier temps discuté. La description des processeurs de calcul a ensuite été détaillée. Cette description a montré la faible complexité calculatoire du décodeur. Cette propriété s'accompagne cependant d'un contrôle des données relativement important, tel que la détermination des indices, la reconstruction des signaux.

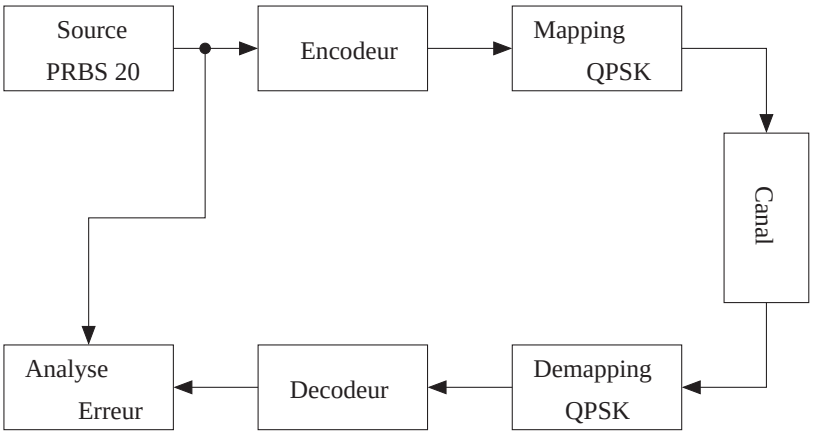


FIG. 4.15 – Chaîne de test mis en oeuvre sur le composant programmable.

Le problème relatif à la quantification nous a amené à analyser les effets de la limitation de la dynamique des chemins de données. Nous avons pu mettre en évidence l'importance du choix du degré maximal de connexion des noeuds de données dans la complexité du décodeur. Enfin, une brève description de l'organisation des mémoires a été proposée.

4.3 Résultats expérimentaux

Nous avons discuté dans la section précédente de la mise en oeuvre du décodeur. Nous proposons d'illustrer dans cette section quelques résultats de performances et de complexité obtenus une fois l'intégration sur composant programmable réalisée. Dans un premier temps, la chaîne de test sera présentée. Dans une second partie, la complexité du décodeur sera discutée à partir des résultats de synthèse sur une cible FPGA. Enfin , quelques comparaisons de performances seront présentées.

4.3.1 Mise en oeuvre de la chaîne de test

Dans le but de tester le décodeur proposé, une chaîne complète de simulation a été réalisée et intégrée sur un composant programmable. Cette chaîne est illustrée sur la figure 4.15. Elle est composée d'un générateur de séquence aléatoire binaire obtenue à partir d'un registre à décalage (PRBS : Pseudo Random Bit Sequences). Le générateur utilisé est réalisé à partir d'un registre à décalage composé de 20 étages dont la structure est présentée sur la figure 4.16. La séquence pseudo-aléatoire est traitée par le codeur dont la réalisation est très simple grâce à la structure bi-diagonale. Le codeur peut être être vu comme un code convolutif récursif à une bascule dont la sortie est poinçonnée. Le mot de code est alors dirigé vers l'organe de modulation qui construit les symboles à transmettre. Ces symboles sont alors traités par le canal.

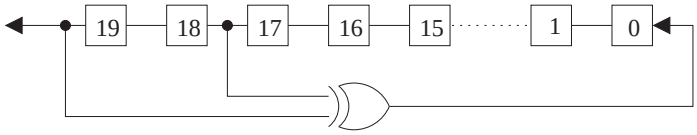


FIG. 4.16 – Registre à décalage générant la séquence pseudo-aléatoire.

Le canal considéré dans cette chaîne est un canal AWGN (Additive White Gaussian Noise) qui est intégré sur le composant à partir de la méthode proposée dans [106]. Cette technique consiste en la génération d'échantillons suivant une loi de probabilité normale centrée à partir d'une source de nombres aléatoires suivant une loi uniforme. Par l'utilisation conjointe de la méthode de Box-Muller [107] et du théorème de la limite centrale [(8)], la validité de la méthode a été démontrée [106]. Ce générateur d'échantillons peut également être mis en oeuvre pour la réalisation d'un canal de Rayleigh.

Les données en sortie du canal sont alors démodulées (et égalisées dans le cas d'un canal de Rayleigh), et re-quantifiées sur Q_c bits. Le décodeur traite alors ces données et génère une estimée des bits d'informations. Ces données sont alors comparées à la séquence initiale.

Pour permettre au décodeur de travailler au maximum de son débit, la mémoire stockant l'information *a posteriori* initialisée par les données du canal de la partie systématique et la FIFO mémorisant les observations du canal de la partie redondante du mot de code ont été doublées. Nous avons également considéré un décodeur avec un parallélisme de 2. Le décodeur est interfacé avec une mémoire ROM pouvant caractériser plusieurs paramètres de codes. Le schéma du décodeur intégré dans le composant programmable est illustré sur la figure 4.17.

L'ensemble de cette chaîne a été intégré dans un FPGA de la famille ALTERA STRATIX EP1S80F-C6. Ce composant est implanté sur une carte fille, comme illustré sur la figure 4.18, capable de communiquer via un bus PCI à une carte mère. Celle-ci est interfacée par un système d'exploitation temps réel (QNX). Une interface graphique a été développée de manière à automatiser les mesures de performances. Une copie d'écran de cette interface est illustrée sur la figure 4.19.

4.3.2 Considérations technologiques

Nous présentons dans cette section quelques résultats de complexité de notre décodeur. Celui ci a été entièrement décrit en VHDL (Very high speed integrated circuit Hardware Description Language) et synthétisé par l'outil QUARTUS de ALTERA. Ce décodeur est générique et peut être programmé par un fichier de paramètres

[(8)] ce qui consiste à accumuler les échantillons provenant de la transformation de Box-Muller

dont un exemple est décrit dans l'annexe C. Suivant le système considéré chaque paramètre peut être modifié et le décodeur peut alors être re-synthétisé. Il existe donc un grand nombre de configurations possibles. Dans le but d'illustrer quelque résultat de complexité, nous proposons d'étudier un cas équivalent à celui des codes normalisés dans le cadre de l'IEEE 802.11n et un cas plus exploratoire appliqué à une future version de télé-diffusion numérique terrestre (DVB-T2).

4.3.2.1 Application au contexte du type IEEE 802.11n

Les codes LDPC de la norme IEEE 802.11n sont des codes dont les matrices de contrôle de parité sont construites à partir de matrices identité permutées circulairement. La table 4.2 résume les différents paramètres qui ont été choisis pour cette norme. Il existe 4 rendements de codage différents associés à 3 tailles de mots de code.

Le but de cette section est de montrer comment les choix des paramètres du décodeur peuvent être réalisés pour correspondre aux paramètres d'un système. Pour que notre décodeur puisse traiter l'ensemble des rendements, la matrice équivalente doit être de rendement 1/2. Dans le cas où la taille du mot de code est fixée, il faut donc considerer le cas du rendement le plus fort. Dans le cas des codes LDPC de la norme IEEE 802.11n, le nombre de bits d'information correspondant au rendement le plus fort est égal à 1620. Nous fixerons les paramètres de notre décodeur de manière à ce qu'il puisse traiter un nombre de bits d'information de $2048 = 2^{11}$ bits. Nous fixerons également les paramètres k maximum à 16, et z maximum à 128. Ce choix nous permet donc de traiter un mot de code de taille allant jusqu'à 4096 bits pour un rendement 1/2 et 2035 bits pour un rendement 5/6. Contrairement au paramètre défini dans la norme IEEE 802.11n nous avons à chercher à nous rapprocher d'une puissance de 2.

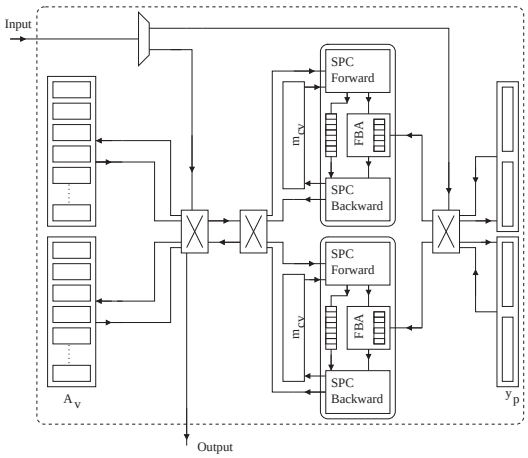


FIG. 4.17 – Illustration du décodeur intégré dans le composant programmable.

FIG. 4.18 – Illustration de la carte fille composée de deux FPGA STRATIX EP1S80F-C6 et d'un FPGA EP1S25F-C6. Une partie de ce dernier est utilisé pour la communication avec le bus PCI. La plate-forme de test dans un châssis de PC est aussi illustrée.

La table 4.3 présente les paramètres retenus pour la synthèse d'un décodeur. Nous avons considéré 4 cas. Le premier consiste à utiliser la méthode de seuillage ZF avec une quantification $Q_A = 6$. La seconde utilise la même technique de seuillage mais avec $Q_A = 7$. La troisième méthode utilise l'approche APP variable avec $Q_A = 7$. La dernière configuration est celle où l'on ne réalise pas de post traitement avec $Q_A = 8$. Ce décodeur associé à des codes dont le degré de connexion est inférieur à 8 garantit que l'information *a posteriori* ne sera pas écrêtée.

La table 4.4 présente les résultats de synthèse de ces différentes configurations. La complexité du décodeur est d'environ 7000 éléments logiques (Logic Cells ALTERA). Cette complexité varie en fonction de la stratégie adoptée pour la quantification. La mémoire requise par le décodeur est d'environ 11 Kilo octets. Ces chiffres peuvent paraître assez important mais il faut garder à l'esprit que la partie calculatoire de notre décodeur a été multipliée par 2 ($p = 2$). On peut noter une légère différence entre le décodeur autorisant le pipeline et celui ne l'autorisant pas. Cette différence provient, pour la mémoire, d'une taille supérieure des buffers et d'une complexité supérieure de leur gestion. Il faut aussi noter que ce décodeur peut traiter des blocs d'information jusqu'à 2048 bits avec un rendement maximum théorique de 7/8. Dans le cas des décodeurs présentés, la technique de poinçonnage telle que décrite dans [58] n'a pas été mise en oeuvre.

Il peut être aussi intéressant d'analyser la complexité des différents organes caractérisant le décodeur. La table 4.5 illustre en détail la complexité du décodeur pour la configuration 3. Cette décomposition permet de localiser les éléments du décodeur les plus couteux en terme d'éléments logiques et de mémoires. Tout d'abord, la stratégie d'une double mémoire d'entrée, telle que réalisée, pénalise fortement notre décodeur. En effet 40% de la complexité du décodeur provient des deux bancs mémoires utilisés pour le stockage de l'information *a posteriori*. On rappelle que cette mémoire est ini-

FIG. 4.19 – Illustration de l'interface graphique communicant utilisée pour l'automatisation des mesures.

tialisée par les observations du canal de la partie systématique du mot de code. Il a été choisi de dupliquer cette mémoire pour permettre le traitement continu de deux trames. Pendant le traitement d'une trame sur un banc mémoire, l'autre banc est vidé et initialisé par les données du canal. Le choix et la manière de dupliquer cette mémoire est donc sans doute à revoir de manière à minimiser cette complexité. La définition d'un organe de contrôle et de routage plus performant pour ces bancs mémoires est aussi à étudier. Nous rappelons que le décodeur réalisé n'est pas un décodeur optimisé. Celui-ci a été réalisé pour vérifier des concepts et permettre des mesures rapides. **Il est donc possible d'améliorer la complexité totale du décodeur en optimisant finement chaque fonction de manière à minimiser la complexité**.

Concernant les aspects de mémorisation, cette décomposition illustre la part importante de mémoire nécessaire pour le stockage de l'information extrinsèque (messages m_{cv}). On peut aussi noter la part importante de mémoire utilisée par l'organe de contrôle. Cette propriété est due aux mémoires ROM utilisées pour le stockage des codes. Celles-ci sont légèrement sur-dimensionnées pour permettre le stockage d'un maximum de codes à tester.

Un autre critère définissant la qualité de l'architecture est le débit atteint. La figure 4.20 illustre les bornes inférieures et supérieures des débits utiles en sortie du décodeur. Du fait de l'utilisation de fenêtres de petites tailles, le décodeur non pipeliné atteint des débits jusqu'à trois fois moins importants que le décodeur pipeliné. Il faut aussi noter que dans le cas d'un décodeur pipeliné, les contraintes sur les codes sont beaucoup plus fortes, ce qui peut rendre difficile la définition de codes de petites tailles (i.e. une restriction sur le choix des codes possibles). On peut noter que pour une fréquence d'horloge de 50MHz, le décodeur peut atteindre 10Mbit/s pour 10 itérations dans sa version pipelinée et au minimum 2.5Mbit/s dans sa version non

R	N	n	k	m	z
1/2	648 1296 1944	24	12	12	27 54 81
2/3	648 1296 1944	24	16	8	27 54 81
3/4	648 1296 1944	24	18	6	27 54 81
5/6	648 1296 1944	24	20	4	27 54 81

TAB. 4.2 – Paramètres des codes LDPC définis dans le cadre de la normalisation IEEE 802.11n

	J	p	Q_c	$Q_{m_{cv}}$	Q_A	Méthode de seuillage
Config 1	4	2	4	5	6	ZF
Config 2	4	2	4	5	7	ZF
Config 3	4	2	4	5	7	APP variable
Config 4	4	2	4	5	8	Seuil sans post-traitement

TAB. 4.3 – Paramètres de configuration du décodeur pour un contexte se rapprochant de celui défini dans le cadre de la normalisation IEEE 802.11n

pipelinée.

Nous avons illustré des résultats relatifs à l'architecture intégrée sur un FPGA des décodeurs proposés. Dans un contexte particulier, différentes stratégies ont été comparées. L'approche de type APP variable nous semble le meilleur compromis. Cette méthode permet à la fois de compresser l'information extrinsèque à mémoriser et de seuiller l'information *a posteriori*, le tout pour une complexité raisonnable. Quand au choix d'un décodeur pipeliné ou non, nos différentes manipulations ont pu montrer la difficulté de construire des codes de petites tailles respectant la règle de pipeline. Une approche hybride où le décodeur peut supporter les deux modes semble la meilleure solution. On peut noter que cette approche a été retenue dans [67] pour le réalisation d'un décodeur LDPC pour les codes de la norme IEEE 802.16e. L'architecture présentée, basée sur un algorithme de type *Layered BP*, supporte plusieurs modes de fonctionnement.

	Config 1	
	Logic cells	Mémoire (ko)
Pipeliné	6760	11.40
non-Pipeliné	6422	11.03

	Config 2	
	Logic cells	Mémoire (ko)
Pipeliné	7319	11.91
non-Pipeliné	6968	11.53

	Config 3	
	Logic cells	Mémoire (ko)
Pipeliné	7320	10.91
non-Pipeliné	6967	10.53

	Config 4	
	Logic cells	Mémoire (ko)
Pipeliné	7772	11.43
non-Pipeliné	7437	11.04

TAB. 4.4 – Résultats de synthèse de décodeurs dans un contexte se rapprochant de celui défini dans le cadre de la normalisation IEEE 802.11n

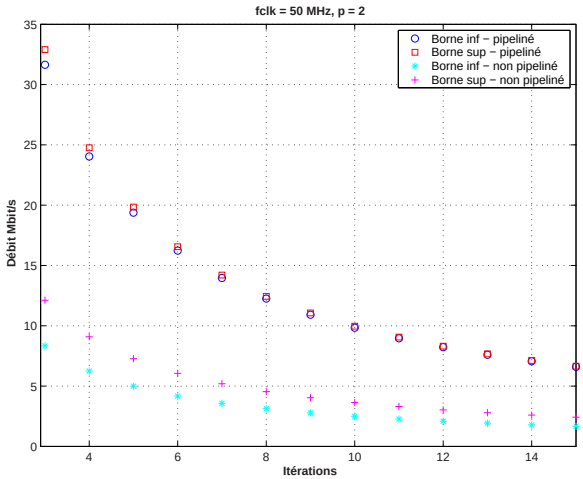


FIG. 4.20 – Variation du débit utile en sortie du décodeur en fonction du nombre d'itérations.

4.3.2.2 Application au contexte du type DVB-T2

Dans le cadre de la normalisation de la prochaine télé-diffusion numérique terrestre, il a été retenu certaines hypothèses sur la définition de la structure de codage. Des grandes tailles de blocs, de l'ordre de celles normalisées pour le standard DVB-S2, sont considérées. Dans ce contexte, nous proposons d'illustrer la complexité d'un décodeur sous l'hypothèse de grandes tailles de mots de code.

Nous nous focalisons sur des codes dont le nombre maximal de bits d'information est de l'ordre de 30000 bits. On considère également un rendement de codage minimum de 1/2. Les paramètres k maximum et z maximum seront respectivement fixés à 32 et 1024. Ce choix nous permet en théorie de traiter des rendements allant jusqu'à $16/17 \simeq 0.941$ et une taille de mots de code de 64k bits.

La table 4.6 illustre les résultats de synthèse d'un décodeur dans le contexte présenté. La configuration 3 de la table 4.3 a été sélectionnée. L'information *a poste-*

	Pipeliné				Non pipeliné			
	Logic cells		Mémoire (ko)		Logic cells		Mémoire (ko)	
TOTAL	**7320**		**10.914**		**6967**		**10.535**	
Contrôle	277	3.82%	-	-	274	3.94%	-	-
FIFO y_p 1	97	1.30%	1.000	9.16%	97	1.39%	1.000	9.49%
FIFO y_p 2	97	1.30%	1.000	9.16%	97	1.39%	1.000	9.49%
Mémoire A_v 1	1705	23.30%	1.615	14.80%	1705	24.47%	1.615	15.33%
Mémoire A_v 2	1461	19.96%	1.615	14.80%	1461	20.97%	1.615	15.33%
Coeur de décodage	**3683**	50.32%	**5.684**	52.08%	**3333**	47.84%	**5.305**	50.36%
FIFO Initialisation des fenêtres	54		0.125		54		0.125	
Contrôle	914		1.713		859		1.462	
Processeur de décodage 1	1297		1.923		1123		1.859	
Processeur de décodage 2	**1418**		**1.923**		**1297**		**1.859**	
Contrôle	50		-		20		-	
Multiplication par $\beta_f = 2^{-1} + 2^{-2}$	26		-		26		-	
Mémoire m_{cv}	-		1.750		-		1.750	
Buffers	219		0.136		212		0.070	
SPC Forward	246		-		246		-	
SPC Backward	357		-		357		-	
Processeur FB (Accumulateur)	520		0.037		436		0.037	
Processeur Forward	91		-		104		-	
Processeur Backward	290		-		168		-	
Buffers	104		0.037		150		0.037	

TAB. 4.5 – Complexité détaillé du décodeur de la configuration 3

	Logic cells	Mémoire (ko)
Non pipeliné	11377	147.57
Pipeliné	11793	148.53

TAB. 4.6 – Résultats de synthèse de décodeurs dans un contexte de type DVB-T2

riori est donc seuillée suivant la méthode APP variable. Les besoins en mémorisation sont très importants du fait de la grande taille des mots de code. La complexité du décodeur est de l'ordre de 11500 éléments logiques. Dans le cas de grandes tailles, il est très facile de construire des codes respectant la règle de pipeline pour le parallélisme considéré. C'est donc naturellement ce type de décodeur qui sera le plus adapté dans ce contexte.

Plusieurs campagnes de mesures ont été menées dans ce contexte. Nous avons en particulier construit un ensemble de codes dont la taille des mots de code est de l'ordre de 16K bits pour différents rendements. Les courbes de performances sont illustrées sur la figure 4.21. Le code de rendement 1/2 est un code irrégulier ayant des degrés de connexion 2, 3 et 6. Le code de rendement 2/3 a des degrés equivalents. Pour les codes de rendements supérieurs, le degré de connexion des noeuds de données correspondant à la partie systématique du mot de code est fixé à 4. L'ensemble de ces codes ont été construits en utilisant l'algorithme de construction incrémentale présenté dans le chapitre 2. Les performances ont été obtenues avec 15 itérations de décodage. Le but de ces mesures n'a pas été de déterminer les meilleurs codes possibles en terme de seuil de convergence, mais pour démontrer la stabilité de notre décodeur dans les

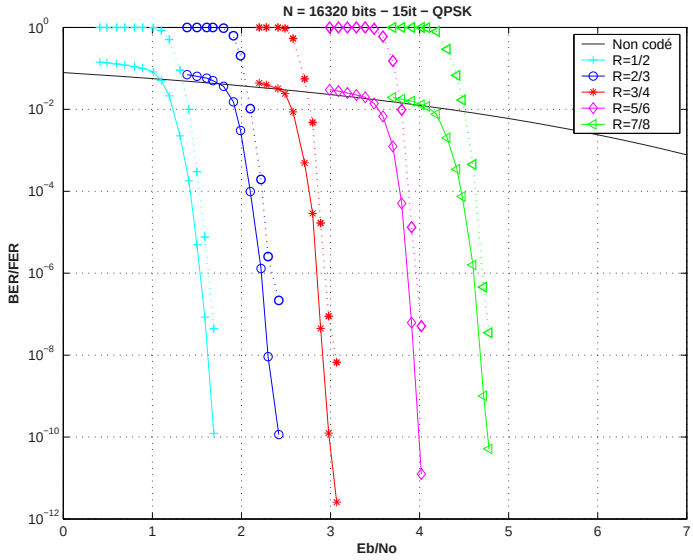


FIG. 4.21 – Illustration de performances de codes de taille $N = 16320$ bits avec des rendements de 1/2, 2/3, 3/4, 5/6 et 7/8. Le taux d'erreur binaire correspond aux courbes en trait plein, le taux d'erreur paquet aux courbes en pointillé.

Notations	N	z	m	Distribution des degrés	Seuil (dB)
Code 1	60000	1000	30	$\tilde{\lambda}(x) \simeq \frac{1}{6}x^5 + \frac{1}{3}x^3 + \frac{1}{2}x^2$	0.626
Code 2	60000	1000	30	$\tilde{\lambda}(x) \simeq \frac{1}{12}x^8 + \frac{5}{12}x^3 + \frac{1}{2}x^2$	0.711

TAB. 4.7 – Distribution des degrés des codes de tailles $N = 60000$. L'unique noeud de données de degré 1 a été volontairement oublié pour plus de clarté dans l'écriture des distributions. La distribution des noeuds de contrôle est dans les deux cas égale à $\tilde{\rho} \simeq x^5$.

zones de faibles taux d'erreur. Ces mesures ont aussi confirmé le fait qu'il est possible de construire des codes LDPC ayant de très bonnes distances minimales. La comparaison avec les codes LDPC de DVB-S2 (short frame) est assez difficile à réaliser. Tout d'abord les rendements des codes LDPC seuls ne concordent pas avec ceux choisis. Le rendement le plus proche de $R = 1/2$ est égal à $R = 4/9$. Nos codes (non optimisés) ont cependant des performances légèrement moins bonnes de quelques dixièmes de dB.

Le cas de codes de tailles supérieures a aussi été exploré. Nous avons en particulier étudié des codes de rendement 1/2 avec deux profils d'irrégularité différents. Les paramètres des codes étudiés sont illustrés à la table 4.7. Les simulations de performances de ces codes sont illustrées sur la figure 4.22. Les performances du code 1 à 1.4dB sont à interpréter avec précaution. Ce dernier point a été simulé durant plus de 14 jours avec un débit de 6.667Mbit/s. Durant cette période de mesure, seules 6 erreurs sont apparues. Dans le cas des autres mesures, le critère d'arrêt a été fixé a 50 paquets erronés. Les performances obtenues sont distantes de moins de 1.2dB de

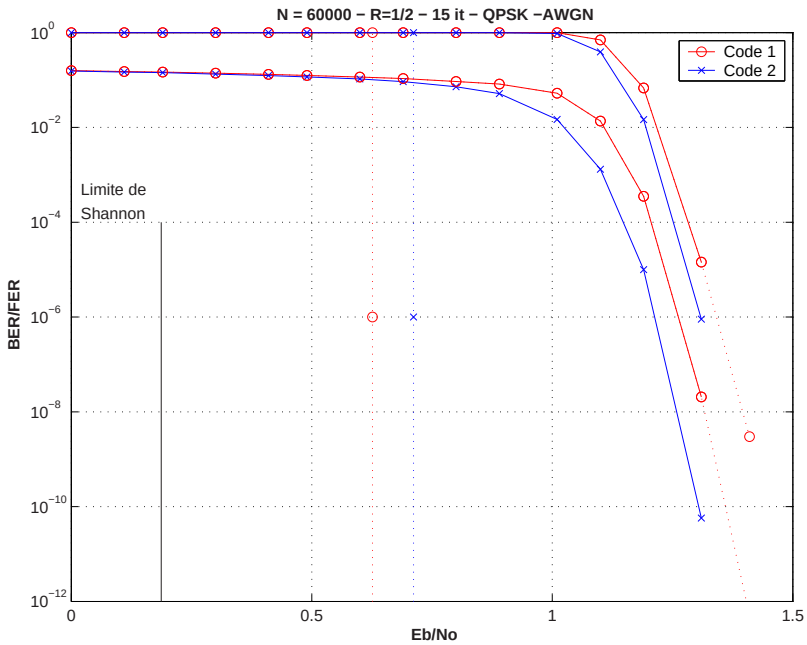


FIG. 4.22 – Illustration de performances de codes de taille $N = 60000$ pour une distribution avec des noeuds de degrés 6 (Code 1) et des degrés 9 (Code 2). La limite de Shannon ainsi que les seuils de convergences calculés sont aussi illustrés.

la limite de Shannon pour un décodage avec 15 itérations. Les deux codes simulés ne font pas apparaître de plancher d'erreur visibles. Ces simulations montrent également un avantage pour le code 2 de moins de 0.05dB alors que le calcul du seuil prédisait un avantage pour le code 1 d'environ 0.08dB. Ces résultats nous conduisent à formuler plusieurs remarques. Premièrement, le seuil a été calculé en prenant pour hypothèse un algorithme BP avec un ordonnancement par inondation pour une taille de code et un nombre d'itérations infini. Dans notre cas, aucune de ces hypothèses ne sont respectées, ce qui peut expliquer une telle différence. Enfin, pour permettre une optimisation plus fine des codes, une méthode prenant en compte les spécificités de notre séquencement doit être mise en place. Un bon point de départ se trouve dans [55] où il a été montré des différences dans les résultats d'optimisation de codes en fonction de la méthode et de l'algorithme utilisé.

La comparaison avec le code de taille $N = 64800$ de la norme DVB-S2 n'est pas simple à mettre en oeuvre, n'ayant pas accès à toutes les informations concernant les décodeurs publiés ou disponibles sur le marché (débit, quantifications, complexité). Nous proposons néanmoins une comparaison illustrée sur la figure 4.23. Les simulations ont été réalisées avec une précision en virgule flottante et un algorithme BP avec ordonnancement par inondation. Le but de ces simulations est de comparer les performances intrinsèques des codes. Le code du DVB-S2 est un code irrégulier ayant des noeuds de données de degré 1,2,3 et 8, et un degré moyen de connexion des noeuds

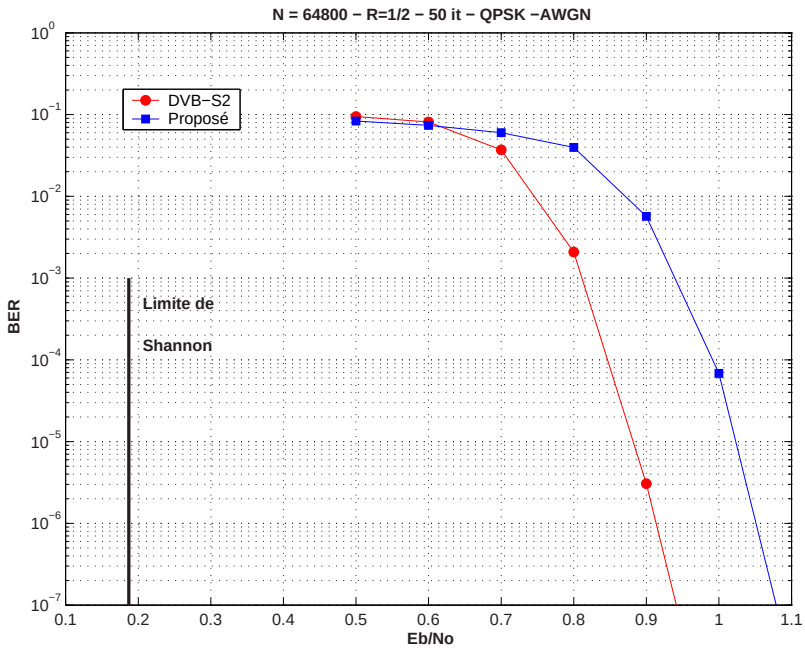


FIG. 4.23 – Illustration des performances du code LDPC DVB-S2 de taille $N = 64800$ et de rendement $R = 1/2$ et comparaison avec un code proposé dont la matrice de contrôle de parité est de densité inférieure. Le décodage a été réalisé en virgule flottante avec 50 itérations de décodage BP.

de contrôle de 6.99. En comparaison, le code proposé a un degré moyen des noeuds de contrôle de 5.99 (Code 1 de la table 4.7). Cette différence a pour conséquence une matrice de densité inférieure (En moyenne, les équations de parité ont un élément en plus). Le décodage du code proposé nécessitera donc moins d'opérations et sera moins complexe. Cette différence se voit également sur les performances du code : le code du DVB-S2 a des performances meilleures de 0.12dB. Bien sûr, ce gain est à contre-balancer avec la complexité supplémentaire nécessaire. Comme dans toutes les études illustrées, il s'agit encore de discuter du compromis entre performance et complexité.

4.3.3 Exemple d'application

4.3.3.1 Comparaison avec un Turbo-code

Dans le but de comparer notre architecture à celle d'un Turbo-code nous avons intégré un décodeur turbo duo-binaire 8 états développé par la société Turbo Concept (2X - TC1000-xX DVB-RCS)[108]. Les entrées de ce décodeur sont quantifiées sur 4 bits. Le décodeur peut traiter une taille de mot d'information inférieure ou égale à $K = 256$ octets (2048 bits) et non multiple de 7[9]. L'entrelaceur utilisé est du

[9] Cette contrainte est liée au fait que les treillis sont circulaires.

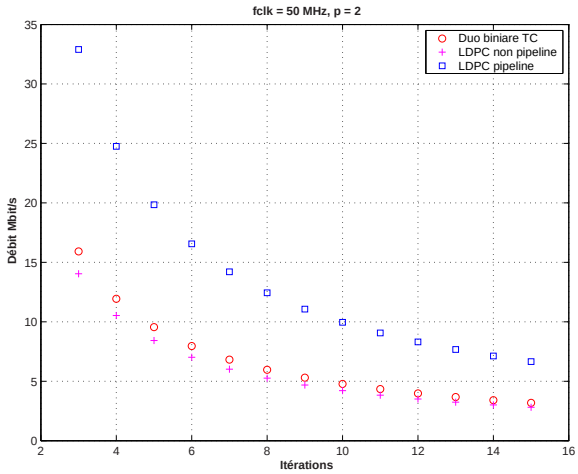


FIG. 4.24 – Variation du débit utile en sortie des décodeurs en fonction du nombre d'itérations. Les débits illustrés sont les débits maximum (Dans le cas du décodeur TC, on considère la plus grande taille ce qui minimise l'effet du paramètre ϵ).

type ARP (Almost Regular Permutation) [109]. Le débit utile de l'architecture 2x est donné par la relation :

$$D = \frac{pK}{2\,(K+\epsilon)\,it} f_{clk} \tag{4.10}$$

où ϵ est un paramètre proportionnel à la latence introduite par le décodage d'une fenêtre. La figure 4.24 illustre les débits utiles pour le décodeur 2x ($p = 2$) à partir des données fournies par le concepteur (paramètre ϵ). A partir de l'expression du débit utile, on peut remarquer que les deux treillis du code sont décodés séquentiellement : le traitement d'un treillis commence une fois que l'autre est complètement réalisé. Il est possible d'améliorer cet ordonnancement si des contraintes sont appliquées à l'entrelaceur (cf [63, 110]). Il faut noter qu'un couple décodeur/entrelaceur optimisé pour permettre l'utilisation continue de la ressource aurait un débit légèrement supérieur à celui d'une version non pipelinée . Si un parallélisme 4x est mis en oeuvre, les débits seraient équivalents[10] a ceux de notre décodeur pipeliné.

La comparaison des débits utiles ne peut se faire qu'avec la connaissance de la complexité du décodeur. La table 4.8 présente les résultats de complexité du décodeur intégré dans le même environnement que le décodeur de type LDPC étudié. Ces résultats sont à analyser avec beaucoup de précautions. Si l'on compare ces chiffres de complexité avec ceux de la table 4.5, qui correspondent à un contexte équivalent, le turbo décodeur est moins complexe que le décodeur proposé. Cependant l'architecture proposée est moins mature que celle du turbo décodeur. De plus, certains de nos choix de réalisation sont discutables. Le fait de dupliquer deux fois la mémoire

[10] Cette différence provient du fait que le décodeur duo-binaire traite des couples de données. Il a donc un parallélisme naturel de 2.

stockant l'information *a posteriori* nous pénalise d'environ 1600 éléments logiques. Ce chiffre correspond à l'écart constaté entre la complexité de notre décodeur et celui du turbo décodeur (cf table 4.5)[11]. Cependant, notre décodeur est incontestablement plus coûteux en terme de mémorisation. Alors que le décodeur turbo consomme 8K octets, notre solution nécessite plus de 10K octets de mémoire (9.3K octets si la mémoire stockant l'information *a posteriori* n'est pas dupliquée). Le fait d'utiliser une approximation du minimum combinée avec le traitement de la matrice bi-diagonale sous forme d'un treillis décodé par fenêtre permet une réduction non négligeable des ressources de mémorisation par rapport à un algorithme de propagation de croyance classique.

Pour rendre compétitif notre décodeur, il serait aussi nécessaire d'étudier une caractérisation moins coûteuse en terme de mémorisation des codes. Dans le cas du turbo décodeur, seulement 4 paramètres suffisent pour la définition d'un entrelaceur. Dans notre cas, il est nécessaire de mémoriser mJ coefficients de permutation et mJ positions (correspondant aux indices des positions des coefficients de permutation dans une ligne de la matrice de base). Même si dans certains cas des techniques telles que celles utilisées dans la norme IEEE 802.16e peuvent être mises en oeuvre (caractérisation d'un code pour un rendement et calcul des coefficients de permutation des codes de longueurs différentes suivant une règle), ce point reste un désavantage pour les codes LDPC.

Concernant les performances, plusieurs contextes ont été simulés. Dans le cadre de ce manuscrit nous proposons d'illustrer des performances obtenues dans le contexte du projet européen OPUS [111]. Ce projet a pour but de développer une plate-forme expérimentale pour les évolutions futures de l'UMTS. Le nombre de bits d'informations est fixé à $K = 1800$ bits. Le code LDPC et le Turbo-code duo-binaires 8 états ont été optimisés pour un rendement $R = 1/2$. A partir de ces rendements initiaux, des codes de rendements supérieurs (2/3, 3/4 et 5/6) sont obtenus par simple poinçonnage régulier des bits de redondances. Les codes de rendements supérieurs à 1/2 ne sont donc pas optimisés. Deux codes LDPC ont été simulés. Le premier code (noté code 1) a été conçu dans le but d'obtenir une grande distance minimale sous la contrainte d'un degré de connexion des noeuds de contrôle de 6. Ce code comprend donc des noeuds de données de degrés 2 et 4. Le second code (noté code 2) a été construit pour obtenir de bonnes performances dans la région du *waterfall*, sous la contrainte d'un

[11] Le décodeur turbo a aussi un double buffer d'entrée. Ce buffer est cependant *a priori* beaucoup moins complexe que les bancs mémoires considérés dans notre décodeur

	Logic cells	Mémoire (ko)
TC1000 2x	5503	8.147

TAB. 4.8 – Résultats de complexités du décodeur turbo intégré sur le même composant et la même chaîne que le décodeur LDPC.

degré de connexion des noeuds de contrôle de 6. Ce code est un code irrégulier dont le graphe contient des noeuds de données de degrés 2,3 et 6. Le Turbo-code est décodé avec 10 itérations ($\simeq$5Mbit/s@50MHz) contre 20 itérations ($\simeq$5Mbit/s@50MHz) pour les codes LDPC. Le but est d'obtenir dans les deux cas les meilleurs performances possibles. Dans le cas du code LDPC, la différence de performances entre 20 et 15 itérations est inférieure au dixième de dB. De la même manière, les performances sont quasi-identiques pour le Turbo-code de cette taille décodé avec 8 et 10 itérations.

Les performances sont illustrées sur les figures 4.25, 4.26, 4.27, 4.28. Les simulations montrent dans tous les cas un avantage inférieur à 0.3-0.4dB pour les Turbo-codes duo-binaire 8 états dans la région du *waterfall*. Les codes LDPC simulés ont par contre un avantage à des taux d'erreur très bas. Cet avantage est dû à une meilleure distance minimale qui se traduit par un plancher d'erreur plus bas. Les performances du code 1 sont dégradées par rapport à celui du code 2 dans la région du *waterfall*. Le code 1 a cependant un très bon comportement à des taux d'erreur très bas.

En conclusion, la comparaison entre les codes LDPC étudiés et les Turbo-codes duo-binaire est très délicate à réaliser. En terme de complexité, le décodeur duo-binaire est moins complexe que le décodeur proposé. Cependant, comme nous l'avons déjà mentionné, notre décodeur n'est pas optimisé pour minimiser la complexité. On peut donc présager que la complexité d'un décodeur LDPC soit équivalente en terme d'éléments logiques mais supérieure en terme de mémoire. Nous avons pu aussi démontrer que la flexibilité requise par un standard en terme de rendements de codage était tout a fait à la portée du décodeur tel que défini. Bien sur, cette flexibilité est moins évidente que celle des Turbo-codes mais très simple à mettre en oeuvre. Du coté des performances, la comparaison est également difficile. Incontestablement, les Turbo-codes duo-binaires sont l'une des meilleurs familles de codes pour le seuil de convergence quand les tailles sont de l'ordre du kilo bit. Une des conséquences de cette propriété est l'apparition d'un léger plancher d'erreur. A des taux d'erreur binaire très bas, le code LDPC sera meilleur que le Turbo-code seul. En résumé, il n'existe pas une technique de codage optimale. Bien sûr, chaque point mentionné peut être discuté et débattu en fonction d'un contexte ou d'une application particulière.

4.3.3.2 Analyse du comportement des codes

La définition d'un décodeur matériel permet l'étude de phénomènes intervenant à des taux d'erreur très bas. Dans le chapitre 2, nous avons mis en évidence l'influence des pseudo-mots de codes sur le plancher d'erreur du code. La validation des concepts et la définition des algorithmes ont été réalisés grâce au décodeur intégré sur le FPGA. En effet, il est possible d'intégrer dans le composant FPGA, en plus du décodeur, un analyseur logique. Cet analyseur nous a permis de détecter, à haut rapport signal à bruit, la séquence d'entrée appliquée au décodeur qui le fait diverger. L'analyse de cette séquence dans un décodeur programmé en virgule flottante nous a permis de déduire le type de configuration faisant diverger le décodeur (mot de code à

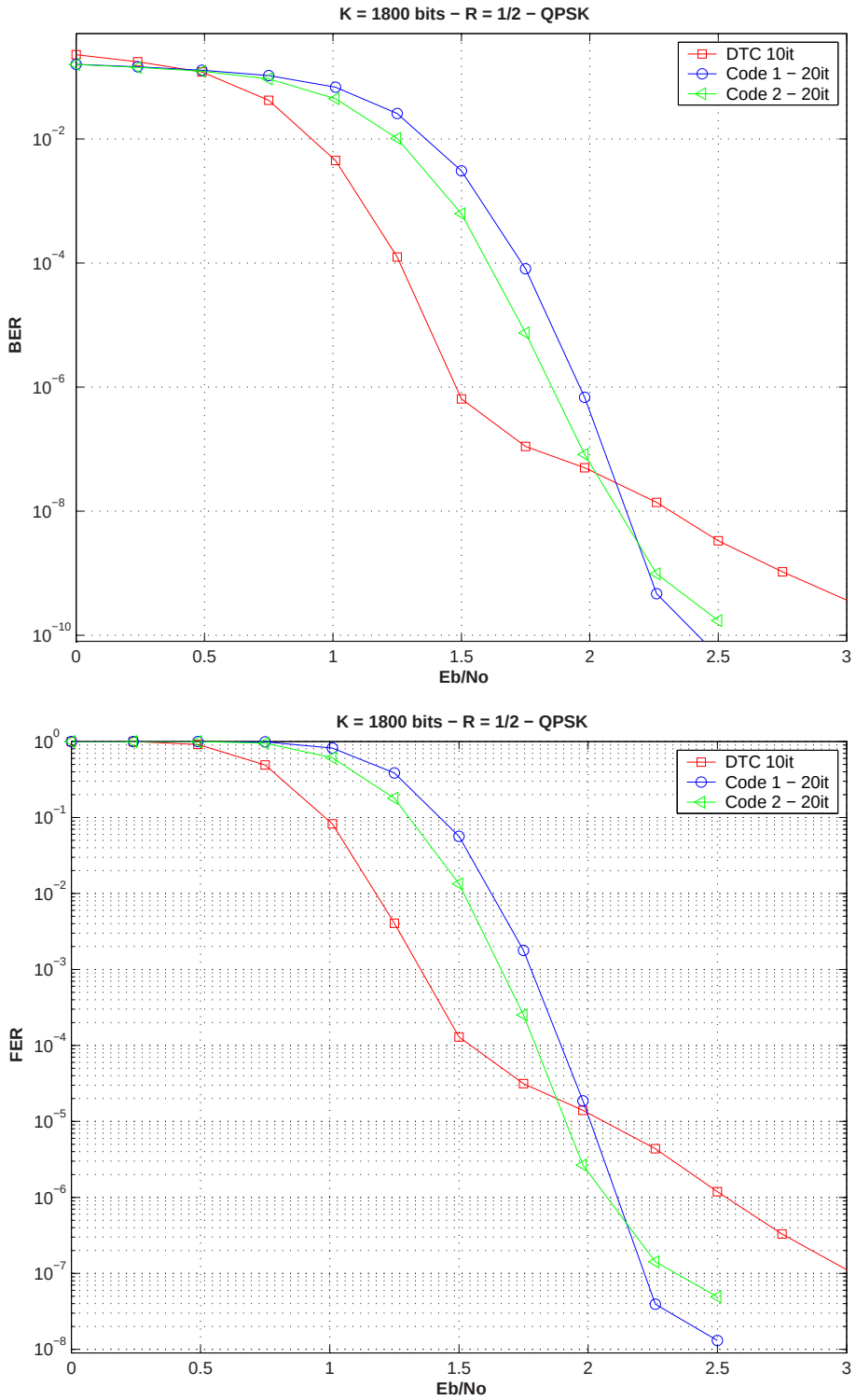


FIG. 4.25 – Illustration de performances de codes LDPC et Turbo-code duo-binaire 8 états de taille $K = 1800$ pour un rendement de codage de 1/2.

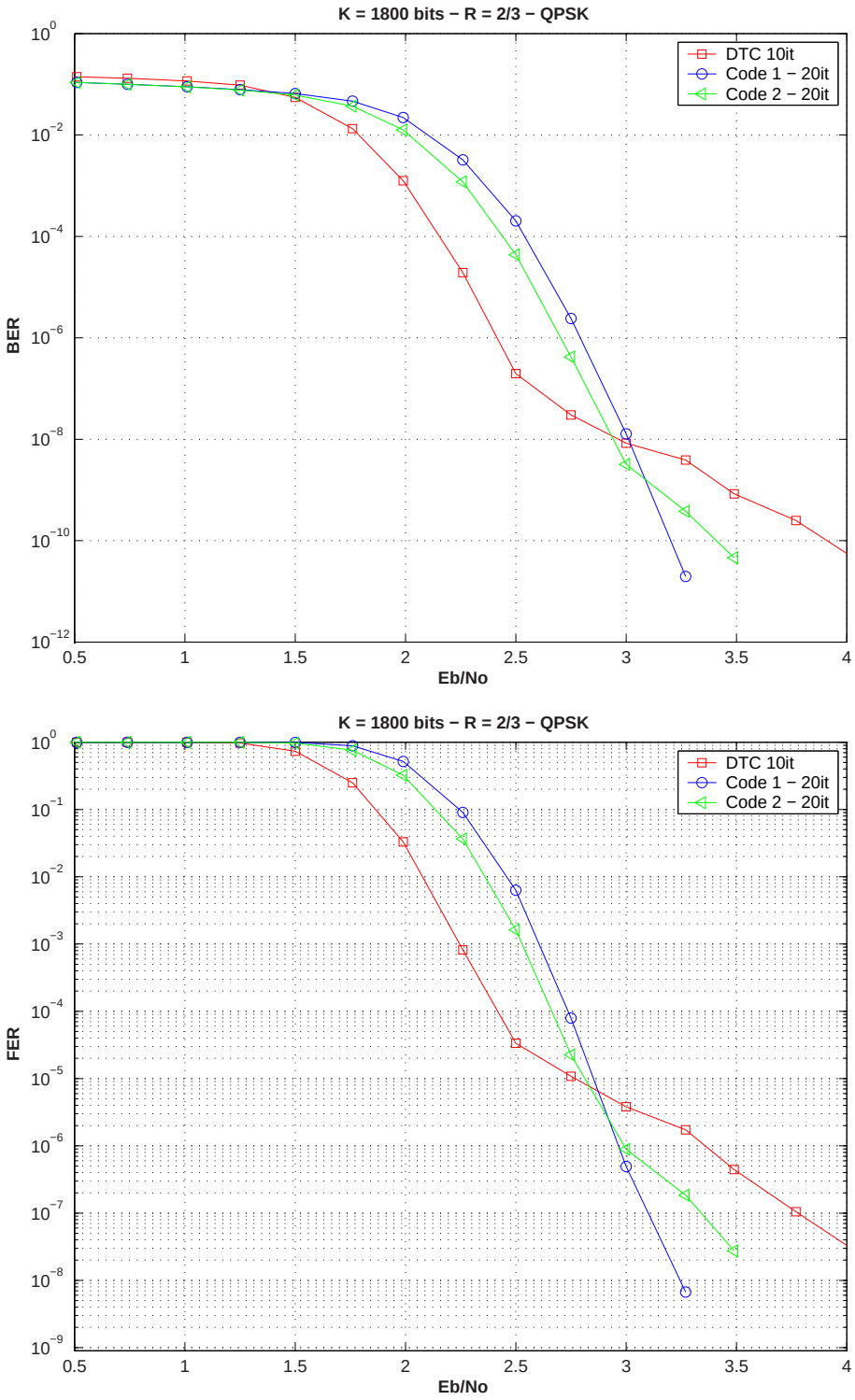


FIG. 4.26 – Illustration de performances de codes LDPC et Turbo-code duo-binaire 8 états de taille $K = 1800$ pour un rendement de codage de 2/3.

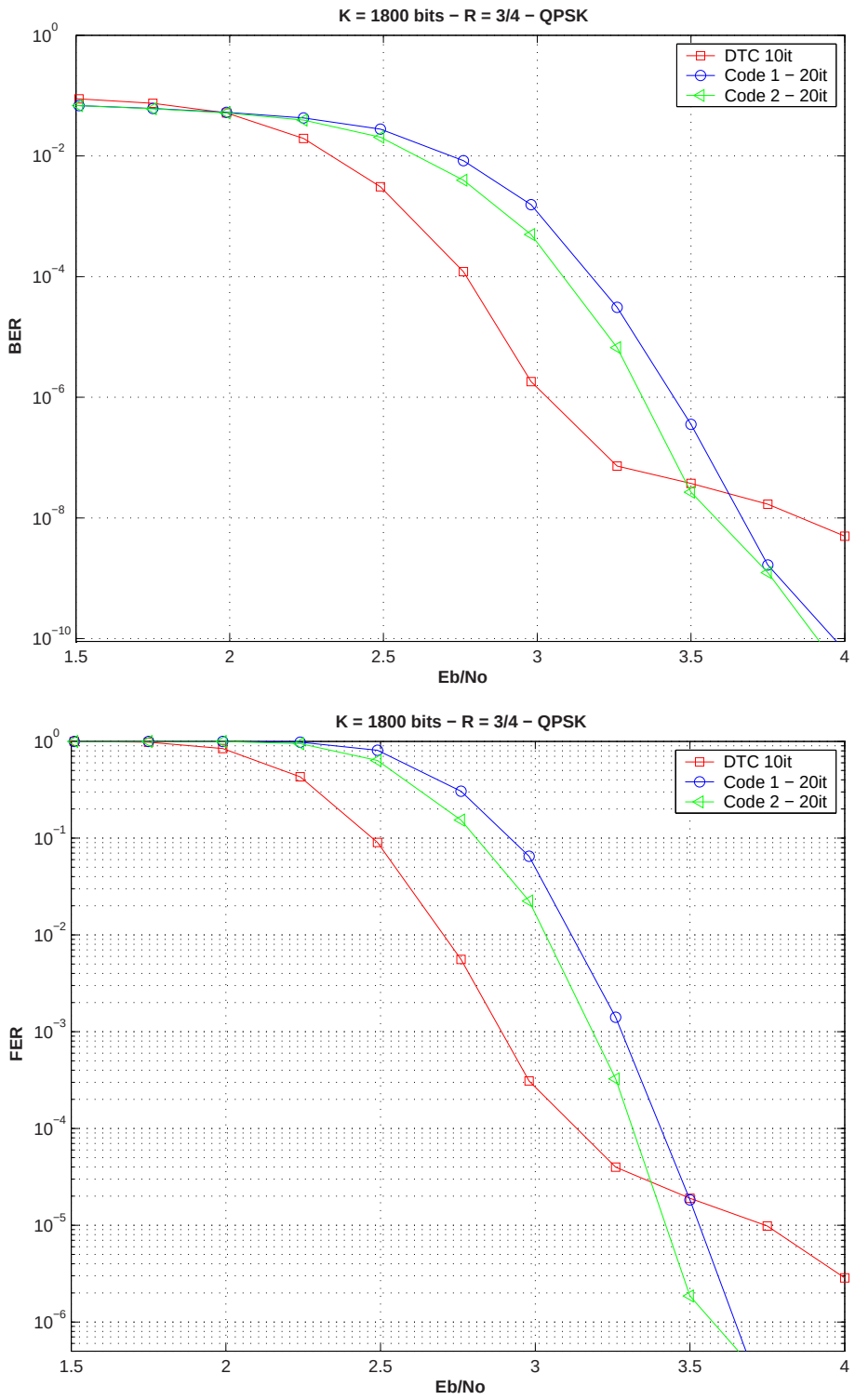


FIG. 4.27 – Illustration de performances de codes LDPC et Turbo-code duo-binaire 8 états de taille $K = 1800$ pour un rendement de codage de 3/4.

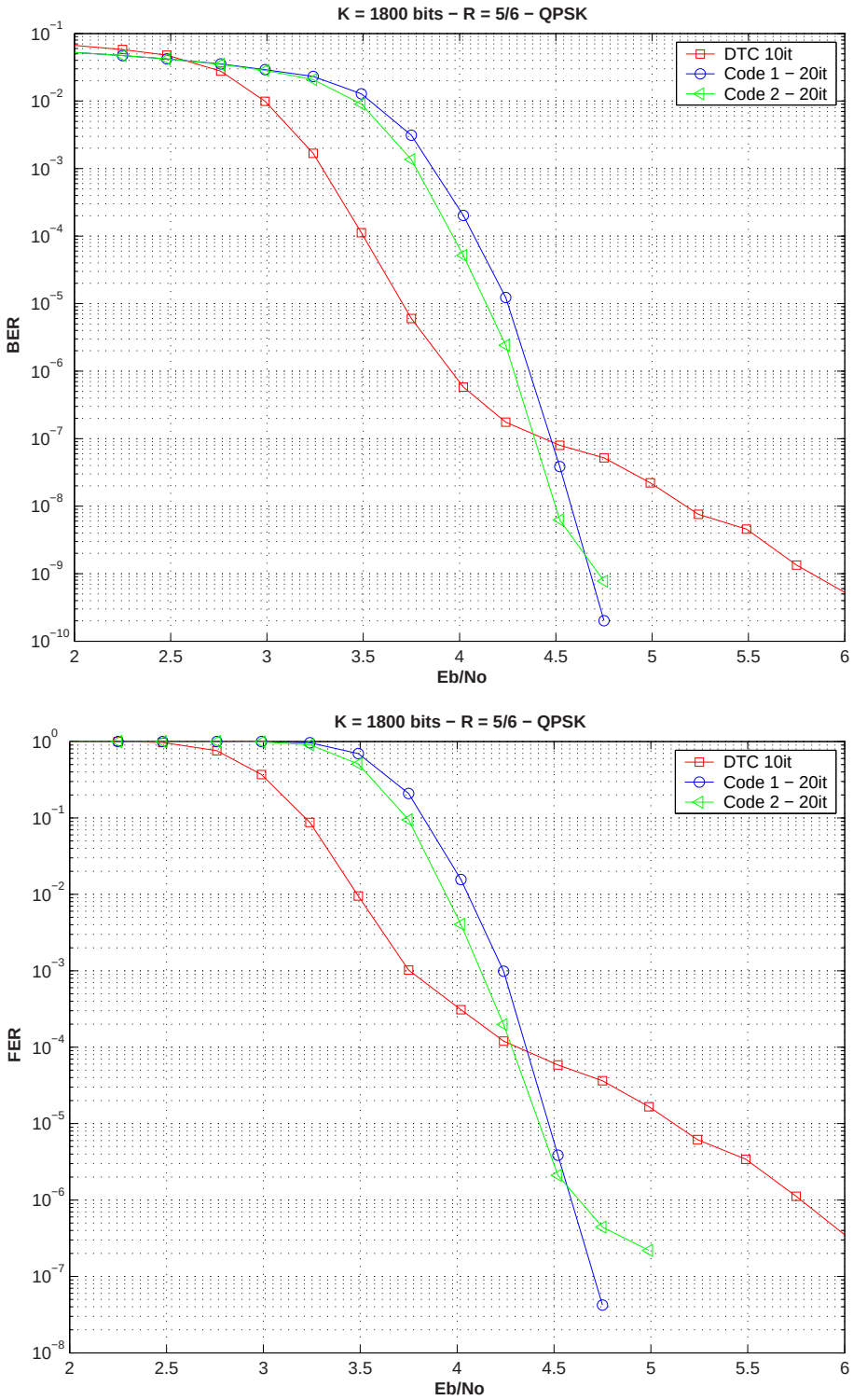


FIG. 4.28 – Illustration de performances de codes LDPC et Turbo-code duo-binaire 8 états de taille $K = 1800$ pour un rendement de codage de 5/6.

la distance minimale ou configurations pièges). L'estimation de la distance minimale du code est alors possible. Ce travail nous a permis de définir les méthodologies et les algorithmes de choix de codes présentés dans le chapitre 2. Il faut noter que ce type d'approche a aussi été exploré dans [112] pour la détection de configurations de trapping sets dans le cadre de codes QC LDPC.

4.4 Conclusion

Ce dernier chapitre a principalement été consacré à la description de l'architecture matérielle du décodeur qui a été mise en oeuvre dans le composant programmable. Après avoir décrit les simplifications algorithmiques utilisées, la stratégie de quantification a été discutée. Différentes méthodes ayant pour but de réduire la dynamique des données internes ont été illustrées et analysées. Il est apparu que la technique qui consiste à utiliser la propriété de saturation après l'avoir détectée est relativement performante tant au point de vue des performances que de la complexité.

Sur la base de ces résultats, nous avons par la suite illustré des résultats relatifs à l'intégration du décodeur sur un composant programmable de type FPGA. Les mesures de performances dans différents contextes ont été décrites. Il a été montré la validité des paramètres choisis sur la stabilité du décodeur. Nos mesures n'ont pas montré l'apparition de planchers d'erreur visibles du fait des paramètres retenus pour la réalisation. La flexibilité du décodeur en terme de rendements de codage a été démontrée.

Enfin, une comparaison avec un décodeur turbo duo-binaire 8 états a été présentée. Malgré la difficulté de comparaison, il a été démontré la quasi-équivalence entre les deux systèmes de codage en termes de complexité et de performances avec un léger avantage au décodeur turbo dû à sa maturité. Ces résultats nous ont conforté dans le choix du séquencement et de l'architecture proposée.

Conclusions et Perspectives

Cette thèse est consacrée à l'étude et à la réalisation matérielle de décodeurs pour les codes LDPC. De part leurs bonnes performances, leur simplicité de caractérisation et d'encodage, nous nous sommes naturellement intéressés aux codes LDPC structurés, et plus particulièrement à la famille des codes Repeat Accumulate. Nous montrons comment, par une défintion conjointe du code et de l'architecture de décodage, il est possible d'obtenir des décodeurs flexibles et performants. Ce travail constitue donc une approche originale de définition de structures de codes et d'architectures de décodage pour les codes LDPC structurés.

Hormis la mise en évidence d'un lien fort entre certains aspects relatifs aux codes LDPC structurés et aux codes convolutifs, qui est un fil conducteur de notre travail, le **chapitre 1** de cette thèse est principalement consacré à la présentation des codes LDPC. Un état de l'art relatif à la construction des codes, les méthodes d'encodage et de décodage est succinctement présenté. Ce premier chapitre aborde également les problématiques de réalisation par la description d'architectures de décodage couramment décrites dans la littérature. Des concepts de base nécessaires à la bonne compréhension du manuscrit sont donc présentés.

Le **chapitre 2** constitue une étape importante de cette thèse puisqu'il introduit la famille de codes considérée. Après la description des raisons qui nous ont motivés à considérer la famille des codes de type Repeat Accumulate, une analyse des propriétés de ces codes est proposée. Cette étude originale a permis de mettre en évidence l'importance du choix de certains paramètres sur les performances du code. Le choix du nombre de sous-matrices par colonne de la matrice de contrôle de parité, ainsi que la distribution des degrés des noeuds de données sont des paramètres qui peuvent limiter la distance minimale du code. Dans la même lignée, la description d'outils pour la détection des cycles dans le graphe contraint de la structure de codage étudiée a permis la définition d'un algorithme original de construction de codes. Cet algorithme travaillant sur un graphe de petite taille représentant le graphe de base (ou protographe), choisit incrémentalement les coefficients de permutations à appliquer aux matrices identité permutées caractérisant la matrice de contrôle de parité du code selon des contraintes données. Au travers des exemples expérimentaux, nous avons montré comment cet algorithme peut être paramétré. Ce chapitre introduit également des ordonnancements de décodage conçus en adéquation avec la définition

des codes. En particulier, nous avons défini le séquencement *Turbo Layered BP*, qui traite le décodage comme celui d'une concaténation série d'un code LDGM et d'un code convolutif récursif à deux états. Le décodage est réalisé suivant un séquencement *shuffle* pour obtenir la convergence la plus rapide possible.

Les apports majeurs de notre travail sont présentés au **chapitre 3**. Dans un premier temps, les architectures classiquement mises en oeuvre dans le cadre d'un algorithme de décodage de type *Layered BP* sont présentées. Une modélisation de différentes architectures de processeurs de résolution d'équations de parité est proposée. Cette représentation, issue de celle communément utilisée dans le cadre du décodage des codes représentés par des treillis, permet rapidement de déterminer les spécificités et les contraintes des architectures. Une première série de règles de construction de codes est illustrée dans le cas d'une architecture pipelinée où le degré de parallélisme est égal à la taille d'une sous-matrice. Le cas d'une architecture semi-parallèle est également étudiée. Ce cas est particulièrement intéressant quand on utilise un nombre de processeurs de résolution d'équations de parité inférieur à la taille d'une sous-matrice identité permutée caractérisant la matrice de contrôle de parité du code. En particulier, des règles de construction de codes sont énoncées pour permettre un ordonnancement de décodage autorisant des opérations pipelinées se chevauchant dans le temps. La seconde partie de ce chapitre décrit différentes architectures pour le séquencement *Turbo Layered BP* présenté au chapitre 2. Pour chaque architecture, des règles de construction doivent être imposées sur les codes pour garantir la convergence et, dans certains cas, l'accès à une donnée dans une mémoire une seule fois par cycle. Les règles décrites peuvent être introduites dans l'algorithme de construction de codes présenté au chapitre 2. La structure des processeurs de calcul utilisés dans le cadre de ces ordonnancements est une structure semi-parallèle, où un processeur traite un certain nombre d'éléments d'une équation de parité en un cycle. Ce niveau de parallélisme introduit ne permet pas à première vue une architecture flexible et générique. Une méthode originale basée sur les propriétés de la matrice de contrôle de parité caractérisant un mot de code poinçonné est donc introduite pour répondre à la problématique de flexibilité (toujours dans le cas des codes étudiés). La mise en place de cette technique permet d'obtenir une granularité suffisante pour adresser les niveaux de flexibilité requis par les systèmes modernes de communications. Cette technique est très peu complexe dans le cas d'un séquencement de type *Turbo BP* grâce notamment à la définition du code à travers un treillis. Dans une dernière partie, des architectures hybrides originales sont présentées. L'ensemble des résultats relatifs à ces architectures hybrides ne sont pas présentés dans ce manuscrit. En effet, un décodeur décrit en VHDL a été développé et testé. Comme nous l'avons mentionné, ces études et cette expérience acquise nous ont amené aux résultats présentés dans ce chapitre.

Le **chapitre 4** décrit la mise en oeuvre de l'ensemble des résultats présentés dans les chapitres 2 et 3. Des simplifications algorithmiques introduites par l'utilisation

d'algorithmes dits sous-optimaux sont dans un premier temps illustrées. L'étude des quantifications et de différentes méthodes de réduction des dynamiques des données internes au décodeur sont des contributions originales de notre travail. La réalisation d'un décodeur intégré sur un composant programmable de type FPGA a illustré une analyse fine de la complexité du décodeur. Des simulations dans divers contextes ont contribué à la demonstration de la validité de nos choix. Nous avons en particulier montré l'importance du choix du degré de connexion maximum des noeuds de données sur la complexité du décodeur. Avec une complexité et des débits raisonnables, de bonnes performances sont atteintes quelque soit le rendement de codage. Par les différents choix réalisés sur les codes considérés (structure de codage et distribution des degrés), nous nous sommes attachés à minimiser la densité de la matrice de contrôle de parité, et donc la complexité du décodeur, tout en garantissant des bonnes performances. Dans une dernière partie, le décodeur étudié est comparé à un turbo décodeur duo-binaire 8 états. Nous avons pu montrer le bon comportement global de notre décodeur. En terme de complexité, le décodeur turbo a un léger avantage notamment en termes de mémoires nécessaires. Ces résultats sont à pondérer avec la maturité avec laquelle l'architecture a été définie. Concernant les performances, nous avons pu montrer qu'il n'existe pas une famille de codes surclassant l'autre. Chacune de ces familles a ses spécificités, ses avantages et inconvénients. Les Turbo-codes duo-binaire sont plus intéressants dans la région du *waterfall*, alors que les codes LDPC étudiés sont caractérisés par un plancher d'erreur qui intervient à des taux d'erreur plus bas. L'application visée et le point de fonctionnement du système doit donc être le paramètre de choix de la technique de codage.

La majorité de ces résultats ont donné lieu à des publications dans des conférences internationales, une publication dans un article de revue et ont conduit à plusieurs dépôts de brevets. La liste de ces contributions est décrite en annexe D.

Perspectives

Le travail réalisé dans cette thèse consiste en un point de départ à toute une série de travaux possibles. Plusieurs perspectives sont envisageables.

Tout d'abord, un complément de mesures semble nécessaire pour valider l'approche. Les cas de canaux différents du canal AWGN doivent être explorés. Dans un premier temps, des mesures sur canal de Rayleigh peuvent être menées. Dans un second temps, l'intégration du décodeur proposé dans une chaîne globale incluant un canal réaliste semble nécessaire. Il pourra être judicieux de paramétrer le décodeur conjointement avec le système environnant. Ces différentes études sont actuellement menées au sein de France Telecom.

La définition conjointe des codes et des architectures pour un ordonnancement *Turbo Layered BP* peut être étendue à d'autres familles de codes. En particulier il

peut être judicieux de définir une nouvelle caractérisation de la matrice de contrôle de parité. Un premier objectif peut être d'étendre le principe de décodage aux codes dont la matrice de contrôle de parité n'est pas strictement bi-diagonale. Un second objectif est la définition d'une structure de codage moins contraignante avec une granularité plus importante dans le choix des profils d'irrégularité.

Nous avons exploré dans le cadre de cette thèse des architectures pour des systèmes de communications de type radio, où les débits à atteindre sont inférieurs à la centaine de Mbit/s. Du fait des degrés de liberté offerts par les codes LDPC, l'étude d'architectures hauts débits pour des applications de type optique (où l'hypothèse de travail est le Gbit/s) peut être un travail intéressant. Alors que la technologie limite par exemple les temps d'accès aux mémoires, la définition de nouveaux codes ou/et architectures ultra-parallélisées qui allient performance et flexibilité peut être un challenge pour les prochaines années.

D'un point de vue théorique, l'analyse de la convergence de l'algorithme *Turbo Layered BP* par des outils tel que les EXIT Chart peut permettre une meilleure compréhension des phénomènes, et la définition d'outils plus adaptés pour le choix des distributions d'irrégularité des codes. En parallèle, il peut être intéressant de définir une famille de codes ayant des bonnes propriétés dans la région du *waterfall* tout en gardant une structure exploitable pour la définition d'architectures de décodage efficaces. Ces études adressent donc le problème plus général d'optimisation des codes de longueurs finies.

Annexes

Annexe A

Interprétation graphique des règles de construction pour le séquencement Turbo Layered BP

Le chapitre 3 a illustré des règles de construction pour la définition de code supportant différentes versions du séquencement *Turbo Layered BP*. Cette annexe a pour but d'illustrer les règles énoncées par une interprétation graphique.

Soit Δ l'ensemble des q coefficients de permutation présents dans une colonne de la matrice $\mathbf{H_s}$:

$$\Delta = \{\delta_i, i = 0 \dots q-1\} \tag{A.1}$$

On considère le décodage d'une fenêtre de taille m. Comme nous avons pu le montrer dans le chapitre 3, les noeuds de données intervenant dans le décodage de la fenêtre k sont les noeuds d'indice :

$$\beta k + (\delta_i + k) \bmod z, \ i = 0 \dots q-1 \tag{A.2}$$

où β est l'indice de la colonne de $\mathbf{H_s}$ considérée. Il possible de représenter l'évolution des indices des noeuds décodés en fonction du temps en utilisant un cercle divisé en z sections. Les noeuds sont donc représentés sur un cercle trigonométrique dont les positions sont fonction de l'indice de la fenêtre et des coefficients de permutation. Le calcul des positions $P(k, \delta_i)$ se fait de la manière suivante :

$$P(k, \delta_i) = e^{-j\left(\frac{\pi}{2} - 2\pi \frac{(\delta_i + k) \bmod z}{z}\right)} \tag{A.3}$$

Un exemple est illustré sur la figure A.1 dans le cas d'un séquencement série. Si les noeuds de données mis à jour dans la même fenêtre sont différents, alors la règle est respectée.

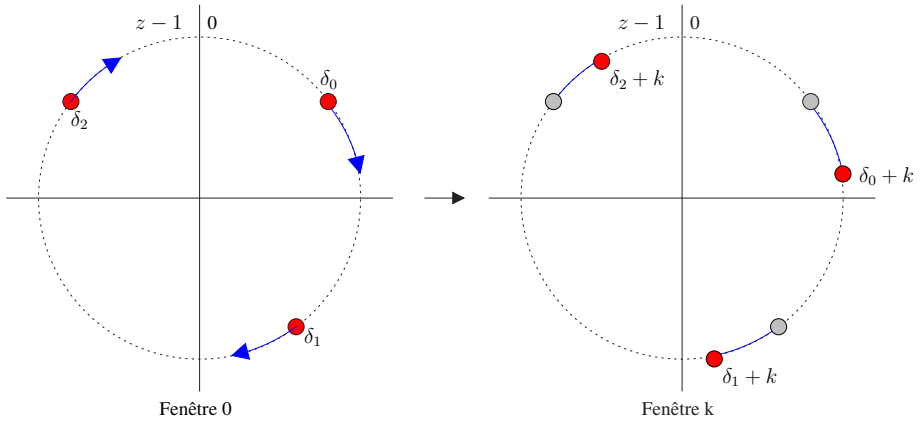


FIG. A.1 – Exemple de représentation graphique illustrant l'évolution des noeuds de données mis à jour en fonction du temps. Dans l'exemple présenté $q = 3$. Cette représentation est celle d'un séquencement série avec une taille de fenêtre égale à m.

Le cas d'un séquencement parallèle est illustré sur la figure A.2. Cette représentation est celle d'un séquencement parallèle avec $p = 2$ et $p = 4$ en considérant une fenêtre de taille égale à m. Dans l'exemple illustré, le cas d'un parallélisme de 2 respecte les règles énoncées. Dans le cas d'un parallélisme de 4, la règle n'est plus respectée. Des noeuds interviennent deux fois dans le décodage des fenêtres traitées en parallèle.

La figure A.3 illustre un exemple de caractérisation de l'évolution des noeuds de données mis en jour pour un séquencement série pipeliné. Dans le cas présenté, la règle de construction est respectée puisque un noeud intervenant dans une fenêtre de décodage n'intervient pas dans une autre tant qu'il n'a pas été mis à jour.

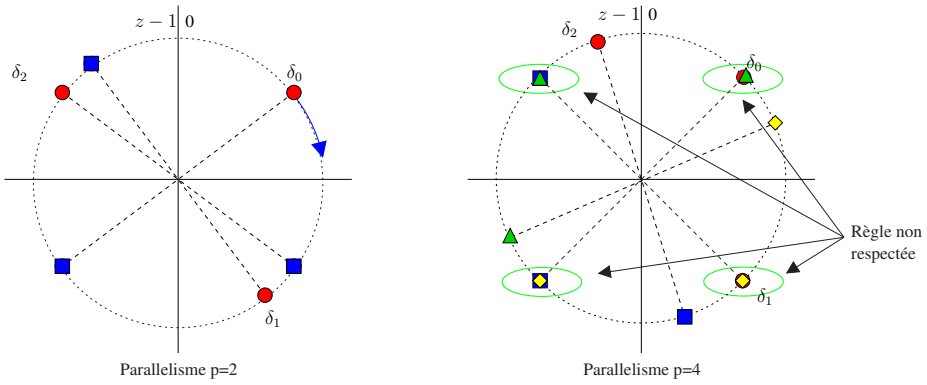


FIG. A.2 – Exemple de représentation graphique illustrant l'évolution des noeuds de données mis à jour en fonction du temps. Dans l'exemple présenté $q = 3$. Les noeuds intervenant dans une même fenêtre sont représentés par le même symbole.

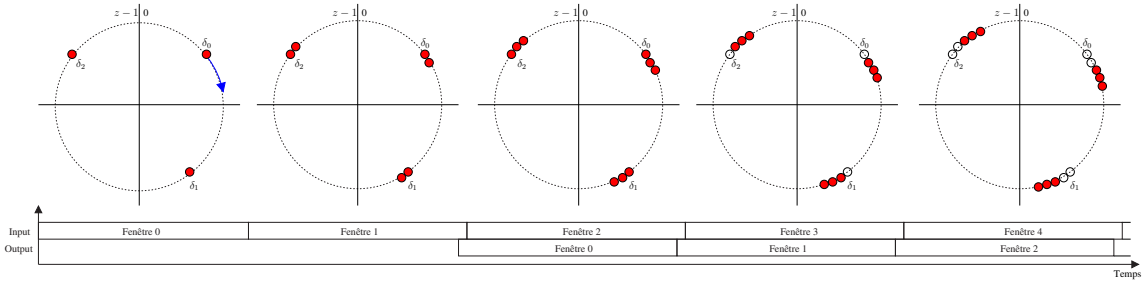


FIG. A.3 – Exemple de représentation graphique illustrant l'évolution des noeuds de données mis à jour en fonction du temps pour une séquencement série pipeliné.

Ces représentations simples permettent de comprendre et d'identifier rapidement les codes respectant une contrainte donnée par le séquencement. Il faut noter que cette représentation doit être réalisée et analysée pour chaque colonne de la matrice $\mathbf{H_s}$.

Annexe B

Matrices équivalentes de structures non strictement bi-diagonales

La méthode de définition d'une matrice équivalente pour rendre flexible une architecture a été adressée dans le chapitre 3 dans l'hypothèse d'un code de type Repeat Accumulate dont la matrice $\mathbf{H_p}$ est strictement bi-diagonale et que la distribution des degrés des noeuds de contrôles est régulière. Nous proposons d'illustrer par un exemple simple, un cas où la distribution des degrés des noeuds de contrôles est une distribution irrégulière et que la matrice de contrôle de parité n'est pas strictement de type bi-diagonale.

Soit la matrice de contrôle de parité suivante définissant un code de rendement $R = 2/3$:

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

et le mot de code correspondant suivant :

$$\underline{x} = [c_0 \quad c_1 \quad c_2 \quad c_3 \quad c_4 \quad c_5 \quad c_6 \quad c_7 \quad c_8 \quad c_9 \quad c_{10} \quad c_{11} \quad \cdots \quad c_{12} \quad c_{13} \quad c_{14} \quad c_{15} \mid p_0 \quad p_1 \quad p_2 \quad p_3 \quad p_4 \quad p_5 \quad p_6 \quad p_7]$$

La matrice n'étant pas strictement bi-diagonale, on isole dans un premier temps la partie de la matrice de contrôle de parité qui sera traitée comme un treillis à deux

états[1] :

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

Cette transformation montre que le noeud de données correspondant au bit p_0 ne sera pas traité dans le treillis. Celui-ci sera traité comme un noeud de données correspondant à un bit systématique. Le nombre d'éléments non nuls dans la partie non bi-diagonale de la matrice de contrôle de parité est soit 8 où 9. Dans un premier temps on considère que l'on dispose un processeur d'équation de parité capable de traiter au maximum 4 noeuds de données. La matrice équivalente associée à cette hypothèse est la suivante :

$$\mathbf{H}_{eq} = \left[\begin{array}{cccccccccccccccccc|cccccccc}
1 & 1 & & & 1 & 1 & & & & & & & & & & & & & \mathbf{1} & & & & & & & & \\
 & & & & & & 1 & 1 & & 1 & 1 & & & & & & & & \mathbf{1} & \mathbf{1} & & & & & & & \\
 & & & & & & & & & & & & & & & & & 1 & & \mathbf{1} & 1 & & & & & & & \\
 & & 1 & & 1 & & & 1 & 1 & & & & & & & & & & & & 1 & \mathbf{1} & & & & & & \\
 & & & & & & & & & & & 1 & 1 & & 1 & 1 & & & & & & & 1 & 1 & & & & \\
 & & 1 & 1 & & 1 & & 1 & & & & & & & & & & & & & & & 1 & \mathbf{1} & & & & \\
 & & & & & & & & & 1 & & 1 & & 1 & 1 & & & & & & & & & 1 & 1 & & & \\
 & & 1 & 1 & & 1 & 1 & & & & & & & & & & & & & & & & & & \mathbf{1} & & & \\
 & & & & & & & & 1 & & 1 & & & & 1 & 1 & & & & & & & & & 1 & 1 & & \\
1 & & 1 & & & 1 & 1 & & & & & & & & & & & & & & & & & & & \mathbf{1} & & \\
 & & & & & & & & 1 & & 1 & & 1 & & & & 1 & & & & & & & & & 1 & 1 & \\
 & & 1 & & 1 & 1 & & & 1 & & & & & & & & & & & & & & & & & & \mathbf{1} & \\
 & & & & & & & & & 1 & & 1 & & 1 & & & & 1 & & & & & & & & & 1 & \\
1 & 1 & & 1 & & 1 & & & & & & & & & & & & & & & & & & & & & & \mathbf{1} \\
 & & & & & & & & 1 & & & 1 & & 1 & 1 & & & & & & & & & & & & & \\
 & 1 & 1 & & & & 1 & & 1 & & & & & & & & & & & & & & & & & & & \\
 & & & & & & & & & 1 & & 1 & 1 & & 1 & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & & & & & & & & & & & & & & 1
\end{array}\right]$$

Le mot de code équivalent est alors :

$$\underline{x}_{eq} = [c_0 \quad c_1 \quad c_2 \quad c_3 \quad c_4 \quad c_5 \quad c_6 \quad c_7 \quad c_8 \quad c_9 \quad c_{10} \quad c_{11} \quad c_{12} \quad c_{13} \quad c_{14} \quad c_{15} \quad p_0 \quad \cdots$$
$$\cdots \quad e_1^0 \quad e_1^1 \quad p_1 \quad e_2^0 \quad p_2 \quad e_3^0 \quad p_3 \quad e_4^0 \quad p_4 \quad e_5^0 \quad p_5 \quad e_6^0 \quad p_6 \quad e_7^0 \quad p_7 \quad e_8^0 \quad e_8^1]$$

Dans le cas présenté ci dessus, la séquence d'éléments neutres à présenter au treillis est moins régulière que dans les cas présenté au chapitre 3. On peut noter que dans cette configuration, un processeur de noeuds de contrôle aura à traiter un seul noeud de données (ce qui peut être réalisé en considérant un processeur à 4 entrées dont 3 sont mise à la valeur correspondant à un 0 avec la plus forte fiabilité possible).

[1] Il est aussi possible de modifier le treillis de manière à décoder la partie non strictement bi-diagonale de la matrice comme un treillis circulaire

Il est aussi possible de décrire une matrice équivalente construite pour un processeur d'équations de parité capable de traiter au maximum 5 noeuds de données. La matrice équivalente associée à cette hypothèse est la suivante :

$$\mathbf{H}_{eq} = \left[\begin{array}{cccccccccccccccccccc|cccccccccccccccc}
1 & 1 & & & 1 & 1 & & & & & & & & & & & & & & & \mathbf{1} & & & & & & & & & & & & & & & & \\
& & & & & & & & 1 & 1 & & 1 & 1 & & & & & & & 1 & \mathbf{1} & 1 & & & & & & & & & & & & & & & & \\
& & 1 & & 1 & & & & 1 & 1 & & & & & & & & & & & & 1 & \mathbf{1} & & & & & & & & & & & & & & & \\
& & & & & & & & & & & & 1 & 1 & & 1 & 1 & & & & & & \mathbf{1} & 1 & & & & & & & & & & & & & & \\
& & 1 & 1 & & 1 & & 1 & & & & & & & & & & & & & & & 1 & \mathbf{1} & & & & & & & & & & & & & & \\
& & & & & & & & & & 1 & & 1 & & 1 & 1 & & & & & & & & \mathbf{1} & 1 & & & & & & & & & & & & & \\
& & 1 & 1 & & 1 & 1 & & & & & & & & & & & & & & & & & 1 & \mathbf{1} & & & & & & & & & & & & & \\
& & & & & & & & & 1 & & 1 & & & & & 1 & 1 & & & & & & & \mathbf{1} & 1 & & & & & & & & & & & \\
1 & & 1 & & & 1 & 1 & & & & & & & & & & & & & & & & & & 1 & \mathbf{1} & & & & & & & & & & & \\
& & & & & & & & & & 1 & & 1 & & 1 & & & 1 & & & & & & & & 1 & \mathbf{1} & & & & & & & & & & \\
& & 1 & & 1 & 1 & & & & 1 & & & & & & & & & & & & & & & & & 1 & \mathbf{1} & & & & & & & & & \\
& & & & & & & & & & & 1 & & 1 & & 1 & & & 1 & 1 & & & & & & & & & 1 & \mathbf{1} & & & & & & & \\
1 & 1 & & 1 & & 1 & & & & & & & & & & & & & & & & & & & & & & & & 1 & \mathbf{1} & & & & & & \\
& & & & & & & & & 1 & & & 1 & & 1 & 1 & & & & & & & & & & & & & & & 1 & \mathbf{1} & & & & & \\
& 1 & 1 & & & & 1 & & & 1 & & & & & & & & & & & & & & & & & & & & & & 1 & \mathbf{1} & & & \\
& & & & & & & & & & 1 & & 1 & 1 & & 1 & & & & & & & & & & & & & & & & & 1 & \mathbf{1} & \\
& & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & 1 & \mathbf{1}
\end{array}\right]$$

Le mot de code équivalent associé est :

$$\begin{aligned}\underline{x}_{eq} = [&c_0 \quad c_1 \quad c_2 \quad c_3 \quad c_4 \quad c_5 \quad c_6 \quad c_7 \quad c_8 \quad c_9 \quad c_{10} \quad c_{11} \quad c_{12} \quad c_{13} \quad c_{14} \quad c_{15} \quad p_0 \quad \cdots \\ &\cdots \quad \mathbf{e_1^0} \quad p_1 \quad \mathbf{e_2^0} \quad p_2 \quad \mathbf{e_3^0} \quad p_3 \quad \mathbf{e_4^0} \quad p_4 \quad \mathbf{e_5^0} \quad p_5 \quad \mathbf{e_6^0} \quad p_6 \quad \mathbf{e_7^0} \quad p_7 \quad \mathbf{e_8^0}]\end{aligned}$$

Dans le cas de cette configuration, l'introduction d'éléments neutres dans le treillis se fait d'une manière régulière. Cependant, la capacité du processeur à traiter 5 entrées en parallèle n'est que très peu exploitée (2 équations sur les 16).

Le choix de cet exemple n'a pas été pris au hasard. La matrice de contrôle de parité étudiée dans ce cas correspond à la matrice caractérisant le protographe des codes LDPC de rendement 2/3 (plus particulièrement les codes 2/3-A) de la norme IEEE 802.16.e [21].

Annexe C

Fichier VHDL de configuration du décodeur

Le fichier de configuration du décodeur écrit en VHDL est le suivant :

```
package def_params is
        --------------------------------
        -- Statique
        --------------------------------
        constant lg2_J       : natural := 3;
        constant J           : natural := 4;
        constant lg2_it      : natural := 5; -- 0 à 31 itérations
        constant Delay       : natural := 4; -- Paramètre de réglage du délai
                                             -- de lecture dans les bancs mémoires
        constant P           : natural := 2; -- Parallélisme
        -------------------------------------
        -- Dépendant du code
        -------------------------------------
        constant M_max       : natural := 20;  -- Défini le nombre de bancs mémoire
                                               --(ex: k=20 -> M_max=20 et lg2_m_max=5)
        constant z_max       : natural := 81;--
        constant lg2_m_max   : natural := 5;   --
        constant lg2_z_max   : natural := 7;  --
        constant lg2_Rate    : natural := 3;   -- 2:=> 1/2 to 4/5, -- 3: 1/2 to 6/7
        -------------------------------------
        -- Quantifications
        -------------------------------------
        constant lg2_Channel  : natural := 4;
        constant lg2_A        : natural := 8;
        constant lg2_mcv      : natural := 5;
        constant lg2_mvc      : natural := 9;
        constant lg2_FBA      : natural := 9;
        -------------------------------------
end def_params;
```

Annexe D

Liste des contributions

Conférences internationales

- "*A Structured LDPC Code Construction for Efficient Encoder Design*", **J. B. Doré**, M. H. Hamon, P. Pénard, 4th International Symposium on Turbo-Codes and Related Topics, Avril 2006.
- "*Design and Decoding of a Serial Concatenated Code Structure based on Quasi-Cyclic LDPC Codes*", **J. B. Doré**, M. H. Hamon, P. Pénard, IEEE International Conference on Communications (ICC 2006), Juin 2006.
- "*High Speed Decoding of Serial Concatenated Codes*", **J. B. Doré**, M. H. Hamon, P. Pénard, IEEE Global Telecommunications Conference (GLOBECOM 2006), Décembre 2006.
- "*Interleaver for High Parallelizable Turbo Decoder*", L. Boher, **J. B. Doré**, M. Hélard, C. Gallard, 6th International Workshop on Multi-Carrier Spread Spectrum (MCSS 2007), Mai 2007.
- "*Cycle and minimum distance properties of structured LDPC codes based on circulant permutation matrices*", **J. B. Doré**, M. H. Hamon, P. Pénard, 10th Canadian Workshop on Information Theory (CWIT 2007), Juin 2007.
- "*On Flexible Design and Implementation of Structured LDPC codes*", **J. B. Doré**, M. H. Hamon, P. Pénard, 18th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC 2007), Septembre 2007.
- "*Architecture and Design Methodology for Structured LDPC Decoder*", **J. B. Doré**, M. H. Hamon, P. Pénard, 66th IEEE Vehicular Technology Conference (VTC fall 2007), Septembre 2007.

Articles de revues

- "*Design Methodology and Decoder Architecture for Structured LDPC codes based on Circulant Permutation Matrices*", **J. B. Doré**, M. H. Hamon, P. Pénard, soumis à IEEE Transactions on communications.

Contribution en normalisation

- R1-063097, "*A new highly parallelizable interleaver for LTE turbo-codes*", France Telecom, 3GPP TSG RAN1 WG1 Meeting #47, Novembre 2006.

Brevets - (Déposés au nom de France Telecom)

- "*Procédé et dispositif d'encodeur de code LDPC*", **J. B. Doré**, M. H. Hamon, P. Pénard, FR 05 07007.
- "*Procédés d'encodage et de décodage rapides et dispositifs associés*", **J. B. Doré**, M. H. Hamon, P. Pénard, FR 06 50196.
- "*Codage et décodage d'un signal de données en fonction d'un code correcteur*", **J. B. Doré**, M. H. Hamon, P. Pénard, FR 06 54864.
- "*Codage et décodage de signaux de données de rendements variables*", **J. B. Doré**, M. H. Hamon, P. Pénard, FR 07 53801.
- "*Procédés d'entrelacement et de désentrelacement d'une trame de symboles, produits programme d'ordinateur, entrelaceur, désentrelaceur, codeur et décodeur correspondants*", **J. B. Doré**, L. Boher, M. Hélard, FR 06 09475.

Table des figures

1.1 Modèle de communication numérique. 2
1.2 Schéma simplifié d'un codeur de canal qui à partir d'un mot d'information $\underline{c}$de K bits génère un mot de code $\underline{x}$et $N = K + M$ bits. . . . 2
1.3 Illustration des trois régions caractérisant les performances d'un système de codage. 6
1.4 Concatenation série d'un code interne et d'un code externe. 7
1.5 Schéma de principe d'un Turbo-code, construit à partir de deux codes convolutifs récursifs et d'un entrelaceur Π. 8
1.6 Schéma de principe d'un turbo décodeur. La notation Π représente la fonction d'entrelacement et Π^{-1} le dé-entrelacement. A partir des symboles reçus $\underline{c_{obs}}$et $\underline{p^0_{obs}}$le premier décodeur (DEC 1) calcule une information extrinsèque qui est échangée avec le décodeur 2 (DEC 2). De la même façon, le décodeur 2 calcule à partir des observations du canal $\underline{c_{obs}}$entrelacées et $\underline{p^1_{obs}}$une nouvelle information extrinsèque accompagnée d'une estimation des bits d'information. Ce processus peut être itéré autant de fois que le nombre maximum d'itérations choisi. 9
1.7 Graphe factoriel d'un code LDPC. 12
1.8 Exemple de cycles de longueur 4, 6 et 8 13
1.9 Représentation de la fonction $f(.)$ définie équation 1.26. 16
1.10 Illustration de la mise à jour des messages se propageant d'un noeud de données vers un noeud de contrôle m_{vc}. 17
1.11 Illustration de la mise à jour des messages se propageant d'un noeud de contrôle à un noeud de données m_{cv}. 17
1.12 Comparaison des algorithmes de décodage à travers une représentation dans le plan performance complexité. 21
1.13 Illustration de l'ordonnancement Horizontal Shuffle Scheduling et Vertical Shuffle Scheduling au cours de l'itération i 23
1.14 Forme particulière de la matrice $\mathbf{H_p}$ normalisée dans des standards. . 26

1.15 Illustrations des différentes représentations d'un code de type Repeat Accumulate. Ces codes peuvent être vus comme des codes LDPC (a). Ils peuvent aussi être représentés par une concaténation d'un code de parité, alimenté par un organe de répétition (Rep) dont les sorties sont entrelacées (Π), et d'un accumulateur (b). Enfin, ces codes peuvent être vus comme la concaténation série d'un répéteur, d'un accumulateur. Les sorties sont alors poinçonnées (Punct) pour obtenir le rendement codage choisi (c). 26
1.16 Exemple de construction d'un code à partir d'un protographe et de matrices d'expansion de type identité permutées. 29
1.17 Exemple d'architecture parallèle des processeurs associés aux noeuds de données et de contrôle. Le décodage se fait suivant l'algorithme BP. Seul le calcul des fiabilité des messages de sorties du noeud de contrôle est illustré. Comme décrit dans [27], le traitement par la fonction $f(.)$ peut être indifféremment réalisé soit au niveau du noeud de contrôle, soit un niveau du noeud de données 33
1.18 Exemple d'architecture série des processeurs associés aux noeuds de données et de contrôle. Le décodage se fait suivant l'algorithme BP. Seul le calcul des fiabilités des messages de sorties du noeud de contrôle est illustré. 34
1.19 Exemple d'architecture série associée à un décodage suivant un treillis. 34
1.20 Représentation en treillis d'une équation de parité. La première étape de la transformation consiste en l'introduction de noeuds cachés (noeuds noirs) qui représentent l'état du treillis. Finalement la représentation en treillis est déduite. Chaque branche représente l'état d'un noeud de données ("1"trait pointillé, "0"trait plein). 35

2.1 Exemple d'un cycle de longueur 8 faisant intervenir quatre noeuds de donnés de degré 2 entraînant un mot de poids 4. 39
2.2 Exemple de noeuds de degré 2 connectés sous forme d'une chaîne. . . 39
2.3 Exemple d'une matrice identité de taille 4×4 circulairement permutée de $p = 1$ position vers la droite. 42
2.4 Synthèse de la méthode et des arguments en faveur de la structure proposée. 46
2.5 Illustration de la densité de probabilité relative à la fonction IOWEF d'un accumulateur pour une longueur $N = 1000$. 47
2.6 Exemple de calcul du poids de sortie en fonction des positions non nulles du mot présentées à l'entrée de l'accumulateur. Le mot codé est dans ce cas de poids 3. 48
2.7 Illustration du repère orthonormal associé à une matrice identité permutée. 54
2.8 Projection verticale (a) et horizontale (b) du point A sur la matrice identité $\mathbf{I}_{\delta_B}$ 54
2.9 Exemple d'une configuration avec un cycle de longueur 4. 55

2.10 Illustration d'un graphe de base. 57
2.11 Illustration de la construction de l'arbre du graphe de base à l'étape 3 pour la recherche de cycles de longueur 4. Dans le cas présenté deux cycles sont détectés. 62
2.12 Exemple de comportement d'un décodeur BP vis à vis de deux configurations différentes. Une des configurations du mot d'entrée a excité un ensemble piège, résultant en une instabilité du décodeur. 63
2.13 Exemple d'un pseudo-mot de code $(7, 1)$. Les noeuds de données grisés correspondent à une valeur binaire de 1. Les noeuds de données blancs correspondent à une valeur binaire de 0. Dans ce cas, une seule équation de parité n'est pas respectée. 64
2.14 Distribution des cycles dans le graphe du code et dans des sous graphes regroupant les noeuds de même degrés. 66
2.15 Simulation des deux codes issus de l'algorithme d'optimisation avec des paramètres de constructions différents 66
2.16 Comparaison des performance sur canal AWGN de codes LDPC de taille $N = 3600$ et de rendement $R = 1/2$. Les paramètres de chaque code sont décrits dans la table 2.4. Les performances ont été mesurées sur un décodeur matériel. 68
2.17 Schéma du décodeur associé à un décodage d'une équation de parité dans un séquencement *Layered BP*. d_c représente le nombre de bits intervenant dans l'équation de parité. 70
2.18 Schéma du décodage par l'algorithme *Layered BP* d'un code dont la matrice de contrôle de parité $\mathbf{H}$ est construite à partir de matrice identité permutées. Le concept de décodage par couche (Layer) est ici illustré. 72
2.19 Simulation d'un code de taille $N = 960$ et de rendement $R = 1/2$ lorsque celui ci est décodé par un algorithme BP avec un ordonnancement par inondation (BP), et par un algorithme *Layered BP* (LBP). La vitesse de convergence de l'algorithme *Layered BP* (LBP) est supérieur à celle de l'algorithme BP. 73
2.20 Illustration de l'équivalence entre une chaîne et un treillis. 74
2.21 Illustration du calcul Aller (Forward) des messages de branches (a) et le calcul Retour (Backward). 75
2.22 Illustration du décodage par fenêtres du code. La matrice de contrôle de parité est tout d'abord représentée. Les éléments intervenant dans une même fenêtre ont des couleurs identiques. Le graphe de l'accumulateur et sa décomposition en fenêtres sont également illustrés. 77
2.23 Illustration de la phase Forward et Backward du calcul des messages du codes LDPC défini par la matrice $[\mathbf{H_s}\ \mathbf{I}]$. 78
2.24 Illustration de l'initialisation des fenêtres par la méthode du pointeur. 79

2.25 Comparaison de performances entre les différents algorithmes étudiés : *Layered BP* (LBP), *Turbo BP* (TBP) et *Turbo Layered BP* (TLBP). Le séquencement TLBP qui correspond à un décodage conjoint des codes interne et externe apporte la meilleure convergence. 80

3.1 Modélisation générique d'un processeur CNP adapté pour un algorithme *Layered BP*. Le nombre de cycles nécessaires à chaque traitement est illustré. 88
3.2 Synoptique du temps de traitement d'un processeur CNP^{Σ} série et exemple d'architecture associée 89
3.3 Synoptique du temps de traitement d'un processeur CNP^{X} série avec un parallélisme interne de 2 et exemple d'architecture associée 90
3.4 Synoptique du temps de traitement d'un processeur CNP^{Σ^2} série où les noeuds sont traités par couple et exemple d'architecture associée. 91
3.5 Mise en évidence d'un ordonnancement avec un processeur CNP^{Σ} où la ressource n'est pas utilisée de manière efficace. L'utilisation de la ressource pour chaque processeur est illustrée. 92
3.6 Séquencement de l'algorithme de décodage utilisant un processeur CNP^{σ} permettant l'utilisation en continu des ressources matérielles. . 93
3.7 Exemple de séquencement pipeliné à l'interface entre deux matrices identité permutées $\mathbf{I}_{\delta_0}$ et $\mathbf{I}_{\delta_1}$. Pour respecter la règle énoncée, il est nécessaire que $(\delta_0+z-1)\mathrm{mod}\ z \neq \delta_1$, $(\delta_0+z-1)\mathrm{mod}\ z \neq (\delta_1+1)\mathrm{mod}\ z$. Dans ce cas le rapport $\lceil L/T_c \rceil = 3$ 95
3.8 Séquencement de l'algorithme de décodage utilisant un processeur CNP^{X} permettant de maximiser l'utilisation des ressources matérielles. Comme le montre la figure, ce séquencement ne permet pas l'utilisation optimale des ressources. 98
3.9 Illustration du séquencement série. Une fenêtre de décodage peut commencer à être traitée quand le résultat du décodage de la précédente est disponible. 99
3.10 Schéma de architecture d'un décodeur *Turbo Layered BP* basée sur trois processeurs de décodage. Le processeur annoté *Forward SPC* (Single Parity Check) calcule les messages du code interne vers le code externe m_{IO}. Le processeur traitant les messages de sortie de l'accumulateur est quant à lui annoté *Backward SPC* 100
3.11 Séquencement série de l'algorithme *Turbo Layered BP*, où les différentes étapes du décodage sont représentées. 101
3.12 Illustration du séquencement série pipeliné. Le décodage des fenêtres se fait sans interruption. 102
3.13 Illustration du séquencement série pipeliné. 103
3.14 Illustration du séquencement parallèle. Le décodage de p fenêtres se fait simultanément. Un nouveau groupe de fenêtres est décodé quand le décodage du groupe précédent est terminé. 105
3.15 Illustration du séquencement parallèle pipeliné. 106

3.16 Illustration dans un repère Débit/Complexité/Contraintes des architectures étudiées en fonction du parallélisme p. 107
3.17 Variation du degré de connexion moyen des noeuds de contrôle en fonction du rendement de codage. Les moyennes ont été calculées à partir des codes LDPC normalisés dans le cadre des standards DVB-S2, IEEE 802.16e, IEEE 802.11n et de la base de données de codes LDPC optimisés (sous la contrainte d'un degré de connexion des noeuds de données maximum de 20, et une part de noeuds de données de degré 2 compatible avec la forme bi-diagonale) de l'EPFL [98]. Cet ensemble de code est noté LdpcOpt. 113
3.18 Illustration des codes de type S-SCP ((a) et (b)) et ARA (c). I correspond à une fonction d'entrelacement, Rpt a une répétition et Punct a une fonction de poinçonnage. 116
3.19 Représentation simplifiée du graphe du code S-SCP étudié dans [60]. Le code interne et externe sont des codes dont le treillis est à deux états. La représentation reprend le formalisme introduit par Richardson *et al.* dans [102] où chaque type de branche est représenté par une couleur différente. 117

4.1 Exemple de performances, en terme de taux d'erreur binaire (BER), de différentes approximations. 124
4.2 Illustration du fonctionnement du décodeur. Une fois que les mémoires sont initialisées (représentées par des cylindres), le décodage de la trame peut se réaliser suivant le séquencement décrit. La déclaration de fin de décodage dépend du nombre d'itérations maximum choisi. . 125
4.3 Exemple de calcul de messages m_{cv} en fonction des messages m_{vc} et m_{OI} 127
4.4 Architecture pipelinée du processeur SPC forward. Les étages de pipeline sont modélisés par des bascules. Les accès aux mémoires sont représentés par des bus et sont modélisés par des rectangles. 128
4.5 Architecture pipelinée du processeur SPC backward. Les étages de pipeline sont modélisés par des bascules. 128
4.6 Architecture pipelinée du processeur FBA décodant le treillis de l'accumulateur. 129
4.7 Exemple d'initialisation de fenêtre dans le cas d'un parallélisme de 2. Il est possible d'initialiser les dernières fenêtres de l'itération $i+1$ par une métrique calculée elle aussi à l'iteration $i+1$. 130
4.8 Illustration de l'effet de la quantification des données d'entrée sur les performances du décodeur. BP correspond à un décodeur en virgule flottante n'utilisant aucune approximation. Min correspond à un décodeur en virgule flottante utilisant l'approximation du minimum avec $\beta_f = 0.75$, $\beta_b = 1$, $\beta_g = 1$. 132

4.9 Illustrations des performances de différentes stratégies de pondération dans le cas d'un décodeur en virgule flottante (courbes continues) et dans le cas d'un décodeur en virgule fixe avec une représentation des données d'entrées sur 4 (a) et 5 bits (b) (courbes pointillées). 133
4.10 Illustration de l'effet de la quantification de l'information extrinsèque. 137
4.11 Illustration des différentes méthodes pour le calcul de l'information extrinsèque après seuillage de l'information *a posteriori*. 138
4.12 Variation de la mémoire totale du décodeur en fonction de la quantification de l'information *a posteriori*. ml_p représente la profondeur des buffers (l_p est un facteur qui dépend de la latence des processeurs). . 140
4.13 Illustration des performances d'un code irrégulier de taille $N = 3600$ pour différentes stratégie de quantification. 141
4.14 Illustration de l'organisation de la mémoire stockant l'information *a posteriori*. 143
4.15 Chaîne de test mis en oeuvre sur le composant programmable. 144
4.16 Registre à décalage générant la séquence pseudo-aléatoire. 145
4.17 Illustration du décodeur intégré dans le composant programmable. . 146
4.18 Illustration de la carte fille composée de deux FPGA STRATIX EP1S80F-C6 et d'un FPGA EP1S25F-C6. Une partie de ce dernier est utilisé pour la communication avec le bus PCI. La plate-forme de test dans un châssis de PC est aussi illustrée. 147
4.19 Illustration de l'interface graphique communicant utilisée pour l'automatisation des mesures. 148
4.20 Variation du débit utile en sortie du décodeur en fonction du nombre d'itérations. 150
4.21 Illustration de performances de codes de taille $N = 16320$ bits avec des rendements de 1/2, 2/3, 3/4, 5/6 et 7/8. Le taux d'erreur binaire correspond aux courbes en trait plein, le taux d'erreur paquet aux courbes en pointillé. 152
4.22 Illustration de performances de codes de taille $N = 60000$ pour une distribution avec des noeuds de degrés 6 (Code 1) et des degrés 9 (Code 2). La limite de Shannon ainsi que les seuils de convergences calculés sont aussi illustrés. 153
4.23 Illustration des performances du code LDPC DVB-S2 de taille $N = 64800$ et de rendement $R = 1/2$ et comparaison avec un code proposé dont la matrice de contrôle de parité est de densité inférieure. Le décodage a été réalisé en virgule flottante avec 50 itérations de décodage BP. 154
4.24 Variation du débit utile en sortie des décodeurs en fonction du nombre d'itérations. Les débits illustrés sont les débits maximum (Dans le cas du décodeur TC, on considère la plus grande taille ce qui minimise l'effet du paramètre ϵ). 155

4.25 Illustration de performances de codes LDPC et Turbo-code duo-binaire 8 états de taille $K = 1800$ pour un rendement de codage de 1/2. . . . 158
4.26 Illustration de performances de codes LDPC et Turbo-code duo-binaire 8 états de taille $K = 1800$ pour un rendement de codage de 2/3. . . . 159
4.27 Illustration de performances de codes LDPC et Turbo-code duo-binaire 8 états de taille $K = 1800$ pour un rendement de codage de 3/4. . . . 160
4.28 Illustration de performances de codes LDPC et Turbo-code duo-binaire 8 états de taille $K = 1800$ pour un rendement de codage de 5/6. . . . 161

A.1 Exemple de représentation graphique illustrant l'évolution des noeuds de données mis à jour en fonction du temps. Dans l'exemple présenté $q = 3$. Cette représentation est celle d'un séquencement série avec une taille de fenêtre égale à m. 170
A.2 Exemple de représentation graphique illustrant l'évolution des noeuds de données mis à jour en fonction du temps. Dans l'exemple présenté $q = 3$. Les noeuds intervenant dans une même fenêtre sont représentés par le même symbole. 170
A.3 Exemple de représentation graphique illustrant l'évolution des noeuds de données mis à jour en fonction du temps pour une séquencement série pipeliné. 171

Liste des tableaux

1.1 Liste des algorithmes de décodage de code LDPC couramment utilisés dans la littérature. Les différentes simplifications utilisées peuvent être combinées pour produire d'autres algorithmes de décodage et atteindre un objectif de compromis performance-complexité 19

2.1 Exemple de paramètres pour un code irrégulier de type $(q = 3, x = 2)$ de rendement $R = 1/2$, avec $J = 4$. 41
2.2 Illustration du choix du paramètre m en fonction des distributions des degrés. 45
2.3 Paramètres des codes simulés. On rappelle que w_i correspond au plus petit poids des mots de codes issus du codage de l'ensemble des mots d'information de poids i. 66
2.4 Paramètres des codes simulés figure 2.16 67

4.1 Recapitulation de la taille des mémoires utilisées dans le décodeur proposé. 142
4.2 Paramètres des codes LDPC définis dans le cadre de la normalisation IEEE 802.11n 149
4.3 Paramètres de configuration du décodeur pour un contexte se rapprochant de celui défini dans le cadre de la normalisation IEEE 802.11n 149
4.4 Résultats de synthèse de décodeurs dans un contexte se rapprochant de celui défini dans le cadre de la normalisation IEEE 802.11n 150
4.5 Complexité détaillé du décodeur de la configuration 3 151
4.6 Résultats de synthèse de décodeurs dans un contexte de type DVB-T2 151
4.7 Distribution des degrés des codes de tailles $N = 60000$. L'unique noeud de données de degré 1 a été volontairement oublié pour plus de clarté dans l'écriture des distributions. La distribution des noeuds de contrôle est dans les deux cas égale à $\tilde{\rho} \simeq x^5$. 152
4.8 Résultats de complexités du décodeur turbo intégré sur le même composant et la même chaîne que le décodeur LDPC. 156

Bibliographie

[1] Sous la direction d'Alain Glavieux, *Codage de Canal, des bases théoriques aux turbocodes*. Hermes Science, 2005.

[2] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding :turbo-codes," *IEEE International Conference on Communications*, vol. 2, may 1993.

[3] R. G. Gallager, "Low-density parity-check codes," Ph.D. dissertation, 1963.

[4] J. Proakis, *Digital Communications*, McGraw-Hill, Ed., 2001.

[5] P. Elias, "Coding for noisy channels," *IRE Conv. Record*, 1955.

[6] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, April 1967.

[7] G. D. Forney, "Concatenated codes," *MA : MIT Press Cambridge*, 1966.

[8] A. Hocquenghem, "Codes correcteurs d'erreurs," *Chiffre*, 1959.

[9] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. SIAM*, 1960.

[10] *Digital Video Broadcasting (DVB) ; Framing structure, channel coding and modulation for digital terrestrial television*, ETSI EN 300 744 Std. v1.5.2, 2004.

[11] G. Battail, "Coding for the gaussian channel : the promise of weighted-output decoding," *International Journal of Satellite Communications*, 1989.

[12] J. Hagenauer and P. Hoeher, "A viterbi algorithm with soft-decision outputs and its applications," *IEEE Global Conference on Communications*, Nov 1989.

[13] C. Berrou, Y. Saouter, C. Douillard, S. Kerouedan, and M. Jezequel, "Designing good permutations for turbo codes : towards a single model," *IEEE International Conference on Communications*, June 2004.

[14] R. Pyndiah, A. Glavieux, A. Picart, and S. Jacq, "Near-optimum decoding of products codes," *IEEE Global Conference on Communications*, Nov 1994.

[15] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial concatenation of interleaved codes : Performance analysis, design, and iterative decoding," *IEEE Transactions on Information Theory*, May 1998.

[16] D. MacKay and R. M. Neal, "Near shanon limit performance of low density parity-check codes," *Electronic Letter*, August 1996.

[17] J. Pearl, *Probabilistic reasoning in intelligent systems : networks of plausible inference*, S. Mateo, Ed. Morgan Kaufmann Publishers, 1988.

[18] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Analysis of low density codes and improved designs using irregular graphs," *Proceeding of 30th ACM Symp. on Theory of Computing*, 1998.

[19] T. Richardson, A. Shokrollahi, and R. Urbanke, "Design of capacity approaching irregular low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, Feb 2001.

[20] *Digital Video Broadcasting (DVB) ; Second Generation framing structure, channel coding and modulation systems for broadcasting, Interactive services, news gathering and other broadband satellite applications*, ETSI EN 302 307 Std. v1.1.1, 2004.

[21] *Draft Amendment to IEEE Standard for Local and Metropolitan Area Networks - Part 16*, IEEE Project 802.16.e Std. 10, 2005.

[22] *Draft Amendment to IEEE Standard for Information Technology-Telecommunications and information exchange between systems-Local and Metropolitan networks-specific requirements-Part 11 : Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications : Enhancements for Higher Throughput*, IEEE Project 802.11.n Std. 2, 2007.

[23] R. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on Information Theory*, vol. 27, sept 1981.

[24] I. Andriyanova, J. Tillich, and J. Carlach, "Asymptotically good codes with high iterative decoding performances," *International Symposium on Information Theory*, Sept 2005.

[25] M. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *IEEE Transactions on Communications*, vol. 47, May 1999.

[26] J. Chen and M. Fossorier, "Density evolution of two improved BP-based algorithms for LDPC decoding," *IEEE Communications Letters*, March 2002.

[27] F. Guilloud, "Architecture generique de decodeur de codes LDPC," Ph.D. dissertation, 2004.

[28] C. Jones, E. Valles, M. Smith, and J. Villasenor, "Approximate-MIN constraint node updating for LDPC code decoding," *IEEE Military Communications Conference*, vol. 1, Oct 2003.

[29] J. Zhang, M. Fossorier, and D. Gu, "Two-dimensional correction for min-sum decoding of irregular LDPC codes," *IEEE Communications Letters*, Mar 2006.

[30] E. Yeo, P. Pakzad, B. Nikolic, and V. Anantharam, "High throughput low-density parity-check decoder architectures," *IEEE Global Conference on Communications*, Nov 2001.

[31] H. Kfir and I. Kanter, "Parallel versus sequential updating for belief propagation decoding," *Physica A*, 2003.

[32] J. Zhang and M. Fossorier, "Shuffled belief propagation decoding," *Conference Record of the Thirty-Sixth Asilomar Conference on Signals, Systems and Computers*, vol. 1, Nov 2002.

[33] E. Sharon, S. Litsyn, and J. Goldberger, "An efficient message-passing schedules for LDPC decoding," *23rd IEEE Convention of Electrical and Electronics Engineers in Israel*, Sept 2004.

[34] J. Zhang, Y. Wang, M. Fossorier, and J. Yedidia, "Replica shuffled iterative decoding," *IEEE International Symposium on Information Theory*, 2005.

[35] E. Sharon, S. Litsyn, and J. Goldberger, "Convergence analysis of serial message-passing schedules for LDPC decoding," *4th International Symposium on Turbo-Codes and Related Topics*, April 2006.

[36] O. Muller, A. Baghdadi, and M. Jézéquel, "On the parallelism of convolutional turbo decoding and interleaving interference," *IEEE Global Conference on Communications*, Nov 2006.

[37] Y. Mao and A. Banihashemi, "Decoding low-density parity-check codes with probabilistic schedule," *IEEE Pacific Rim Conference on Communications, Computers and signal Processing*, vol. 1, Aug 2001.

[38] R. Bresnan, "Novel code construction and decoding techniques for LDPC codes," Ph.D. dissertation, 2004.

[39] F. Kienle, T. Lehnigk-Emden, and N. Wehn, "Fast convergence algorithm for LDPC codes," *IEEE Vehicular Technology Conference*, May 2006.

[40] T. Richardson and R. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, Feb 2001.

[41] D. Divsalar, H. Jin, and R. McEliece, "Coding theorems for turbo-like codes," *Proceeding of the 36th Allerton conference on communication, control and computing*, 1998.

[42] H. Jin, A. Khandekar, and R. McEliece, "Irregular repeat-accumulate codes," *Second International Conference on Turbo Codes*, Sept 2000.

[43] H.-A. Loeliger, "New turbo-like codes," *IEEE International Symposium on Information Theory*, 1997.

[44] C. Berrou, Y. Saouter, and H. Gonzalez, "Adding a rate-1 third dimension to turbo codes," *Workshop on Signal Processing for Wireless Communications*, 2004.

[45] J. Fan, "Array codes as low-density parity-check codes," *Second International Conference on Turbo Codes*, Sept 2000.

[46] L. Chen, "Construction of structured low-density-parity-check codes : Combinatorial and algebraic approaches," Ph.D. dissertation, 2004.

[47] V. Mannoni, "Optimisation des codes LDPC pour communications multiporteuses," Ph.D. dissertation, 2004.

[48] S. ten Brink, "Convergence of iterative decoding," *Electronics Letters*, vol. 35, June 1999.

[49] S.-Y. Chung, T. Richardson, and R. Urbanke, "Analysis of sum-product decoding of low-density parity-check codes using a gaussian approximation," *IEEE Transactions on Information Theory*, vol. 47, Feb 2001.

[50] X.-Y. Hu, E. Eleftheriou, and D.-M. Arnold, "Progressive edge-growth tanner graphs," *IEEE Global Telecommunications Conference*, vol. 2, Nov 2001.

[51] M. Fossorier, "Quasi-cyclic low-density parity-check codes from circulant permuation matrices," *IEEE Transactions on Information Theory*, vol. 50, Aug 2004.

[52] R. Tanner, D. Sridhara, A. Sridharan, T. Fuja, and D. Costello, "LDPC block and convolutional codes based on circulant matrices," *IEEE Transactions on Information Theory*, vol. 50, Dec 2004.

[53] R. M. Tanner, "On quasi-cyclic repeat-accumulate codes," *in Proc. of. the 37th Allerton Conference*, 1999.

[54] J. Thorpe, "Low density parity check (LDPC) codes constructed from protographs," *JPL INP Progress Report 42-154*, 2003.

[55] A. Roumy, S. Guemghar, G. Caire, and S. Verdu, "Design methods for irregular repeat accumulate codes," *IEEE Transaction on Information theory*, vol. 50, August 2004.

[56] M. Yang, W. Ryan, and Y. Li, "Design of efficiently encodable moderate-length high-rate irregular LDPC codes," *IEEE Transactions on Communications*, vol. 52, April 2004.

[57] R. Echard, "On the construction of some deterministic low-density parity-check codes," Ph.D. dissertation, 2002.

[58] J. B. Doré, M. H. Hamon, and P. Pénard, "A structured LDPC code construction for efficient encoder design," *IEEE International Conference on Communications*, June 2006.

[59] A. Abbasfar, D. Divsalar, and K. Yao, "Accumulate repeat accumulate codes," *IEEE International Symposium on Information Theory*, 2004.

[60] K. M. Chugg, P. Thiennviboon, G. D. Dimou, P. Gray, and J. Melzer, "A new class of turbo-like codes with universally good performance and high-speed decoding," *IEEE Milcom 2005*, Oct 2005.

[61] C. Howland and A. Blanksby, "A 220 mW 1 Gb/s 1024-bit rate-1/2 low density parity check code decoder," *IEEE Conference on Custom Integrated Circuits*, May 2001.

[62] A. Tarable, S. Benedetto, and G. Montorsi, "Mapping interleaving laws to parallel turbo and LDPC decoder architectures," *IEEE Transactions on Information Theory*, Sept 2001.

[63] D. Gnaedig, E. Boutillon, J. Tousch, and M. Jezequel, "Towards an optimal parallel decoding of turbo codes," *4th International Symposium on Turbo-Codes and Related Topics*, April 2006.

[64] E. Boutillon, J. Castura, and F. R.Kschischang, "Decoder-first code design," *Proceedings of the 2nd International Symposium on Turbo Codes and Related Topics*, Sept 2000.

[65] E. Liao, E. Yeo, and B. Nikolic, "Low-density parity-check code constructions for hardware implementation," *IEEE International Conference on Communication*, June 2004.

[66] K. Shimizu, T. Ishikawa, N. Togawa, T. Ikenaga, and S. Goto, "Partially-parallel LDPC decoder based on high-efficiency message-passing algorithm," *IEEE International Conference on Computer Design*, Oct 2005.

[67] F. Kienle, T. Brack, M. Alles, and N. Wehn, "A synthesizable IP core for WIMAX 802.16e LDPC code decoding," *International Symposium on Personal Indoor and Mobile Radio Communications*, 2006.

[68] M. Rovini and L. F. N. E. L'Insalata, F. Rossi, "VLSI design of a high-throughput multi-rate decoder for structured LDPC codes," *8th Euromicro conference on Digital System Design*, Aug 2005.

[69] M. Mansour, "High-performance decoders for regular and irregular repeat-accumulate codes," *IEEE Global Telecommunications Conference*, December 2004.

[70] F. Kienle and N. Wehn, "Design methodology for IRA codes," *Proceedings of the Asia and South Pacific Design Automation Conference*, 2004.

[71] F. Kienle, T. Brack, and N. Wehn, "A synthesizable IP core for DVB-S2 LDPC code decoding," *Proceedings of Design, Automation and Test*, 2005.

[72] X.-Y. Hu, E. Eleftheriou, D.-M. Arnold, and A. Dholakia, "Efficient implementations of the sum-product algorithm for decoding LDPC codes," *IEEE Global Telecommunications Conference*, vol. 2, Nov 2001.

[73] M. Mansour and N. Shanbhag, "Turbo decoder architectures for low-density parity-check codes," *IEEE Global Conference on Communications*, Dec 2002.

[74] T. Richardson, "Error floors of LDPC codes," *41st Annual Allerton Conference on Communications, Control, and Computing*, Oct 2003.

[75] J.-P. Tillich and G. Zemor, "On the minimum distance of structured LDPC codes with two variable nodes of degree 2 per parity-check equation," *IEEE International Symposium on Information Theory*, July 2006.

[76] T. Tian, C. Jones, J. D. Villasenor, and R. D. Wesel, "Construction of irregular LDPC codes. with low error floors," *IEEE International Conference on Communications*, June 2003.

[77] H. Zhong and T. Zhang, "Design of VLSI implementation-oriented LDPC codes," *IEEE Vehicular Technology Conference*, Oct 2003.

[78] D. Hocevar, "LDPC code construction with flexible hardware implementation," *IEEE International Conference on Communications*, vol. 4, May 2004.

[79] T. Fuja, D. Sridhara, and R. Tanner, "Low density parity check codes from permutation matrices," *in Conf. On Info. Sciences and Sys. The John Hopkins University*, March 2001.

[80] S. Benedetto and G. Montorsi, "Average performance of parallel concatenated block codes," *Electronics Letters*, Feb 1995.

[81] C.-H. Hsu and A. Anastasopoulos, "Asymptoticweight distributions of irregular repeat-accumulate codes," *IEEE Global Conference on Communications*, July 2005.

[82] R. Podemski, W. Holubowicz, C. Berrou, and G. Battail, "Hamming distance spectra of turbo codes," *Ann. Telecomm.*, vol. 50, sept 1995.

[83] S. Myung, K. Yang, and Y. Kim, "Lifting methods for quasi-cyclic LDPC codes," *IEEE Communications Letters*, June 2006.

[84] J. Rosenthal and P. O. Vontobel, "Constructions of LDPC codes using ramanujan graphs and ideas from margulis," *In Proceedings of the 38th Annual Allerton Conference on Communication*, 2000.

[85] D. J. C. MacKay and M. J. Postol, "Weaknesses of Margulis and Ramanujan–Margulis low-density parity-check codes," in *Proceedings of MFCSIT2002, Galway*, ser. Electronic Notes in Theoretical Computer Science, vol. 74. Elsevier, 2003. [Online]. Available : http ://www.inference.phy.cam.ac.uk/mackay/abstracts/margulis.html

[86] L. Dinoi, F. Sottile, and S. Benedetto, "Design of variable-rate irregular LDPC codes with low error floor," *IEEE International Conference on Communications*, June 2005.

[87] G. Richter and A. Hof, "On a construction method of irregular LDPC codes without small stopping sets," *IEEE International Conference on Communications*, June 2006.

[88] J. Chen, R. Tanner, J. Zhang, and M. Fossorier, "Construction of irregular LDPC codes by quasi-cyclic extension," *IEEE Transactions on Information Theory*, April 2007.

[89] G. D. Forney, "Codes on graphs : Normal realizations," *IEEE Transactions on Information Theory*, Feb 2001.

[90] J. Li, K. R. Narayanan, and C. N. Georghiades, "An efficient decoding algorithm for cycle-free convolutional codes and its applications," *IEEE Global Conference on Communications*, Nov 2001.

[91] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Transactions on Information Theory*, vol. 20, March 1974.

[92] J. Garcia-Frias and W. Zhong, "Approaching shannon performance by iterative decoding of linear codes with low-density generator matrix," *IEEE Communications Letters*, vol. 7, June 2003.

[93] S. Benedetto, D. Divsalar, G. Montorsi, and A. F. Pollara, "Soft output decoding algorithms for continuous decoding of parallel concatenated convolutional codes," *IEEE International Conference on Communications*, June 1996.

[94] J.Dielissen and J.Huisken, "State vector reduction for initialization of sliding windows MAP," *2nd International Symposium on Turbo Codes and Related Topics*, Sept 2000.

[95] K. Gunnam, W. Wang, G. Choi, and M. Yeary, "VLSI architectures for layered decoding for irregular LDPC codes of IEEE 802.11n," October 2006.

[96] M. Rovini, N. L'Insalata, F. Rossi, and L. Fanucci, "VLSI design of a high-throughput multi-rate decoder for structured LDPC codes," *8th Euromicro Conference on Digital System Design*, Sept 2005.

[97] E. Boutillon, W. Gross, and P. Gulak, "VLSI architectures for the MAP algorithm," *IEEE Transactions on Communications*, vol. 51, Februray 2003.

[98] "http ://lthcwww.epfl.ch/research/ldpcopt/."

[99] *Multiplexing and Channel Coding (FDD)*, 3GPP, TS25.212 Std. v5.9.0, 2004.

[100] J. Ha, J. Kim, and S. W. McLaughlin, "Rate-compatible puncturing of low-density parity-check codes," *IEEE Transactions on Information Theory*, Nov 2004.

[101] J. Li and K. R. Narayanan, "Rate-compatible low density parity check (RC-LDPC) codes for capacity-approaching ARQ schemes in packet data communications," *IEEE International Conference on Communications, Internet and Information Technology*, Nov 2002.

[102] T. Richardson and R. Urbanke, "Multi-Edge Type LDPC Codes," Tech. Rep., 2004, submitted IEEE IT.

[103] D. Divaslar and F. Pollara, "Hybrid concatenated codes and iterative decoding," *JPL-TDA Progress Report 42-130*, August 1997.

[104] J. B. Doré, M. H. Hamon, and P. Pénard, "Design and decoding of a serial concatenated code structure based on quasi-cyclic LDPC codes," *4th International Symposium on Turbo-Codes and Related Topics*, April 2006.

[105] E. Boutillon, C. Douillard, and G. Montorsi, "Iterative decoding of concatenated convolutional codes : Implementation issues," *Proceedings of the IEEE, special issue on turbo techniques*, June 2007.

[106] J. Danger, A. Ghazel, E. Boutillon, and H. Laamari, "FPGA implementation of gaussian noise generator for communication channel emulation," *The 7th IEEE International Conference on Electronicsm Circuits and Systemes*, vol. 51, Dec 2000.

[107] G. E. P. Box and M. E. Muller, "A note on the generation of random normal deviates," *anms*, vol. 29, pp. 610–611, 1958.

[108] "http ://www.turboconcept.com/prod_tc1000.php."

[109] C. Berrou, Y. Saouter, C. Douillard, S. Kerouedan, and M. Jezequel, "Designing good permutations for turbo codes : towards a single model," *IEEE International Conference on Communications*, June 2004.

[110] L. Boher, J. B. Doré, M. Hélard, and C. Gallard, "Interleaver for high parallelizable turbo decoder," *Proceedings of MC-SS'07*, may 2007.

[111] P. RNRT, "OPUS - optimisation des futures évolutions de l'UMTS," *ANR-05-RNRT-007-03*.

[112] Z. Zhang, L. Dolecek, B. Nikoliæ, V. Anantharam, and M. Wainwright, "Investigation of error floors of structured low-density parity-check codes by hardware emulation," *IEEE Global Conference on Communications*, Nov 2006.

INSA de Rennes
Service des formations

AVIS DU JURY SUR LA REPRODUCTION DE LA THESE SOUTENUE

Titre de la thèse : Optimisation conjointe de codes LDPC et de leurs architectures de décodage et mise en oeuvre sur FPGA.

Nom Prénom de l'auteur : DORE Jean-Baptiste

Membres du jury :

Monsieur	BOUTILLON
Monsieur	FOSSORIER
Monsieur	PYNDIAH
Monsieur	BERROU
Madame	HAMON
Madame	HELARD
Monsieur	LESTABLE

Président du jury : M. Claude BERROU

Date de la soutenance : 26/10/2007

Reproduction de la thèse soutenue :

- [x] Thèse pouvant être reproduite en l'état
- [ ] Thèse ne pouvant être reproduite
- [ ] Thèse pouvant être reproduite après corrections suggérées

Le Directeur,

A. JIGOREL

Rennes, le 26/10/2007

Signature du Président du jury

C. Berrou.