Équipes Traitement de l'Information et Systèmes

THÈSE

Pour obtenir la grade de

DOCTEUR DE L'UNIVERSITÉ DE CERGY-PONTOISE

Spécialité : **Sciences et Technologies de l'Information et de la Communication (STIC)**

Présentée par

Joachim Rosseel

Thèse dirigée par **Inbar Fijalkow** et codirigée par **Valérian Mannoni** et **Valentin Savin**

Préparée au sein de :
Laboratoire des Signaux, Protocoles et Plateformes Radio (LS2PR, CEA-LETI)
Équipe de Traitement des Images et du Signal (ETIS, CNRS UMR 8051)

DÉCODAGE DE CODES CORRECTEURS D'ERREURS ASSISTÉ PAR APPRENTISSAGE POUR L'IOT

Thèse soutenue publiquement le 13 décembre 2023, devant le jury composé de :

Dr. Camille LEROUX Maitre de conférence, Bordeaux INP, Bordeaux	Rapporteur
Prof. Didier LE RUYET Professeur, CNAM, Paris	Rapporteur
Dr. Elsa DUPRAZ Maîtresse de conférence, IMT Atlantique, Brest	Examinatrice
Prof. Catherine DOUILLARD Professeure, IMT Atlantique, Brest	Examinatrice/Présidente
Prof. Charly POULLIAT Professeur des Universités, INP-ENSEEIHT, Toulouse	Examinateur
Prof. Inbar FIJALKOW Professeure des Universités, ENSEA, Cergy-Pontoise	Directrice de thèse
Dr. Valérian MANNONI Docteur-Ingénieur de Recherche, CEA-LETI, Grenoble	Encadrant de thèse
Dr. Valentin SAVIN Docteur-Ingénieur de Recherche, CEA-LETI, Grenoble	Encadrant de thèse

Remerciements

Cette section est l'occasion pour moi de remercier les personnes qui m'ont accompagné au long de ces trois ans de thèse et sans qui ce travail n'aurait pas pu être mené à bout.

En premier lieu, je tiens à remercier chaleureusement Valérian Mannoni et Valentin Savin, mes encadrants aux CEA-LETI. Merci pour vos conseils, votre expertise et votre bienveillance. Vous avez toujours su m'expliquer pédagogiquement les notions délicates rencontrées et orienter mes travaux de recherche tout en me laissant un champ d'exploration suffisant. Je remercie également Inbar Fijalkow du laboratoire ETIS, qui a su parfaitement diriger cette thèse. Merci pour ton suivi constant de mes travaux ainsi que pour ta grande bienveillance malgré la distance.

Je tiens également à remercier mon jury de thèse, la présidente Catherine Douillard, les rapporteurs Camille Leroux et Didier Le Ruyet, et les examinateurs Charly Poulliat et Elsa Dupraz, qui m'ont fait l'honneur de juger mon travail et qui ont tous su proposer des discussions très intéressantes.

Merci aussi à l'ensemble de l'équipe du CEA-LETI, qui m'a accueillie avec bienveillance tout au long de ces trois ans. Merci à tous les permanents, y compris Benoît D., David, Mattia, Nicolas C., Christophe, Benoît M., François, Johan, Mohamed, Josua, Jean-Baptiste, Mickaël. Merci aussi aux non permanents, doctorants et stagiaires, Gaston, Thomas, Tomàs, Yaya, Khaled, Alexis, Lionel, Antoine, Marion, Esteban, Mustafa, Nihal, Asma, Fatima, et tout ceux que j'oublie. Un grand merci à mes camarades de bureau Paul, Aleksandra et Ibrahim pour leur gentillesse et pour tous ces échanges. Je remercie chaleureusement Oscar pour toutes les discussions au bureau lors de la dernière année de thèse ainsi que pour avoir organisé mon cadeau de départ du CEA, et Marie, pour toutes les discussions, les innombrables pauses au CEA et les sorties au bar. Je voulais ensuite remercier l'équipe d'ETIS pour leur accueil chaleureux lors de mes quelques passages. Un merci tout d'abord à Guillaume pour les discussions techniques très enrichissantes. Merci aussi aux doctorants, Louis, Théo, Claire, Clara, Etienne, Arnaud, Mouktar, Hajar, Rony et tout ceux que j'oublie. Enfin, un grand merci à Laure et Alexandre pour m'avoir intégré à ETIS malgré ma faible présence.

Je tiens ensuite à remercier mes amis de classes préparatoires, Vincent, Violette, Iñigo et Grégoire pour ces nombreux week-ends qui m'ont permis de relâcher la pression. Merci aussi à Étienne C., Jean-Baptiste, Thomas et Maël pour m'avoir permis de profiter de faire de la musique et de me changer les idées. Également merci à mes amis sur Grenoble, Landry, Amélie, Paola, Étienne M., Eloïse, Paul, Vicky, Nicolas, Amandine, Carolina, Armando, Ana B., Quentin, Ana L., Joaquim, Diego R., Alexis et Agathe pour tous les moments chouettes partagés ensembles. Je remercie particulièrement Timothée, Diego C. et Maël pour toutes ces années de collocation vécues ensemble. Enfin, un très grand merci à mes parents et mon petit frère Yannis qui m'ont toujours soutenu et inspiré à poursuivre des études.

Résumé

Les communications sans fil, déjà très présentes dans notre société, soulèvent de nouveaux défis dans le cadre du déploiement de l'Internet des Objets (IoT) tels que le développement de nouvelles méthodes de décodage au niveau de la couche physique permettant d'assurer de bonnes performances pour la transmission de messages courts. En particulier, les codes LDPC (Low Density Parity Check) sont une famille de codes correcteurs d'erreurs très connus pour leurs excellentes performances asymptotiques lorsqu'ils sont décodés par l'algorithme de propagation de croyance (BP, pour Belief Propagation, en anglais). Cependant, la capacité de correction de l'algorithme BP se retrouve fortement dégradée pour les codes LDPC courts. Ainsi, cette thèse porte sur l'amélioration du décodage des codes LDPC courts, grâce notamment à des outils d'apprentissage automatique, tels que les réseaux de neurones.

Après avoir introduit les notions et caractéristiques des codes LDPC et du décodage BP, ainsi que la modélisation du BP par un réseau de neurones récurrent (BP-Recurrent Neural Network ou BP-RNN), nous développons de nouvelles méthodes d'entraînement afin de spécialiser le décodeur BP-RNN sur des motifs d'erreurs partageant des propriétés structurelles similaires. Ces approches de spécialisation sont associées à des architectures de décodage composées de plusieurs BP-RNNs spécialisés, où chaque BP-RNN est entraîné à corriger un type différent de motif d'erreurs (diversité de décodage). Nous nous intéressons ensuite au post-traitement du BP (ou du BP-RNN) avec un décodage par statistiques ordonnées (Ordered Statistics Decoding ou OSD) afin de se rapprocher de la performance du décodage par maximum de vraisemblance. Pour améliorer les performances du post-traitement, nous optimisons son entrée grâce à un neurone simple, puis nous introduisons une stratégie de décodage pour un post-traitement par OSD multiples. Il est alors montré que cette stratégie tire efficacement partie de la diversité de ses entrées, fournissant ainsi un moyen efficace de combler l'écart avec le décodage par maximum de vraisemblance.

Abstract

Wireless communications, already very present in our society, still raise new challenges as part of the deployment of the Internet of Things (IoT) such as the development of new decoding methods at the physical layer ensuring good performance for the transmission of short messages. In particular, Low Density Parity Check (LDPC) codes are a family of error correcting codes well-known for their excellent asymptotic error correction performance under iterative Belief Propagation (BP) decoding. However, the error correcting capacity of the BP algorithm is severely deteriorated for short LDPC codes. Thus, this thesis focuses on improving the decoding of short LDPC codes, thanks in particular to machine learning tools such as neural networks.

After introducing the notions and characteristics of LDPC codes and BP decoding, as well as the modeling of the BP algorithm by a Recurrent Neural Network (BP-Recurrent Neural Network or BP-RNN), we develop new training methods specializing the BP-RNN on decoding error events sharing similar structural properties. These specialization approaches are subsequently associated decoding architectures composed of several specialized BP-RNNs, where each BP-RNN is trained to decode a specific kind of error events (decoding diversity). Secondly, we are interested in the post-processing of the BP (or the BP-RNN) with an Ordered Statistics Decoding (OSD) in order to close the gap the maximum likelihood (ML) decoding performance. To improve the post-processing performance, we optimize its input thanks to a single neuron and we introduce a multiple OSD post-processing decoding strategy. We then show that this strategy effectively takes advantage of the diversity of its inputs, thus providing an effective way to close the gap with ML decoding.

Table des matières

Remerciements iii

Résumé v

Abstract vii

Listes des Acronymes xix

Introduction xxi

1 Codes LDPC et réseaux de neurones 1
- 1.1 Codes LDPC et décodage par passage de messages . . . 2
  - 1.1.1 Définition d'un code LDPC . . . 2
  - 1.1.2 Décodage par passage de messages . . . 3
  - 1.1.3 Construction des codes LDPC . . . 8
  - 1.1.4 Ensembles absorbants . . . 10
  - 1.1.5 Performances des codes LDPC courts décodés par BP . . . 12
- 1.2 Réseaux de neurones . . . 14
  - 1.2.1 Définition d'un neurone . . . 15
  - 1.2.2 Réseau de neurones à propagation avant . . . 16
  - 1.2.3 Réseau de neurones récurrent . . . 17
  - 1.2.4 Entraînement d'un réseau de neurones . . . 19
- 1.3 Décodage LDPC assisté par réseaux de neurones . . . 23
  - 1.3.1 BP neuronal . . . 23
  - 1.3.2 Entraînement standard du BP neuronal . . . 29
  - 1.3.3 Autres méthodes de décodage par approche neuronale . . . 32
- 1.4 Conclusion . . . 35

2 Spécialisation du BP-RNN et diversité de décodage 37
- 2.1 Motivation de la spécialisation . . . 38
- 2.2 BP-RNNs spécialisés . . . 39
  - 2.2.1 Classification des supports des motifs d'erreurs . . . 40
  - 2.2.2 Jeu d'entraînement spécialisé . . . 41
  - 2.2.3 Architecture parallèle de BP-RNNs . . . 42
  - 2.2.4 Sélection optimale par complémentarité . . . 43

2.3 Résultats FER 45
2.3.1 Paramètres de simulation 45
2.3.2 BP-RNNs spécialisés 46
2.3.3 MS-RNNs spécialisés 49
2.3.4 Impact de la configuration de poids 50
2.3.5 BP-RNNs entraînés avec une fonction de coût multiple 51
2.3.6 Analyse des classes sélectionnées 52
2.4 Conclusion 54

3 Diversité de BP-RNNs spécialisés sur des ensembles absorbants 57
3.1 Les ensembles absorbants 58
3.2 Diversité de BP-RNNs spécialisés sur des ensembles absorbants 59
3.2.1 Algorithme de recherche des ensembles absorbants 59
3.2.2 Classification des ensembles absorbants 63
3.2.3 Jeu d'entraînement spécialisé 64
3.3 Sélection des BP-RNNs et architecture de décodage 66
3.3.1 Sélection sous-optimale d'une diversité de BP-RNNs 66
3.3.2 Architecture série d'une diversité de BP-RNNs 68
3.3.3 Métriques de latence et de complexité 68
3.4 Résultats numériques 71
3.4.1 Paramètres de simulation 71
3.4.2 Nombre maximal d'itérations pour l'entraînement et le test 72
3.4.3 Impact du SNR d'entraînement 74
3.4.4 Sélection d'une diversité de BP-RNNs 75
3.4.5 Performances FER 77
3.4.6 Évaluation de la complexité et de la latence 81
3.5 Conclusion 84

4 Post-traitement OSD pour le décodage BP/BP-RNN 87
4.1 OSD 88
4.1.1 Principe de l'OSD 88
4.1.2 OSD intégré dans le BP 90
4.1.3 Post-traitement OSD du BP 91
4.1.4 Réduction de complexité 93
4.2 Post-traitement OSD d'une diversité de BP-RNNs spécialisés 96
4.2.1 Motivations et proposition d'un post-traitement OSD des BP-RNNs 96
4.2.2 Résultats en termes de FER et de complexité 97
4.3 BP-RNNs entraînés avec une fonction de coût focale 102
4.3.1 Motivations et principe 103
4.3.2 Résultats en termes FER 104
4.4 Amélioration du post-traitement OSD du BP 106
4.4.1 Motivations et solutions proposées 106

4.4.2 LLR accumulé et optimisé pour le post-traitement OSD 108
4.4.3 Post-traitement OSD multiples 110
4.4.4 Résultats en termes de FER 111
4.5 Conclusion 118

5 Conclusions et perspectives **119**

Annexe A Étude des hyper-paramètres d'entraînement du BP-RNN **A-1**
A.1 Méthode de descente de gradient A-1
A.2 Pas d'apprentissage initial A-1
A.3 Initialisation des poids A-2

Annexe B Autres approches de spécialisation du BP-RNN **A-5**
B.1 Entraînement sur les échecs du BP A-5
B.2 Entraînement sur les classes les plus présentes dans les échecs du BP A-6

Annexe C Analyse des poids optimisés pour le post-traitement OSD du BP **A-9**
C.1 Poids optimisés A-9

Table des figures

1.1 Exemple d'une matrice de parité et de son graphe de Tanner associé. 3
1.2 Illustrations des étapes check-pass, data-pass et a posteriori du décodage BP. 5
1.3 Exemple d'un cycle de taille 4, en rouge. 9
1.4 Exemple de sous-graphe étendu. 10
1.5 Exemple d'un ensemble absorbant de 4 noeuds de donnée. 12
1.6 FER de codes LDPC courts décodés par BP, avec $I = 25,250$ itérations. . . 13
1.7 Représentation d'un neurone standard. 15
1.8 Exemple d'un réseau de neurones FF. 17
1.9 Exemple d'un RNN, sous sa représentation déroulée et non déroulée. . . . 18
1.10 Schéma de principe d'une époque d'entraînement. 20
1.11 Coût d'entraînement et de validation en fonction des époques 21
1.11 Connexions entre les étages du BP neuronal du code figure 1.1. 25
1.12 BP-RNN non déroulée. 28
1.13 Résultats FER du BP et du BP-RNN standard, avec $I_{\text{test}} = 10$ itérations. . . 32

2.1 Exemple de deux ensembles $\mathcal{V}$, avec $\mathbf{card}(O(\mathcal{V})) = \mathbf{card}(\mathcal{E}(\mathcal{V})) = 3$. . . 41
2.2 Architecture de décodage parallèle. 43
2.3 Motifs d'erreurs de la classe $3 - (5, 2, [(1, 2), (1, 2), (3, 0)])$, composés de deux motifs indépendants de la classe $2 - (2, 2, [(1, 2), (1, 2)])$. 44
2.3 Résultats FER des BP-RNNs spécialisés mis en parallèle. 48
2.4 Résultats FER des MS-RNNs spécialisés mis en parallèle. 49
2.5 Résultats FER du Code-3, pour différentes configurations de poids. 52
2.6 Résultats FER du Code-3, BCE vs Multi-BCE. 53
2.7 Les sous-graphes des classes correspondants aux BP-RNNs de $\mathcal{D}_4^{\text{opt}}$ pour le Code-1. Les types absorbants de chaque classe étant différents, les profils de connexions $\mathfrak{P}$ ne sont pas détaillés pour simplifier la notation. 54
2.8 Résultats FER du Code-1. 55

3.1 Exemple de graphe biparti enraciné $\mathcal{G}_n$. Les connexions prises en comptes pour le choix $\mathcal{A} = \{n_1, n_2, n_3, n_4\}$ sont indiquées en bleu. 62

3.2 Exemple d'ensembles absorbants de type 4-$(2,5)$. 63
3.3 Profils de poids pour différents décodeurs BP-RNNs (Code-1, SNR = 5 dB). 65
3.4 Architecture série. 68
3.4 Impact du paramètre d'entraînement I_{ent} sur la performance en termes de FER, pour les décodeurs BP-RNNs utilisant $I_{\text{test}} = 25$ (Code-1). 74
3.5 Résultats FER du Code-1, pour différents SNRs d'entraînement. 75
3.6 SNR pour une cible FER = 10^{-4}, en fonction de Z. 76
3.7 Résultats FER, BP-RNNs spécialisés. 78
3.8 Résultats FER, MS-RNNs spécialisés. 80
3.9 Complexité moyenne de décodage en termes du nombre moyen d'itérations. 82
3.10 Latence moyenne de décodage en termes du nombre moyen d'itérations. . 84

4.1 OSD-0. 89
4.2 OSD intégré dans le BP. 91
4.3 Post-traitement OSD du BP, avec la fiabilité d'un bit donnée par la valeur absolue de son LLR a posteriori à la dernière itération de décodage du BP. 92
4.4 Exemples typiques d'un phénomène d'oscillation (Code-2, 25 itérations du BP à 3.5 dB, non convergence vers un mot de code). 93
4.5 Nombre d'erreurs vs positions dans les K premiers éléments de $\tilde{\mathbf{c}}_{\text{tri}}^{(\text{S})}$ (Code-2, BP($I_{\text{test}} = 25$), 3.5 dB, 10000 échecs de décodage). 94
4.6 Inversions autorisées dans $\mathbf{c}_{\text{tri}}$ pour l'OSD-2. 95
4.7 Post-traitement OSD-p d'une diversité de BP-RNNs $\mathcal{D}_Z$ organisée en une archietcture de décodage parallèle. 97
4.8 Résultats FER (BP-RNNs) avec l'étape de post-traitement OSD ($p = 0, 1$). . 98
4.9 Résultats FER (BP-RNNs) avec l'étape de post-traitement OSD pour le Code-2 ($p = 1, 2$). 100
4.10 Résultats FER avec l'étape de post-traitement OSD, MS-RNNs spécialisés vs BP-RNNs spécialisés 101
4.11 Fonction de coût focale (4.5) en fonction de la valeur des LLRs a posteriori. 104
4.12 Résultats FER avec l'étape de post-traitement OSD et un entraînement des décodeurs BP-RNNs($I_{\text{test}} = 25$) avec la fonction de coût focale ($\gamma = 10$). . . 105
4.13 Nombre d'erreurs résiduelles sur 10000 non convergences du BP vers un mot de code après post-traitement OSD (Code-2, $I_{\text{test}} = 25$, SNR = 3.5 dB). 107
4.14 Neurone calculant $\acute{L}_n^{(\text{NS})}$ 108
4.15 Fonction de coût focale avec $\gamma = 10$ selon les $\tilde{\mathbf{L}}^{(i)}$, $i \in [0, I_{\text{test}}]$ (Code-2, $I_{\text{test}} = 25$, SNR = 3.5 dB). 109
4.16 BP-$\mathcal{L}_Z$-OSD-p 111
4.17 Résultats FER pour le Code-2, $\tilde{\mathbf{L}}^{(\text{S})}$ vs $\tilde{\mathbf{L}}^{(\text{NS})}$. 114
4.18 Résultats FER pour le Code-2, $\mathcal{D}_3$. 115
4.19 Résultats FER pour le Code-2, $\mathcal{D}_3$ vs $\mathcal{L}_3$. 115
4.20 Résultats FER pour le Code-5 116
4.21 Résultats FER pour le Code-6 117

A.1 Résultats FER du Code-3, Adam vs RMSprop. A-2
A.2 Résultats FER du Code-3, selon le pas d'apprentissage initial. A-3
A.3 Résultats FER du Code-3, selon l'initialisation des poids du BP-RNN. . . . A-3

B.1 Chaîne d'entraînement en cascade. A-6
B.2 Résultats FER du Code-1, D_1^F. A-7
B.3 Résultats FER du Code-2, $\mathcal{D}_{30}^{\text{Pres}}$. A-8

C.1 Poids optimisés $w^{(i)}$ vs l'itération i ($i \in [0, I_{\text{test}}]$). A-10
C.2 Nombre d'erreurs résiduelles sur 10000 échecs du BP($I_{\text{test}} = 25$) après un post-traitement OSD. A-11

Liste des tableaux

1.1 Nombre d'additions et de multiplications pour une itération du BP. 7
1.2 Paramètres des codes LDPC évalués 12

2.1 $P_{\text{err}}(E)$ vs E. 38
2.2 Paramètres des codes LDPC évalués. 45
2.3 Nombre de classes obtenues ν-$(\omega, \varepsilon, \mathfrak{B})$ selon ν. 46
2.4 Paramètres Keras. 46

3.1 Nombre de types étendus (TE) et d'ensembles absorbants (EA). 64
3.2 Paramètres Keras. 72
3.3 Classes ν-$(\omega, \varepsilon, P_c)$ sélectionnées et leurs décodeurs BP-RNNs associés. . . 77
3.4 Complexité de décodage (en termes du nombre de mises à jour du check-pass) pour le Code-2. 83
3.5 Nombre de poids des différentes architectures de décodage évaluées. . . . 85

4.1 Paramètres des codes LDPC 112
4.2 Paramètres Keras 112
4.3 Récapitulatif des différents décodeurs comparés par figure 113

A.1 Paramètres Keras A-1

Listes des Acronymes

Note : les abréviations utilisées correspondent aux termes anglais, car largement répandues au sein de la communauté.

IoT	Internet of Things	Internet des objets
5G	Fifth generation of cellular mobile communications	Cinquième génération des standards pour la téléphonie mobile
5G-NR	5G New Radio	5G Nouvelle Radio
DVB-T	Digital Video Broadcasting	Radiodiffusion Vidéo Numérique
BCH	Bose-Chaudhuri-Hocquenghem codes	Codes de Bose-Chaudhuri-Hocquenghem
LDPC	Low Density Parity Check codes	Codes à matrice de parité creuse
PEG	Progressive Edge Growth	Algorithme par croissance d'arêtes progressives
BP	Belief Propagation	Algorithme par propagation de croyance
MS	MinSum	Minimum-sommation
NMS	Normalized MinSum	Minimum-sommation normalisé
OMS	Offset MinSum	Minimum-sommation avec Offset
NN	Neural Network	Réseau de neurones
NN-FF	Neural Network-Feed Forward	Réseau de neurones à propagation avant
RNN	Reccurent Neural Network	Réseau de neurones récurrent
BP-FF	BP-Feed Forward	BP modélisé par un réseau de neurones à propagation avant

BP-RNN	BP-Reccurent Neural Network	BP modélisé par un réseau de neurones récurrent
MS-RNN	MS-Reccurent Neural Network	MS modélisé par un réseau de neurones récurrent
FAID	Finite-Alphabet Iterative Decoders	Décodeur itératif à alphabet fini
PB-NBP	Pruning Base-Neural BP	BP neuronal basé sur de l'élagage
AED	Automorphism Ensemble Decoding	Décodage par automorphisme d'ensembles
BCE	Binary Cross Entropy	Entropie Croisée Binaire
FL	Focal Loss	Fonction de coût focale
ML	Maximum Likelihood	Maximum de vraisemblance
BER	Bit Error Rate	Taux d'erreur binaire
FER	Frame Error Rate	Taux d'erreur paquet
AWGN	Additive White Gaussian Noise	Bruit additif blanc gaussien
BSC	Binary Symetric Channel	Canal binaire symétrique
BPSK	Binary Phase Shift Keying	Modulation binaire à décalage de phase
SNR	Signal-to-Noise Ratio	Rapport signal à bruit
LLR	Log Likelihood Ratio	Rapport de vraisemblance logarithmique
AS	Absorbing Set	Ensemble absorbant
AS-DFS	Absorbing Set-Depth First Search	Algorithme de parcours en profondeur pour la recherche d'ensembles absorbants
OSD	Ordered Statistics Decoding	Décodage par statistiques ordonnées
CCSDS	Consultative Committee for Space Data Systems	Comité consultatif des systèmes de données spatiales

Introduction

L'Internet des Objets (IoT) se positionne aujourd'hui au centre de l'évolution numérique : plus de 9 milliards d'objets sont déjà connectés, 30 milliards le seront d'ici 2025 [1]. Le principe de l'IoT réside dans la connection de nombreux objets du quotidien via Internet. Cette notion d'objets connectés a ainsi de nombreuses applications déjà prévues pour des appareils numériques tels que les appareils mobiles, les ordinateurs, les capteurs, etc. Toutefois, ce principe repose aussi sur la découverte de nouvelles applications non attendues. De nombreuses exigences sont attendues vis-à-vis des systèmes de communication associés à l'IoT. En particulier, des communications sporadiques avec des messages de petite taille sur la voie montante sont requises pour ces systèmes. L'association de l'ensemble de ces contraintes a conduit à l'émergence de systèmes radios spécifiques à l'IoT, et ses évolutions actuelles dans la cinquième génération des standards pour la téléphonie mobile (5G). Cependant, les besoins de l'IoT ne sont pas tous simultanément satisfaits avec ces technologies. En particulier, il a été mis en évidence qu'une nouvelle couche physique, avec des codes correcteurs d'erreurs ayant de bonnes performances pour les petits paquets doit être proposée [2]. De ce fait, des progrès importants ont été réalisés dans la compréhension des limites de codage en taille courte [3]. Néanmoins, de nombreux défis restent à relever dans la conception de codes courts et d'algorithmes de décodage efficaces [4].

Les codes Low Density Parity Check (LDPC) [5] sont une famille de code correcteurs d'erreurs définis par des graphes bipartis creux [6], utilisés dans différents standards tels que 5G Nouvelle Radio (5G-NR) ou Radiodiffusion Vidéo Numérique (DVB) et étudiés pour être déployés dans la couche physique de l'IoT. Ils sont bien connus pour leurs excellentes performances en régime asymptotique (taille de bloc très importante), pouvant approcher la capacité canal de Shannon lorsqu'ils sont décodés itérativement par l'algorithme par propagation de croyance (Belief Propagation ou BP) [7]. En effet, les codes LDPC longs se caractérisent par des graphes bipartis sans cycles courts, auquel cas le décodeur BP permet d'obtenir une estimation des bits codés proche de celle obtenue par l'estimateur du maximum a posteriori [8]. Cependant, la performance du BP se dégrade considérablement pour les codes courts à cause de l'apparition de nombreux cycles notamment courts dans le graphe.

Pour remédier à ce problème, le domaine de l'intelligence artificielle et tout particulièrement des réseaux de neurones (Neural Networks ou NNs), a été rapproché de celui du décodage itératif par BP. Plus précisément, les auteurs de [9] ont été les premiers à remarquer que le BP se prête naturellement à la modélisation en un réseau de neurones, appelé BP neuronal. La topologie de ce réseau copie ainsi le processus de décodage itératif BP, en y introduisant des poids sur les arêtes du graphe de Tanner. Ces poids sont alors optimisés lors de l'entraînement du BP neuronal. De plus, le BP neuronal peut être implémenté sous

la forme soit d'un réseau de neurones à propagation avant (BP Feed-Forward ou BP-FF) soit d'un réseau de neurones récurent (BP-RNN). Il a ensuite été montré dans [10] que le BP-RNN est capable d'obtenir de meilleures performances que le BP standard pour des codes Bose-Chaudhuri-Hocquenghem (BCH) courts. Le BP-RNN ne permet pas cependant de combler l'écart de performance avec le décodage par maximum de vraisemblance (Maximum Likelihood ou ML), notamment pour les codes LDPC courts. Par conséquent, les décodeurs BP-FF et BP-RNN ont servi de base à de nombreux travaux, proposant entre autres de nouvelles versions du BP neuronal plus élaborées [11–15], et/ou d'apprendre de nouvelles règles de calcul pour remplacer celles effectuées lors du BP standard ou BP neuronal [15–20]. De plus, le MinSum (MS), le MS Normalisé (NMS) et l'Offset MS (OMS), des variantes moins complexes du BP mais moins performantes, ont aussi été déclinées en plusieurs versions neuronales [10, 21–23] afin d'améliorer leurs performances respectives. Enfin, dans un souci de complexité et d'implémentation matérielle pratique, l'ordonnancement des messages lors du décodage itératif est intégré dans des versions neuronales du BP/MS et dans leurs méthodes d'apprentissage [24–27].

Cette thèse se concentre sur le décodage des codes LDPC courts aidé par réseaux de neurones, avec comme objectif d'en améliorer la performance dans la région du *waterfall*. Une difficulté critique rencontrée par le décodage BP d'un code LDPC est la présence de structures particulières dans le graphe biparti, empêchant l'algorithme de décodage de converger. En particulier, pour des codes courts, ces structures possèdent des tailles comparables au nombre d'erreurs que le code doit corriger, ce qui peut entraîner une dégradation significative des performances de correction d'erreur dans la région du *waterfall*. Un problème du BP-neuronal de [10] provient alors de son jeu d'entraînement. En effet, il ne permet pas de représenter correctement les structures dégradant la performance du BP dans la région du *waterfall* pour les codes LDPC courts. Par conséquent, un BP neuronal entraîné avec ce jeu ne peut pas apprendre à corriger les erreurs induites par ces structures et des gains limités sont observés par rapport au BP. Une autre approche intéressante pour améliorer la capacité de correction des codes LDPC courts est la concaténation du BP avec un décodage par statistiques ordonnées (Ordered Statistics Decoding ou OSD) [28], un décodeur non neuronal. L'OSD est en effet connu pour être capable d'estimer la performance du décodage ML, toutefois au prix d'une complexité calculatoire élevée. Afin de remédier à ce problème, l'OSD peut aussi être utilisé comme un outil de post-traitement, exploitant la sortie souple d'un décodeur itératif (tel que le BP) lorsque ce dernier ne converge pas vers un mot de code [29–32]. L'enjeu réside alors à calculer une entrée optimale pour l'OSD à partir de la sortie souple du BP (ou du BP-RNN), notamment en utilisant des réseaux de neurones. Nos recherches se divisent donc en deux axes principaux : l'élaboration de nouvelles stratégies d'entraînement du BP-RNN et la combinaison d'un décodeur itératif comme le BP ou le BP-RNN avec un décodeur non neuronal.

Contributions de la thèse

Les principales contributions développées dans nos deux axes de recherches sont les suivantes :

- ***Diversité de décodeurs BP-RNNs spécialisés [IC1, J1] :*** Dans le cadre de notre premier axe de recherche, nous proposons une première méthode d'entraînement spécialisant le BP-RNN dans le décodage des petits motifs d'erreurs pour des codes LDPC courts avec une faible capacité de correction (rendement élevé). Plus précisément, les motifs d'erreurs sont tout d'abord énumérés exhaustivement, puis classifiés selon leur sous-graphe induit. Un BP-RNN est ensuite entraîné spécifiquement pour chaque classe. Une diversité de décodeurs BP-RNNs est ainsi obtenue, dans laquelle chaque BP-RNN est spécialisé dans le décodage de motifs d'erreurs partageant des propriétés structurelles similaires dans leur sous-graphe. Cette approche, notamment motivée par [16, 17], permet ainsi de compter sur plusieurs décodeurs neuronaux au lieu d'un unique pour améliorer la performance de décodage. Pour traiter des codes LDPC avec une meilleure capacité de correction, nous adaptons cette première méthode d'entraînement en concentrant notre attention sur les motifs d'erreurs avec un support d'ensemble absorbant [33]. Ces derniers sont en effet responsables de nombreux échecs de décodage dans la région du *waterfall* des codes LDPC courts. Nous proposons ainsi un algorithme énumérant de manière efficace les ensembles absorbants d'une taille donnée. Ils sont ensuite classifiés selon leurs propriétés structurelles. Enfin, une diversité de BP-RNNs spécialisés sur les ensembles absorbants est obtenue en entraînant un BP-RNN par classe.
- ***Procédure de sélection et architectures de décodage d'une diversité de BP-RNNs [IC1, J1] :*** Notre seconde contribution traite de l'organisation d'une diversité de BP-RNNs spécialisés en une architecture de décodage. Pour des raisons de complexité calculatoire et matérielle, nous proposons tout d'abord de limiter le nombre de décodeurs BP-RNNs considérés en sélectionnant uniquement les plus complémentaires en termes de probabilités d'erreurs conjointes. Nous proposerons deux architectures de décodage, dans lesquelles les décodeurs BP-RNNs sélectionnés sont exécutés soit en parallèle soit en série.
- ***Post-traitement OSD de BP-RNNs spécialisés [J1] :*** Afin de se rapprocher de la performance de décodage ML, nous combinons notre diversité de BP-RNNs spécialisés sur les ensembles absorbants avec une étape de post-traitement OSD, appliquée uniquement si aucun des BP-RNNs sélectionnés ne converge vers un mot de code. La motivation principale derrière cette étape de post-traitement OSD est de bénéficier de la diversité apportée par l'utilisation de plusieurs décodeurs BP-RNNs spécialisés. Nous montrons en effet que le gain de codage apporté par l'utilisation de plusieurs décodeurs BP-RNNs est amplifié par l'utilisation du post-traitement OSD, résultant en une amélioration significative des performances de correction d'erreurs.
- ***Post-traitement OSD du BP [IC2] :*** Nous abordons également la combinaison du BP

standard avec l'OSD, dans le but de proposer une solution alternative (et donc un autre compromis performance/complexité) à la diversité de BP-RNNs associée avec le post-traitement OSD. Pour réaliser ceci, une nouvelle entrée adaptée au post-traitement OSD est premièrement calculée à partir des sorties du BP, grâce à l'optimisation d'un simple neurone. Pour se rapprocher de la performance ML, une stratégie de décodage pour post-traitement OSD multiples est également présentée. Il est ensuite montrée que cette stratégie permet de combler l'écart à la performance ML pour les codes LDPC courts, pour une complexité maîtrisée.

Le manuscrit est organisé comme suit. Le premier chapitre introduit les notions fondamentales liées aux codes LDPC et aux réseaux de neurones, puis décrit le BP-RNN. Le second chapitre est dédié à la création d'une diversité de BP-RNNs spécialisés sur les petits motifs d'erreurs, ainsi qu'à l'architecture parallèle correspondante. L'adaptation de cette méthode sur les ensembles absorbants est expliquée dans le chapitre 3. Nous y détaillons en particulier l'algorithme énumérant les ensembles absorbants, une procédure de sélection des BP-RNNs spécialisés, et l'architecture série. Le chapitre 4 étudie la combinaison du post-traitement OSD avec une diversité de BP-RNNs, puis avec le BP. Enfin, le chapitre 5 conclut le manuscrit.

Liste des publications

Ces travaux ont permis la publication de :

Journaux Internationaux

[J1] J. Rosseel, V. Mannoni, I. Fijalkow and V. Savin, "Decoding Short LDPC Codes via BP-RNN Diversity and Reliability-Based Post-Processing," in *IEEE Transactions on Communications*, vol. 70, no. 12, pp. 7830-7842, Dec. 2022, doi : 10.1109/TCOMM.2022.3218821.

Conférences Internationales

[IC2] J. Rosseel, V. Mannoni, V. Savin, and I. Fijalkow, "Sets of complementary LLRs to improve OSD post-processing of BP decoding," *12th International Symposium on Topics in Coding (ISTC)*, 2023.

[IC1] J. Rosseel, V. Mannoni, V. Savin, and I. Fijalkow, "Error structure aware parallel BP-RNN decoders for short LDPC codes," *11th International Symposium on Topics in Coding (ISTC)*, pp. 1–5, 2021.

Conférences Nationales

[NC2] J. Rosseel, V. Mannoni, V. Savin, and I. Fijalkow, "Post-traitement OSD pour le décodage BP basé sur des ensembles de LLRs complémentaires", *GRETSI* 2023

[NC1] J. Rosseel, V. Mannoni, V. Savin, et I. Fijalkow, "Décodeurs BP-RNNs mis en parallèle et spécialisés dans le décodage de codes LDPC courts", *GRETSI* 2022

Chapitre 1

Codes LDPC et réseaux de neurones

Contenu du Chapitre 1

1.1 Codes LDPC et décodage par passage de messages 2
1.1.1 Définition d'un code LDPC 2
1.1.2 Décodage par passage de messages 3
1.1.3 Construction des codes LDPC 8
1.1.4 Ensembles absorbants 10
1.1.5 Performances des codes LDPC courts décodés par BP 12
1.2 Réseaux de neurones 14
1.2.1 Définition d'un neurone 15
1.2.2 Réseau de neurones à propagation avant 16
1.2.3 Réseau de neurones récurrent 17
1.2.4 Entraînement d'un réseau de neurones 19
1.3 Décodage LDPC assisté par réseaux de neurones 23
1.3.1 BP neuronal 23
1.3.2 Entraînement standard du BP neuronal 29
1.3.3 Autres méthodes de décodage par approche neuronale 32
1.4 Conclusion 35

Ce chapitre a pour principal objectif d'introduire les notions essentielles du décodage des codes LDPC courts aidé par réseaux de neurones et qui seront utilisées tout au long de ce manuscrit. Dans un premier temps, la définition des codes LDPC est rappelée, suivi de la description de l'algorithme de décodage BP. De plus, les structures problématiques pour la construction et le décodage des codes LDPC courts sont présentées. Ensuite, les concepts propres aux réseaux de neurones et leurs architectures standards sont introduits. Enfin, les approches de [10, 11, 17], modélisant le BP par un réseau de neurones, sont expliquées.

1.1 Codes LDPC et décodage par passage de messages

1.1.1 Définition d'un code LDPC

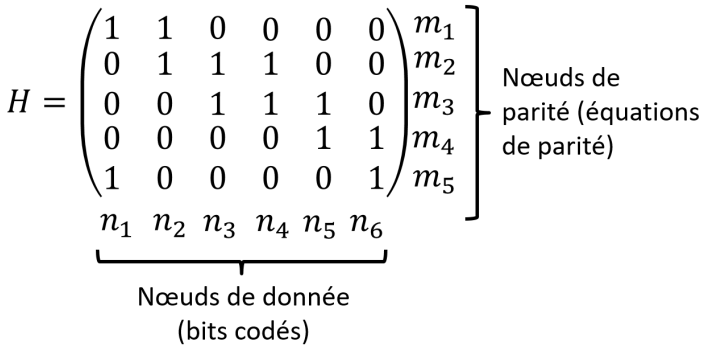
Une classe importante de codes correcteurs d'erreurs en bloc, linéaires est celle des codes LDPC (pour Low Density Parity Check, en anglais), introduits par Gallager en 1962 [5]. Un code LDPC(N, K) est un code linéaire avec N bits codés et K bits d'information, dont la matrice de parité H possède une faible densité de 1 (comparé au nombre de 0). Cette notion de faible densité n'est définie en toute rigueur qu'asymptotiquement, pour une famille de codes dont la longueur N tend vers l'infini, en demandant que le nombre de 1 par ligne ou par colonne de H soit toujours inférieur à une constante donnée. De plus, nous notons par M le nombre d'équations de parité, correspondant aux lignes de H. H est donc de taille $M \times N$. Le nombre de bits d'information K est alors donné par $K = N - \text{rang}(H)$. Si H est une matrice de rang plein, $\text{rang}(H) = M$ et $K = N - M$. Comme tout les codes correcteurs en bloc linéaires, un vecteur $\mathbf{c} = [c_1, \ldots, c_N]$ est un mot de code si et seulement il vérifie l'équation de parité suivante :

$$H\mathbf{c}^T = \mathbf{0}_M \tag{1.1}$$

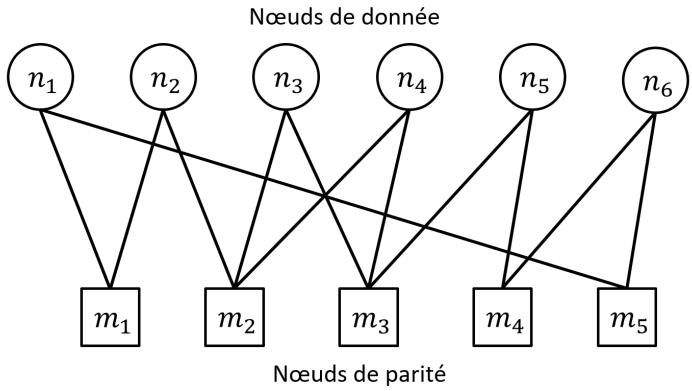
où $\mathbf{c}^T$ est le vecteur transposé de $\mathbf{c}$ et $\mathbf{0}_M$ est le vecteur colonne de taille M uniquement composé de zéro. Enfin, le rendement de codage d'un code LDPC se définit comme $R_c = \frac{K}{N}$.

La matrice de parité H d'un code LDPC se décrit généralement sous la forme d'un graphe biparti. En effet, en 1981, Tanner remarque que cette description graphique s'applique avec succès aux codes LDPC [6] et qu'elle permet également de décrire le décodage probabiliste introduit par Gallager comme un algorithme par passage de messages dans le graphe biparti (algorithme de décodage par propagation des croyances, que nous présenterons dans la sous-section suivante).

Un graphe biparti est un graphe contentant deux types de noeuds tel que deux nœuds du même type ne soient jamais connectés. Pour un code LDPC, les deux ensembles de noeuds sont respectivement les M nœuds de parité correspondant aux équations de parité définies par les lignes de la matrice H et les N nœuds de donnée correspondant aux bits codés (ou de manière équivalente aux colonnes de H). Un nœud de parité $m \in \{1, \ldots, M\}$ est connecté à un nœud de donnée $n \in \{1, \ldots, N\}$ si et seulement si l'élément dans la position (m, n) de



(a) Matrice de parité.



(b) Graphe de Tanner.

Figure 1.1 – Exemple d'une matrice de parité et de son graphe de Tanner associé.

la matrice H est égal à 1. En d'autres termes, la matrice d'incidence du graphe biparti est la matrice de parité du code.

La figure 1.1 donne un exemple d'une matrice de parité et de son graphe de Tanner, avec $N = 6$, $M = 5$. Le rang de H est ici égal à 4, ce qui implique $K = 2$ et $R_c = 0.33$.

1.1.2 Décodage par passage de messages

1.1.2.1 Algorithme Belief Propagation

L'algorithme usuel de décodage des codes LDPC est l'algorithme de décodage itératif par propagation de croyance (connu comme Belief Propagation, BP) [7]. L'algorithme BP se base sur la représentation de la matrice de parité en graphe de Tanner ; et il peut être vu comme un algorithme de propagation de messages sur le graphe de Tanner associé. Le principe de la propagation de croyance est l'application de la règle de Bayes localement (au niveau

de chaque nœud de parité et de chaque nœud de donnée) et itérativement afin d'estimer les probabilités a posteriori de chaque bit. Les messages transitant par les arêtes du graphe peuvent être soient des probabilités soit des rapports de vraisemblance logarithmiques (LLR, pour Log Likelihood Ratio en anglais). Toutefois, afin d'éviter des problèmes d'instabilité numérique, le BP manipule le plus souvent des LLRs au lieu de probabilités brutes. Nous pouvons rajouter que le BP est qualifié d'algorithme de décodage par passage de message car il respecte la propriété d'un échange extrinsèque d'information [34]. En effet, le calcul du message sortant d'un nœud de parité m et allant vers le nœud de donnée n ne prend pas en compte le message venant de ce nœud n. Cela a un impact sur la performance, mais aussi sur la complexité de l'algorithme.

Avant de décrire plus formellement l'algorithme BP, il convient d'introduire les notations suivantes :

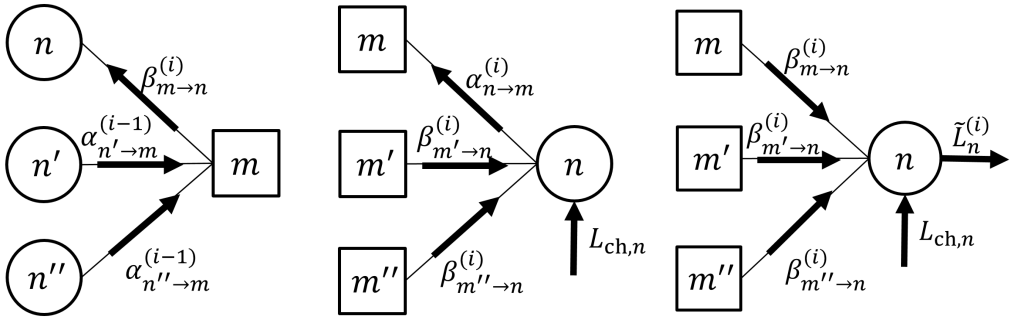
- $n \in \{1, \ldots, N\}$ est le n-ième noeud de donnée (appelé aussi n-ième bit)
- $m \in \{1, \ldots, M\}$ est le m-ième noeud de parité
- $\mathcal{N}(m)$ est l'ensemble de noeuds de donnée connectés au noeud de parité m
- $\mathcal{M}(n)$ est l'ensemble de noeuds de parité connectés au noeud de donnée n
- $\mathbf{c} = [c_1, \ldots, c_N]$ est le mot de code émis
- $\mathbf{y} = [y_1, \ldots, y_N]$ est le signal reçu
- $L_{\text{ch},n}$ est le LLR observé (canal) du bit n, calculer à partir de y_n
- i est une itération du décodage BP
- $\beta_{m\to n}^{(i)}$ est le message allant du noeud de parité m vers le noeud de donnée n, à l'itération i
- $\alpha_{n\to m}^{(i)}$ est le message allant du noeud de donnée n vers le noeud de parité m, à l'itération i
- $\tilde{L}_n^{(i)}$ est le LLR a posteriori (estimé) du noeud de donnée n à l'itération i.

Précisons dès maintenant que tout au long du manuscrit, le modèle de canal utilisé pour les transmissions est le canal à bruit additif blanc gaussien (AWGN pour Additive White Gaussien Noise en anglais). Le rapport signal à bruit (Signal-to-Noise Ratio (SNR) en anglais) se définit alors par $\text{SNR} = -10\log_{10}(\sigma^2)$ dB, où σ^2 est la variance du bruit gaussien. De plus, nous considérerons un canal AWGN à entrée binaire, obtenue par une modulation binaire à décalage de phase (Binary Phase Shift Keying ou BPSK en anglais) du mot de code émis $\mathbf{c}$. Enfin, le signal observé s'obtient avec l'équation suivante :

$$y_n = (1 - 2c_n) + z_n, \quad z_n \sim \mathcal{N}(0, \sigma^2),\ n = 1, \ldots, N, \tag{1.2}$$

Le décodage BP est réalisé comme suit. Tout d'abord, l'information a priori de chaque bit est calculée sous la forme d'un LLR, en fonction du signal reçu du canal $\mathbf{y}$:

$$L_{\text{ch},n} = \log \frac{\Pr(c_n = 0 \mid y_n)}{\Pr(c_n = 1 \mid y_n)} \tag{1.3}$$



(a) Exemple de calcul de $\beta^{(i)}_{m\to n}$ (check-pass).

(b) Exemple de calcul de $\alpha^{(i)}_{n\to m}$ (data-pass).

(c) Exemple de calcul de $\tilde{L}^{(i)}_n$ (a posteriori).

Figure 1.2 – Illustrations des étapes check-pass, data-pass et a posteriori du décodage BP.

Dans le cas du canal gaussien à entrée binaire, le calcul de $L_{\mathrm{ch},n}$ s'écrit :

$$L_{\mathrm{ch},n} = \frac{2}{\sigma^2} y_n \tag{1.4}$$

Les messages partant de noeuds de donnée et en direction des noeuds de parité sont alors initialisés avec les LLRs canal $L_{\mathrm{ch},n}$ correspondant, soit :

$$\alpha^{(0)}_{n\to m} = L_{\mathrm{ch},n},\ n \in [1, N],\ m \in \mathcal{M}(n) \tag{1.5}$$

Le décodage s'effectue ensuite de manière itérative, jusqu'à ce qu'un nombre maximal d'itérations de décodage I soit atteint ou qu'un mot de code soit trouvé. Quatre étapes se succèdent au cours d'une itération $i \in [1, I]$:

1. *Mise à jour des messages allant des noeuds de parité vers les noeuds de donnée (check-pass en anglais)* :

   Pour tous les noeuds de parité m et les noeuds de donnée $n \in \mathcal{N}(m)$, le calcul des $\beta^{(i)}_{m\to n}$ est donné par la formule suivante :

$$\beta^{(i)}_{m\to n} = 2\tanh^{-1}\left(\prod_{n'\in\mathcal{N}(m)\setminus\{n\}} \tanh\left(\frac{\alpha^{(i-1)}_{n'\to m}}{2}\right)\right) \tag{1.6}$$

   Le message $\beta^{(i)}_{m\to n}$ représente le LLR conditionnel du noeud de donnée n connaissant les $\alpha^{(i-1)}_{n'\to m}$ $(n' \in \mathcal{N}(m)\setminus\{n\}^{(i)})$, selon la contrainte imposée par l'équation de parité du noeud m. Cette formule résulte de l'application de la règle de Bayes et sous l'hypothèse d'indépendance des messages entrant $\alpha^{(i-1)}_{n'\to m}$. La figure 1.2(a) illustre cette étape de calcul dans le graphe de Tanner.

2. *Mise à jour des messages allant des noeuds de donnée vers les noeuds de parité (data-pass en anglais)* :

   Pour tout les noeuds de donnée n et les noeuds de parité $m \in \mathcal{M}(n)$, le calcul de $\alpha_{n\to m}^{(i)}$ s'écrit :

$$\alpha_{n\to m}^{(i)} = L_{\mathrm{ch},n} + \sum_{m' \in \mathcal{M}(n)\setminus\{m\}} \beta_{m'\to n}^{(i)} \tag{1.7}$$

   L'équation (1.7) effectue ainsi la mise à jour du LLR conditionnel $\alpha_{n\to m}^{(i)}$ de n, en prenant en compte des LLRs conditionnels provenant des noeuds de parité voisins m' ($m' \in \mathcal{M}(n) \setminus \{m\}$) et le LLR canal $L_{\mathrm{ch},n}$ du bit n. Ce calcul peut s'apparenter à un vote majoritaire souple si le BP est comparé avec un algorithme à décision dure comme le Gallager B [35]. De plus, comme pour le check-pass (1.6), la formule (1.7) est obtenue en appliquant la règle de Bayes et en supposant que les messages rentrants $\beta_{m'\to n}^{(i)}$ sont indépendants. La figure 1.2(b) montre l'étape data-pass dans le graphe de Tanner.

3. *Calcul de l'information a posteriori des noeuds de donnée* :

   Pour tout les noeuds de donnée n, le calcul du LLR a posteriori est donné par :

$$\tilde{L}_n^{(i)} = L_{\mathrm{ch},n} + \sum_{m \in \mathcal{M}(n)} \beta_{m\to n}^{(i)} \tag{1.8}$$

   Cette équation nous indique que $\tilde{L}_n^{(i)}$ est aussi un LLR conditionnel, qui dépend des $\beta_{m\to n}^{(i)}$ ($m \in \mathcal{M}(n)$) et du LLR canal. De plus, les messages rentrants $\beta_{m\to n}^{(i)}$ sont supposés indépendants comme pour les 2 étapes précédentes. L'équation (1.8) est schématisée dans la figure 1.2(c).

4. *Décision dure* :

   La valeur de noeud de donnée n est estimée à partir du signe de $\tilde{L}_n^{(i)}$:

$$\tilde{c}_n^{(i)} = \frac{\left(1 - \mathrm{sign}\left(\tilde{L}_n^{(i)}\right)\right)}{2} \tag{1.9}$$

   Le syndrome du mot estimé à l'itération i, $\tilde{\mathbf{c}}^{(i)} = \{\tilde{c}_1^{(i)}, \ldots, \tilde{c}_N^{(i)}\}$, est alors calculé avec l'équation (1.1). S'il est vérifié (nul), $\tilde{\mathbf{c}}^{(i)}$ est alors un mot de code et le décodage est stoppé. Le vecteur $\tilde{\mathbf{L}}^{(i)} := \left\{\tilde{L}_1^{(i)}, \ldots, \tilde{L}_N^{(i)}\right\}$ constitue alors la sortie souple de l'algorithme BP, comme mis en évidence dans la figure 1.2(c). Nous notons alors par $\tilde{\mathbf{c}} = \tilde{\mathbf{c}}^{(i)}$ le mot de code estimé, en sortie du BP et par $\tilde{\mathbf{L}} = \tilde{\mathbf{L}}^{(i)}$ son vecteur de LLRs a posteriori associé. Si le syndrome n'est pas vérifié et si le nombre maximal d'itérations n'est pas atteint, l'algorithme BP entame une nouvelle itération de décodage avec le calcul des messages $\beta_{m\to n}^{(i+1)}$.

La complexité calculatoire du BP peut être mesurée, entre autres, par le nombre de mises à jour des messages passant par les noeuds de parité. Il s'agit en effet de l'étape la plus

Tableau. 1.1 – Nombre d'additions et de multiplications pour une itération du BP.

Étape du BP	Nombre d'additions	Nombre de multiplications
Check-pass	0	$\sum_{m=1}^{M} d_m(d_m - 2)$
Data-pass	$\sum_{n=1}^{N} d_n(d_n - 1)$	0
A posteriori	Nd_n	0

complexe à cause des fonctions tanh et $\tanh^{-1}$ [36]. La complexité du décodage BP dépend ainsi directement du nombre de noeuds de parité dans le graphe de Tanner et du degré de connexion de ces noeuds. Enfin, le tableau 1.1 montre la complexité des étapes check-pass, data-pass et a posteriori en mesurant leur nombre de multiplications et d'additions. Nous notons par d_m (resp. d_n) le degré de connexion d'un noeud de parité m (resp. d'un noeud de donnée n). Le nombre d'arêtes dans le graphe de Tanner et donc de messages calculés à l'étape check-pass (ou data-pass), est ainsi égal à $\sum_{m=1}^{M} d_m$. De plus, $\sum_{m=1}^{M} d_m = \sum_{n=1}^{N} d_n$. Précisons également que le nombre de termes est de d_n dans l'équation (1.7) et de $d_n + 1$ dans l'équation (1.8). Pour l'étape check-pass, uniquement les multiplications entre les termes $\tanh\left(\frac{\alpha_{n'\to m}^{(i-1)}}{2}\right)$ sont prises en compte, ce qui correspond à $d_m - 2$ termes dans l'équation (1.6). Il convient de mentionner que la mise à jour des messages allant des noeuds de donnée vers les noeuds de parité peut aussi s'écrire :

$$\alpha_{n\to m}^{(i)} = \tilde{L}_n^{(i)} - \beta_{m\to n}^{(i)} \tag{1.10}$$

Ainsi, l'étape data-pass peut être en pratique réalisée après le calcul de l'information a posteriori des noeuds de donnée, avec uniquement $\sum_{n=1}^{N} d_n$ additions au lieu de $\sum_{n=1}^{N} d_n(d_n - 1)$ comme indiqué dans le tableau 1.1.

Enfin, le BP est connu pour être capable d'obtenir d'excellentes performances en régime asymptotique ($N \longrightarrow +\infty$), atteignant la performance de la capacité de Shannon [7]. Néanmoins, en taille courte, ces performances sont dégradées à cause de l'apparition de structures problématiques dans le graphe de Tanner rendant invalide l'hypothèse d'indépendance des messages. Ces structures problématiques seront notamment détaillées dans les sous-sections 1.1.3 et 1.1.4. De plus, l'impact de ces structures sur la performance du BP sera montrée dans la sous-section 1.1.5.

1.1.2.2 Min-Sum

Bien que le BP fournisse une estimation efficace du maximum a posteriori de chaque bit, il possède deux désavantages principaux pouvant limiter son implémentation pratique. La première est l'utilisation des fonctions tanh et $\tanh^{-1}$ lors du check-pass. La seconde est la sensibilité du BP à l'estimation du paramètre du canal (σ^2 dans le cas du canal AWGN), utilisé lors de son initialisation pour calculer les LLRs canal. En pratique, ce paramètre canal doit être estimé. Une mauvaise estimation peut donc fausser l'initialisation et dégrader la performances du BP.

Pour répondre à ces deux problèmes, l'équation (1.6) de mise à jour des $\beta_{m\to n}^{(i)}$ est simplifiée par :

$$\beta_{m\to n}^{(i)} = \left(\prod_{n'\in\mathcal{N}(m)\setminus\{n\}} s_{n'\to m}^{(i-1)}\right)\left(\min_{n'\in\mathcal{N}(m)\setminus\{n\}} a_{n'\to m}^{(i-1)}\right) \tag{1.11}$$

où $s_{n'\to m}^{(i)} = \text{sign}\left(\alpha_{n'\to m}^{(i)}\right)$ et $a_{n'\to m}^{(i)} = \left|\alpha_{n'\to m}^{(i)}\right|$. Le décodeur utilisant cette approximation pour le check-pass est appelé le Min-Sum (MS). Les autres étapes du MS restent les mêmes que celles du BP. Remarquons qu'avec cette modification, la multiplication du vecteur d'information a priori $\mathbf{L}_{\text{ch}}$ par un facteur constant ne modifie pas la sortie du MS. Il se factorise en effet dans le calcul des messages $\beta_{m\to n}^{(i)}$, $\alpha_{n\to m}^{(i)}$ et de l'information a posteriori $\tilde{\mathbf{L}}$. Par conséquent, l'initialisation du MS peut en réalité s'écrire directement $L_{\text{ch},n} = y_n$ dans le cas du canal AWGN à entrée binaire. Pour cet exemple, le MS permet donc de bien de s'affranchir de l'estimation de la variance du bruit σ^2. Cependant, il est important de noter que le MS présente généralement des performances moins bonnes que le BP à cause de la simplification faite à l'équation (1.11). D'autres variantes de décodage par passage de messages des codes LDPC sont entre autres présentées en détail dans [35].

1.1.3 Construction des codes LDPC

La construction d'un code LDPC est une opération visant à établir les arêtes entre les noeuds de donnée et les noeuds de parité dans le graphe biparti selon le degré de connexion d_n de chaque noeud de donnée (ou selon le degré de connexion d_m de chaque le noeud de parité). Plusieurs approches existent dans la littérature pour construire un code LDPC, impactant de manière différente la performance de décodage BP. Ici, nous nous intéressons en particulier à l'algorithme par croissance d'arêtes progressives (Progressive Edge Growth ou PEG, en anglais) [37]. Il permet en effet d'améliorer la performance du BP avec des codes LDPC courts en minimisant le nombre de chemins fermés dans le graphe de Tanner.

1.1.3.1 Cycles

Un cycle est un chemin fermé dans un graphe. Pour un graphe biparti, un cycle est nécessairement de longueur paire, supérieure ou égale à 4. Par exemple, sur la figure 1.3, le graphe biparti admet un cycle de longueur 4 impliquant les nœuds de parité m_2 et m_3 et les nœuds de donnée n_3 et n_4.

Notons que les cycles présents dans le graphe biparti ont un impact direct sur la performance des algorithmes de décodage par passage de messages. En particulier, l'existence de cycles courts dans un graphe biparti induit de mauvaises performances pour ces algorithmes. En effet, les cycles courts empêchent la validité de l'hypothèse d'indépendance des messages se propageant sur le graphe. Or, moins cette hypothèse est valide, plus le BP dévie de l'estimation maximum a posteiori des bits codés [8]. Avantageusement, la faible densité de la

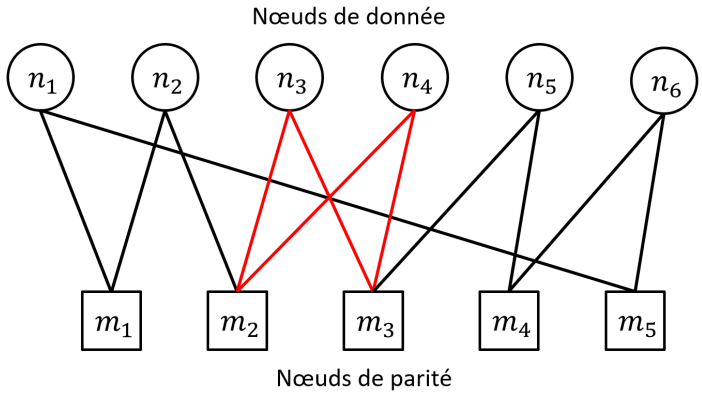


Figure 1.3 – Exemple d'un cycle de taille 4, en rouge.

matrice de parité d'un code LDPC se traduit par un graphe biparti creux, ayant donc peu de cycles courts et ce, particulièrement pour des codes longs. Cela fait ainsi des algorithmes par passage de messages la solution priviligiée pour le décodage des codes LDPC de grande taille.

Cependant, dans cette étude, nous nous intéressons aux codes LDPC courts. Une conséquence directe est que la matrice de parité d'un tel code ne peut pas être suffisamment creuse et que le graphe biparti associé contient inévitablement des cycles courts. Par conséquent, le BP se retrouve avec des performances dégradées. Il apparaît donc important de construire des codes LDPC courts possédant le moins de cycles courts possible.

1.1.3.2 Algorithme PEG

L'algorithme PEG [37] est une méthode de construction des codes LDPC, effectuant le meilleur effort pour éviter d'introduire des cycles courts. Plus précisément, l'idée du PEG est de placer les arêtes du graphe de manière récursive, en connectant à chaque fois un nœud de parité m à un nœud de donnée n, choisi tel que la distance de m à n soit maximale. En d'autres termes, pour N noeud de donnée, M noeuds de parité et un profil de degré de connexion des noeuds de donnée $D_n := [d_1, \ldots, d_N]$, avec d_n le degré de connexion du noeud de donnée n, l'algorithme PEG établit des connexions dans le graphe de telle sorte que l'ajout d'une nouvelle arête ait le moins d'impact possible sur la circonférence g. Cette circonférence représente la longueur minimale des cycles. Cela ne garantit pas l'absence de cycles courts (en particulier des cycles de longueur 4), mais peut réduire significativement leur nombre. Il apparaît donc avantageux d'utiliser le PEG pour générer des codes LDPC courts. Son détail est donné ci-dessous.

Afin d'établir une connexion entre un noeud de donnée n et un noeud de parité, le sous-graphe est étendu à partir du noeud de donnée n. La figure 1.4 illustre ceci, avec ℓ dénotant la profondeur du sous-graphe étendu. Nous notons par $\mathcal{M}(n)^{(\ell)}$ l'ensemble des noeuds de parité atteint à la profondeur ℓ en partant du bit n. Deux situations peuvent alors se produire. Dans le premier cas, les noeuds de parité ne sont pas tous accessibles en partant du noeud

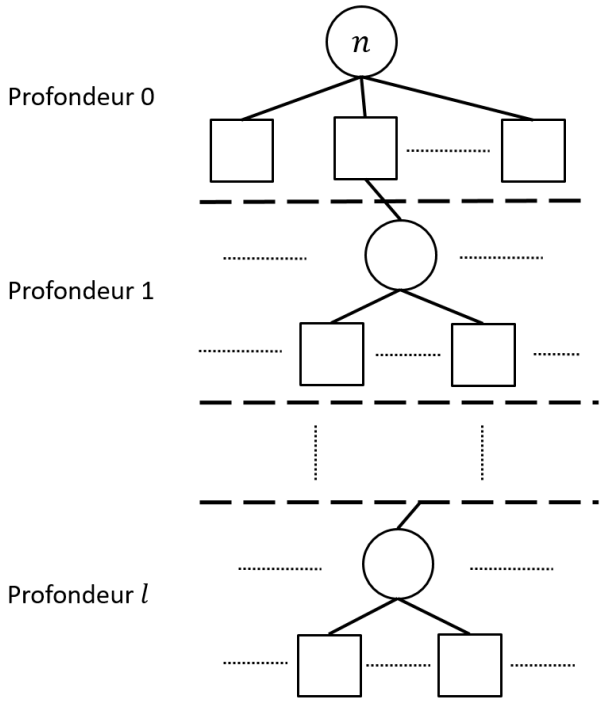


Figure 1.4 – Exemple de sous-graphe étendu.

de donnée n. Le cardinal de $\mathcal{M}(n)^{(\ell)}$ reste ainsi inférieur à M. L'algorithme PEG choisit alors de connecter n avec l'un des noeuds de parité non atteint, ne créant ainsi aucun cycle. Dans le second cas, les noeuds de parité sont tous atteint à une profondeur donnée. Si nous supposons que cette profondeur est égale à $\ell + 1$, le cardinal de $\mathcal{M}(n)^{(\ell)}$ est par conséquent inférieur à M et celui de $\mathcal{M}(n)^{(l+1)}$ est égal à M. Le PEG choisit alors de créer une connexion entre le noeud de donnée n et l'un des noeuds de parité les plus lointains de ce dernier, donc à la distance $\ell + 1$. Le cycle établit possède de ce fait la taille maximale possible $2(\ell + 2)$. L'algorithme 1 donne le pseudo-code résumant le processus PEG.

Nous pouvons également noter que l'algorithme PEG diffère des techniques précédentes de construction [38, 39], où les graphes des codes LDPC sont générés de manière aléatoire tout en évitant les cycles de taille 4. Ces codes LDPC construits aléatoirement comptent surtout sur la faible densité de la matrice H pour éviter la présence de cycles courts. L'algorithme PEG apparaît donc comme une meilleure alternative pour générer des codes LDPC courts.

1.1.4 Ensembles absorbants

Un second inconvénient majeur des codes LDPC de taille finie est la présence de structures particulières dans le graphe biparti, en plus des cycles, empêchant l'algorithme de décodage de converger vers un mot de code. Les ensembles piégeants [40] et les ensembles

Algorithme 1 PEG

Entrée : N, M, D_n
pour $n = 1$ à N **faire**
  pour $d = 0$ à $d_n - 1$ **faire**
    si $d = 0$ **alors**
      ▷ Aucune connexion de n avec un noeud de parité
      Choisir l'un des noeuds de parité avec le plus faible degré de connexion dans l'état actuel de la construction.
    sinon
      Étendre récursivement le graphe depuis n jusqu'à ce que l'une des deux conditions suivantes soient satisfaites
      si $\left|\mathcal{M}(n)^{(\ell)}\right| < M$ et $\left|\mathcal{M}(n)^{(\ell+1)}\right| = M$ **alors**
        Choisir l'un des noeuds de parité atteint à la profondeur $\ell + 1$, puis le relier à n
      sinon si $\left|\mathcal{M}(n)^{(\ell)}\right| < M$ et $\left|\mathcal{M}(n)^{(\ell+1)}\right|$ cesse d'augmenter **alors**
        Choisir l'un des noeuds de parité non atteint, puis le relier à n
      fin si
    fin si
  fin pour
fin pour
Retourner le graphe de Tanner

absorbants [33] sont des exemples typiques de ces structures. Ils sont d'ailleurs étroitement liés aux notions de pseudo mots de code [41] et de quasi mots de code [42], deux concepts caractérisant des vecteurs $\mathbf{c}$ particuliers empêchant le BP de converger vers le mot de code attendu. Notons également qu'un vecteur $\mathbf{c}$ correspondant à un quasi mot de code ou un pseudo mot de code ne vérifie pas le syndrome et n'est donc pas un mot de code.

Dans ce manuscrit, nous nous concentrons en particulier sur la notion d'ensembles absorbants. Ce sont des structures combinatoires du graphe, indépendantes du décodeur par passage de messages considéré et du modèle de bruit. De ce fait, les méthodes pour apprendre à traiter ces structures peuvent se généraliser aux BP et ses variantes.

Pour introduire plus formellement un ensemble absorbant, les définitions introduites dans [33] et [43] sont utilisées. Soit $\mathcal{A} \subset \{1, \ldots, N\}$ un ensemble de noeuds de donnée. Nous considérons l'ensemble des noeuds de parité connectés au moins une fois à $\mathcal{A}$. Cet ensemble est ensuite partitionné en deux sous ensembles disjoints, $\mathcal{O}(\mathcal{A})$ et $\mathcal{E}(\mathcal{A})$, représentant respectivement les ensembles de noeuds de parité connectés à $\mathcal{A}$ un nombre impair ou pair de fois. L'ensemble $\mathcal{A}$ est alors un ensemble absorbant si chaque noeud de donnée de $\mathcal{A}$ a strictement plus de voisins dans $\mathcal{E}(\mathcal{A})$ que dans $\mathcal{O}(\mathcal{A})$. Pour illustrer cette définition, un exemple de quatre noeuds de donnée formant un ensemble absorbant est donné dans la figure 1.5.

Selon cette définition, si les bits en erreur dans le signal reçu y forment un ensemble

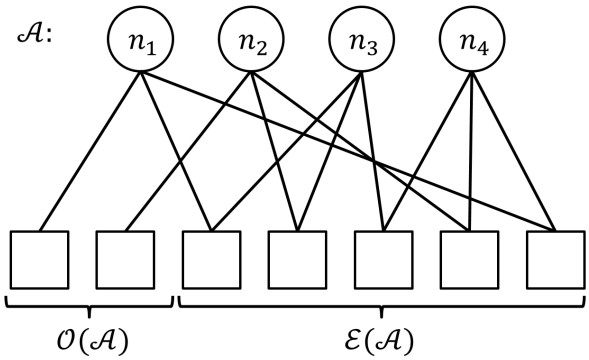


Figure 1.5 – Exemple d'un ensemble absorbant de 4 noeuds de donnée.

absorbant $\mathcal{A}$, alors chacun d'eux est connecté à un nombre supérieur de noeuds de parité satisfaits, ne détectant donc pas l'erreur, que de noeuds de parité non satisfaits et détectant l'erreur. De telles erreurs sont susceptibles de leurrer le décodeur BP et d'entraîner ainsi un échec de décodage avec une grande probabilité. Pour des codes LDPC longs, les ensembles absorbants sont connus pour être responsable du phénomène de "plancher d'erreur" dans les performances de décodage. En effet les tailles des ensembles absorbants sont relativement petites par rapport à la taille du code [33]. Cependant, pour les codes courts, leurs tailles peuvent être comparables au nombre d'erreurs que le code doit corriger, ce qui peut induire une dégradation significative de la capacité de correction des erreurs dans la région du *waterfall*.

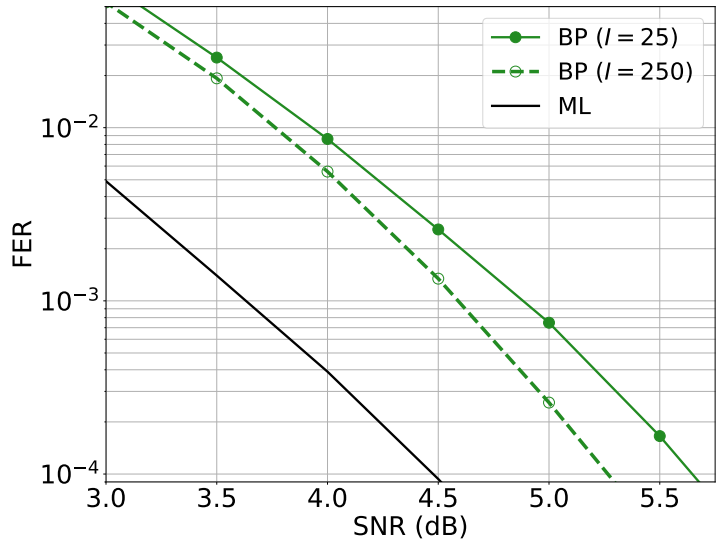
1.1.5 Performances des codes LDPC courts décodés par BP

Deux codes LDPC courts sont considérés pour évaluer leurs performances respectives avec un décodage par BP. Ils seront utilisés dans la suite du manuscrit. Leurs caractéristiques sont données dans le tableau 1.2 ci-dessous, avec en particulier $\mathbf{n}_{\text{g-cycles}}$ la multiplicité de g définie comme le nombre de cycles de taille g.

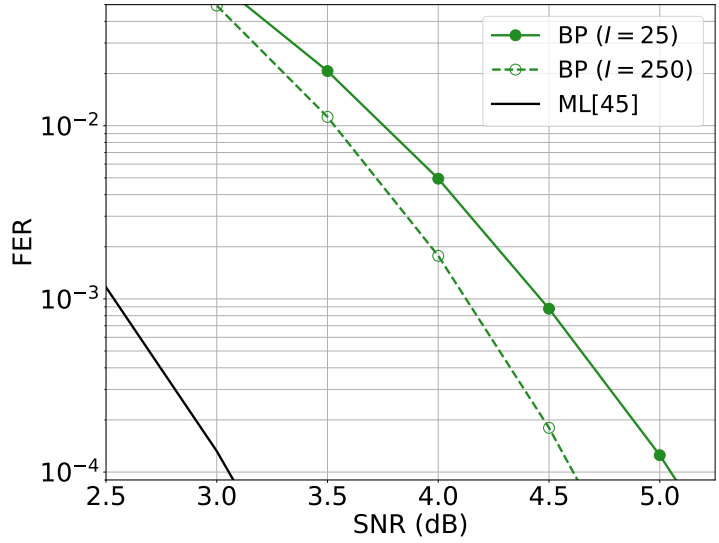
Précisons que le Code-1 a été construit avec l'algorithme PEG, présenté dans le paragraphe 1.1.3.2. De plus, ses noeuds de donnée et de parité sont de degrés réguliers ($d_n = 3\,\forall n \in [1, N]$ et $d_m = 6\,\forall m \in [1, M]$). Le Code-2 est quant à lui un code LDPC standardisé

Tableau. 1.2 – Paramètres des codes LDPC évalués

	N	K	R_c	d_n	d_m	g	$\mathbf{n}_{\text{g-cycles}}$
Code-1	64	32	0.5	3	6	6	164
Code-2 [44]	128	64	0.5	3-5	8	6	2336



(a) FER du Code-1.



(b) FER du Code-2 [44].

Figure 1.6 – FER de codes LDPC courts décodés par BP, avec $I = 25, 250$ itérations.

provenant de [44] et est connu comme le code LDPC CCSDS. Ses noeuds de donnée ont un degré irrégulier, avec une première moitié de degré $d_n = 3$ et une seconde moitié de degré $d_n = 5$.

La figure 1.6 présente pour les deux codes les performances du décodage BP pour un nombre maximal de 25 et 250 itérations, ainsi que la performance du décodage par maximum

de vraisemblance (Maximum Likelihood ou ML en anglais). La performance est évaluée en termes de taux d'erreur paquet (Frame Error Rate ou FER, en anglais) et les gains sont relevés pour un FER de 10^{-4}. La performance ML du Code-2 est fournie par [45], tandis que pour le Code-1, la performance ML a été approximée selon l'approche introduite dans [28]. Il s'agit en réalité d'un décodeur par statistiques ordonnées (Ordered Statistics Decoding ou OSD, en anglais) d'ordre 3, décodeur que nous détaillerons dans la section 4.1. Sur la figure 1.6a, nous observons que le BP n'atteint effectivement pas la performance ML pour le Code-1, avec un écart de 1 dB (resp. 0.79 dB) pour $I = 25$ (resp. $I = 250$). Les écarts sont accentués pour le Code-2, où le BP($I = 25$) se situe à 2 dB de la performance ML et le BP($I = 250$) à 1.61 dB. Le Code-2 possèdent en effet plus de cycle de taille 6 que le Code-1 (2336 contre 164 respectivement). Ces dégradations découlent aussi de la présence de motifs d'erreurs de petite taille formant des ensembles absorbants, empêchant l'algorithme BP de converger correctement vers un mot de code. Nous illustrons ceci pour le Code-1. Nous avons tout d'abord énuméré les ensembles absorbants de taille inférieure ou égale à 8 [1]. 3521016 ensembles absorbants ont ainsi été déterminés. De plus, nous avons calculé la probabilité d'obtenir un motif d'erreurs aléatoire de taille inférieure ou égale à 8. Pour un code de taille N et une variance de bruit gaussien σ^2 fixée, la probabilité d'avoir un motif de E erreurs est donnée par :

$$P_{\text{err}}(E) = \binom{N}{E} p^E (1-p)^{N-E} \tag{1.12}$$

avec p la probabilité d'erreur du canal. Pour le canal AWGN, p se calcule avec l'équation suivante :

$$p = Q\left(\frac{1}{\sigma}\right) = \frac{1}{\sqrt{2\pi}} \int_{\frac{1}{\sigma}}^{+\infty} e^{\frac{-t^2}{2}} dt. \tag{1.13}$$

La probabilité d'obtenir un motif d'erreurs de taille inférieure ou égale à 8 est alors égale à $\sum_{E=0}^{8} P_{\text{err}}(E)$. Elle vaut en particulier 0.99 pour un SNR = 4 dB. En conséquence, certains motifs d'erreurs de taille inférieure ou égale à 8 correspondront bien à des ensembles absorbants dans la région du *waterfall* et résulteront en un échec de décodage BP avec une forte probabilité. Des conclusions similaires peuvent également être établies pour le Code-2. Nous constatons ainsi qu'il est important de traiter les cycles et les ensembles absorbants lors de la construction et/ou du décodage afin de réduire les dégradations induites et de se rapprocher de la performance ML.

1.2 Réseaux de neurones

Les réseaux de neurones (Neural Network ou NN, en anglais) sont de plus en plus mis en avant pour être intégrés dans la prochaine génération de systèmes de communication au

1. La méthode d'énumération sera détaillée dans le chapitre 3.

niveau de la couche physique [46, 47]. Ils peuvent notamment être utilisés soit pour améliorer, soit pour remplacer un algorithme de décodage déjà existant [48]. Pour le décodeur BP et ses variantes, le but du réseaux de neurones est d'améliorer la performance de décodage en limitant l'impact des cycles et des structures problématiques présentées dans la section précédente. En particulier, les auteurs de [10] montrent que le BP se prête naturellement à la modélisation en un réseau de neurones, appelé BP neuronal et proposent ainsi une première amélioration de la performance de décodage. Avant de présenter cette approche, nous proposons de revenir dans cette section sur les notions relatives aux NN, ainsi que sur les architectures neuronales considérées. Nous précisons aussi que cette section se base en partie sur [49], un livre écrit par l'un des auteurs de la librairie d'apprentissage profond Keras dans le langage de programmation python.

1.2.1 *Définition d'un neurone*

Un neurone est la brique opératoire de base d'un réseaux de neurones. Il possède généralement plusieurs entrées (notées $e_1, \ldots, e_T$, avec T le nombre d'entrées du neurone), provenant soit de l'entrée du réseau, soit des étages précédents connectés à ce neurone. Le neurone applique alors un poids w_t sur chacune des entrées, puis il réalise la somme pondérée, avec l'ajout éventuel d'un biais b. Les poids w_t, $t \in [1, T]$ et b sont les paramètres dits entraînables du neurone, car ils correspondent aux paramètres qui seront optimisés lors de l'entraînement du réseau. Pour finir, la sortie $\tilde{s}$ est obtenue après passage par une fonction d'activation f_a. La figure 1.7 représente le neurone décrit ci-dessus et l'équation (1.14) formalise le calcul de la sortie d'un neurone :

$$\tilde{s} = f_a\left(\sum_{t=1}^{T} w_t e_t + b\right) \tag{1.14}$$

La fonction d'activation peut avoir plusieurs objectifs : réduire les problèmes d'instabilités numériques (notamment lors de l'entraînement, où un calcul de gradient est réalisé), faciliter

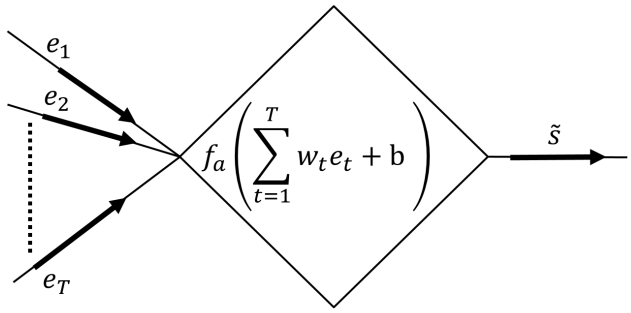


Figure 1.7 – Représentation d'un neurone standard.

l'apprentissage, ou encore modéliser les sorties comme l'utilisateur le souhaite. Par exemple, si le neurone doit réaliser une classification binaire, une fonction sigmoïde $\sigma(x) = (1+e^{-x})^{-1}$ est généralement utilisée afin de fournir la probabilité d'appartenir à la classe 0 ou 1. Parmi les fonctions d'activation les plus courantes, on peut aussi citer ReLU (Rectified Linear Unit), tangente hyperbolique, softmax etc. ; chacune étant ainsi choisie selon la tâche que le neurone doit accomplir. La fonction d'activation ajoute de la non-linéarité dans le NN. L'espace des fonctions approchables par ce dernier est alors plus vaste et il peut mieux apprendre à résoudre le problème posé (classification, régression etc.).

1.2.2 *Réseau de neurones à propagation avant*

Un réseau de neurones est composé de plusieurs couches successives de neurones, appelées étages neuronaux. Les neurones d'un même étage partagent généralement la même fonction d'activation, chaque étage étant ainsi associé à une fonction particulière. Cette combinaison de neurones en couche permet aussi de calculer en parallèle les caractéristiques associées à chacun d'eux. De manière plus formelle, un étage de R neurones, où chacun d'eux prend T entrées, se représente avec une fonction d'activation f_a, une matrice de poids $\mathbf{W} := \{w_{r,t}\}_{r=1,t=1}^{R,T} \in \mathbb{R}^{R,T}$ et un vecteur de biais $\mathbf{b} := \{b_r\}_{r=1}^{R} \in \mathbb{R}^R$. Il effectue l'opération matricielle suivante :

$$\begin{pmatrix} \tilde{s}_1 \\ \vdots \\ \tilde{s}_R \end{pmatrix} = f_a \left[\begin{pmatrix} w_{1,1} & \dots & w_{1,T} \\ \vdots & & \vdots \\ w_{1,R} & \dots & w_{R,T} \end{pmatrix} \begin{pmatrix} e_1 \\ \vdots \\ e_T \end{pmatrix} + \begin{pmatrix} b_1 \\ \vdots \\ b_T \end{pmatrix} \right] \tag{1.15}$$

Chaque ligne de la matrice de poids $\mathbf{W}$ et du vecteur de biais $\mathbf{b}$ correspond aux poids et au biais d'un neurone de l'étage. Appliquer la fonction d'activation f_a sur un vecteur, comme dans (1.14), revient à appliquer f_a sur chacune de ses composantes. L'étage neuronal donne en sortie un vecteur $\mathbf{s} := [s_1, \dots, s_R]$ de taille R, avec s_r la sortie du neurone r. Ce vecteur de sorties sera soit utilisé en tant qu'entrée dans l'étage suivant soit en tant que sortie du NN.

Un réseau de neurones, composé de quatre étages, est illustré sur la figure 1.8. Les étages (aussi appelés couches) se distinguent en réalité en trois catégories. L'étage d'entrée est le premier étage du réseau, il prend en argument les entrées du réseau de neurones et effectue un premier traitement dessus. Le dernier étage fournit les sorties du NN, sous la forme attendue pour répondre au problème (vecteur de probabilité pour de la classification, un signal du même type que l'entrée pour de la régression). Les étages entre l'étage d'entrée et celui de sortie sont appelés étages cachés et ils extraient/traitent les caractéristiques des données manipulées par le réseau. Il convient de préciser que chaque étage peut posséder un nombre différent de neurones et que le nombre de neurones dans l'étage de sortie est égale à la dimension de la sortie attendue. Il n'existe cependant pas de règles universelles pour choisir un nombre optimal de neurones dans les étages cachés et un nombre optimal d'étages cachés. Ces choix s'effectuent généralement de manière assez empirique, en testant plusieurs architectures

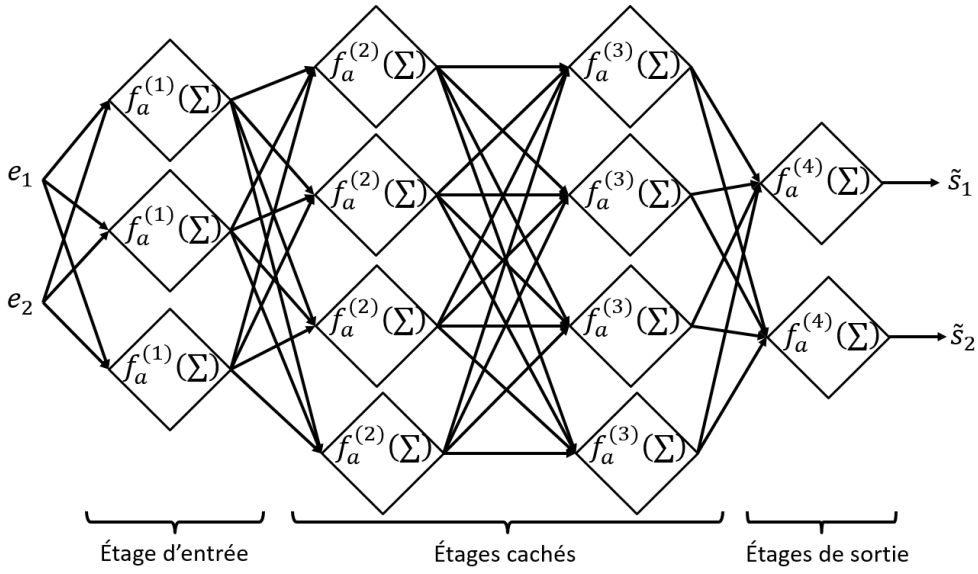


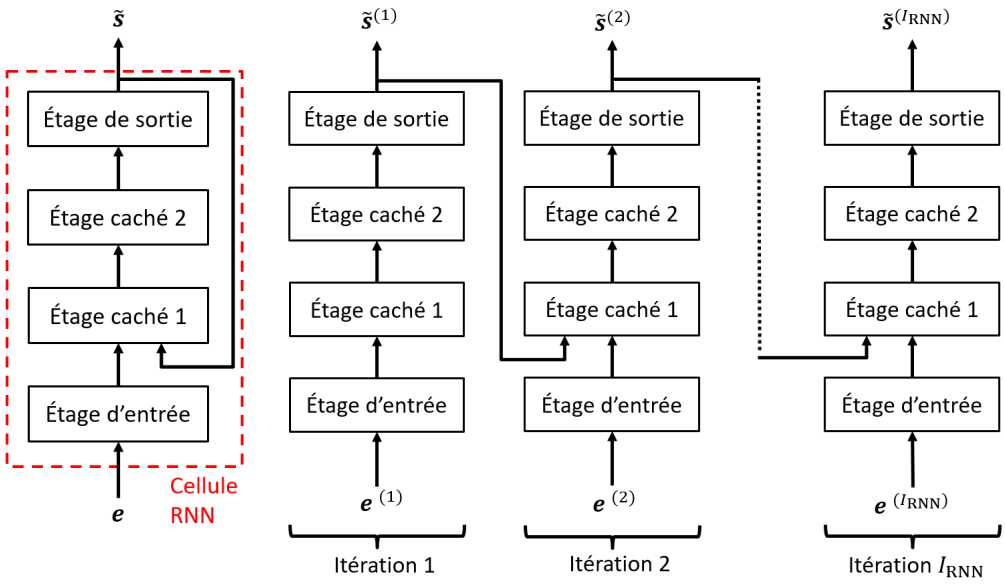
Figure 1.8 – Exemple d'un réseau de neurones FF.

neuronales puis en sélectionnant la meilleure en termes de performance/complexité. De plus, la figure 1.8 montre qu'un réseau peut posséder des connexions particulières, les neurones n'étant pas tous connectés entre eux entre l'étage d'entrée et l'étage caché.

Le NN présenté en figure 1.8 est qualifié de réseau à propagation avant (NN Feed Forward ou NN-FF, en anglais), les connexions entre les neurones ne formant pas de boucles de rétro-actions. L'information circule donc dans un sens unique, de l'étage d'entrée vers la sortie. De plus, chaque étage possède une matrice de poids et une matrice de biais indépendantes et différentes des autres étages.

1.2.3 Réseau de neurones récurrent

Un réseau de neurones récurrent (Recurrent Neural Network ou RNN, en anglais) [50] repose sur les mêmes concepts qu'un réseau à propagation avant, excepté la manière dont les neurones sont connectés entre eux. En effet, les neurones, les fonctions d'activation et les étages sont définis de la même manière. Cependant, des boucles de rétro-actions sont présentes entre les étages neuronaux. Plus précisément, un RNN s'exécute sur plusieurs itérations de calcul et des étages neuronaux sont répétés entre ces itérations (le nombre d'itérations I_{RNN} est choisi par l'utilisateur). Les étages répétés forment une cellule RNN et les boucles sont employées pour transmettre des états de mémoire entre les étages de la cellule au fur et à mesure des itérations. De plus, les mêmes jeux de poids et de biais des étages répétés sont appliqués sur chaque itération. De cette façon, le RNN permet de simuler aisément un processus itératif et de traiter des données séquentielles évoluant avec le temps. Il est donc



(a) RNN non déroulé. **(b)** RNN déroulé.

Figure 1.9 – Exemple d'un RNN, sous sa représentation déroulée et non déroulée.

particulièrement adapté pour de l'analyse sémantique de texte ou de la reconnaissance vocale. Remarquons aussi que le RNN se prête bien à la modélisation du BP puisque ce dernier est un algorithme de décodage itératif.

La figure 1.9 illustre une architecture de type RNN, déroulée et non déroulée, avec une boucle de rétroaction entre l'étage de sortie et le premier étage caché. La représentation déroulée du cas (b) permet de bien montrer le processus itératif d'un réseau RNN, avec un nombre maximal de I_{RNN} itérations, tandis que la représentation non déroulée du cas (a) met en évidence la cellule RNN de cet exemple. La sortie du réseau à l'itération $\tilde{\mathbf{s}}^{(i)}$ est utilisée pour calculer la sortie du premier étage caché à l'itération $i+1$. La cellule RNN est composé de tous les étages neuronaux, qui sont répétés à chaque itération avec les mêmes jeux de poids et de biais. En outre, une matrice de poids U peut aussi être spécialement appliquée aux états mémoires. La sortie de l'étage caché 1 à l'itération i s'écrit alors :

$$\tilde{\mathbf{s}}_{c_1} = f_{a_{c_1}}\left[\mathbf{W}\mathbf{e}_{c_1} + \mathbf{U}_{c_1}\mathbf{s}^{(\mathrm{i}-1)} + \mathbf{b}_{c_1}\right] \tag{1.16}$$

où le sous-indice c_1 indique que les jeux de poids et biais sont propres à l'étage caché 1, $\mathbf{e}_{c_1}$ est l'entrée de l'étage caché 1 (donc la sortie de l'étage d'entrée) et $\mathbf{s}_{c_1}$ sa sortie.

1.2.4 Entraînement d'un réseau de neurones

1.2.4.1 Principe général

Une fois le réseau construit, les paramètres entraînables doivent être optimisés avec un algorithme d'entraînement, afin que le réseau apprenne à réaliser la tâche souhaitée. Dans un premier temps, il est néanmoins nécessaire de constituer un jeu de donnée d'entraînement $\mathcal{T}_{\text{ent}}$, sur lequel le réseau devra apprendre les caractéristiques des données en entrée du réseau. La création du jeu d'entraînement est une étape très importante, influant énormément sur les performances du NN. Par exemple, si un réseau doit apprendre à séparer des images de chien et de chat, il faut une répartition égale entres les deux animaux. En effet, si le jeu est uniquement constitué d'images de chat, le réseau ne pourra intuitivement pas apprendre à classifier les images de chien. En plus du jeu d'entraînement, un jeu de donnée similaires noté $\mathcal{T}_{\text{val}}$, dit de validation, est crée dans le seul but de contrôler les performances du NN au cours de l'entraînement. Le réseau ne se sert donc pas de $\mathcal{T}_{\text{val}}$ pour apprendre des propriétés.

La prochaine étape consiste à définir une fonction de coût, notée f_L, indiquant au réseau ses performances sur les jeux d'entraînement et de validation. Le choix de cette dernière dépend du problème à traiter. Par exemple, une entropie croisée (cross entropy en anglais) est un type de fonction très utilisée pour la classification, tandis que l'erreur quadratique moyenne sert généralement pour la régression [51]. L'entropie croisée sera notamment formalisée dans le paragraphe suivant dans le cadre d'un problème de classification binaire. Ensuite, l'objectif est de minimiser f_L sur le jeu d'entraînement, en mettant à jour les paramètres entraînables du NN. L'algorithme réalisant ceci est appelé l'entraînement du réseau de neurones.

Le coeur de l'entraînement est un processus séquentiel de mise à jour des paramètres entraînables, reposant sur une méthode de type descente de gradient (Optimizer en anglais). Ce processus séquentiel est itéré plusieurs fois jusqu'à convergence vers la solution optimale. Chaque itération de ce processus est appelée une époque. De plus, le jeu d'entraînement $\mathcal{T}_{\text{ent}}$ est généralement découpé en P sous-ensembles de taille quasi-égale, appelés paquets (batch en anglais) [2]. Le processus séquentiel effectué au cours d'une époque consiste alors à mettre à jour les paramètres entraînables P fois de manière séquentielle, avec un traitement paquet par paquet. Les paramètres entraînables obtenus au bout d'une époque correspondent donc à un résultat d'optimisation une fois que tout le jeu d'entraînement a servi à minimiser la fonction de coût. La figure 1.10 affiche le schéma de principe d'une époque d'entraînement (et donc du processus séquentiel) et montre également les étapes menant à une mise à jour des paramètres entraînables lors du traitement de chaque paquet $\mathcal{T}_{\text{ent},p}$, $p \in [1, P]$. Nous détaillons les étapes ci-dessous :

(1) Le réseau calcule les sorties correspondantes au paquet numéro p, avec les poids et les biais mis à jour suite au traitement du paquet précèdent $\mathcal{T}_{\text{ent},p-1}$ [3].

2. Si $|\mathcal{T}_{\text{ent}}|$ n'est pas divisible par P, certains paquets seront plus grands que les autres pour répartir tous les éléments du jeu d'entraînement.

3. Si $p = 1$, les paramètres utilisés sont soit les paramètres initiaux (époque 0), soit ceux obtenus à la fin de l'époque précédente.

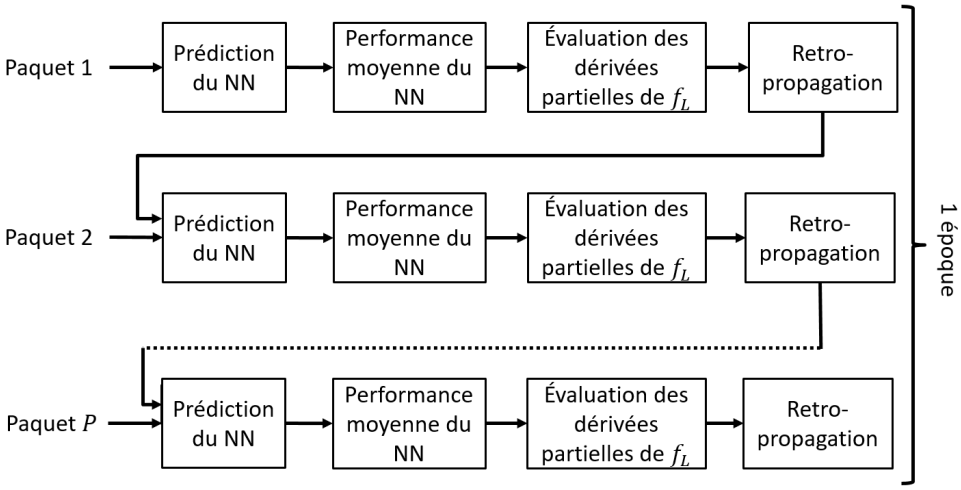


Figure 1.10 – Schéma de principe d'une époque d'entraînement.

(2) La performance moyenne du NN est évaluée en moyennant la fonction de coût appliquée sur chaque sortie prédite.

(3) Les dérivées partielles de f_L selon chaque paramètre entraînable sont calculées grâce au principe de dérivation en chaîne, puis évaluées en la performance moyenne du NN sur le paquet $\mathcal{T}_{\text{ent},p}$.

(4) Une méthode de type descente de gradient est mise en oeuvre pour mettre à jour les paramètres entraînables à travers tout le réseau. Cette dernière étape est appelée retro-propagation du gradient. Les poids et biais mis à jour sont alors utilisés pour le traitement du paquet suivant.

Précisons enfin que la mise à jour des paramètres entraînables pour un paquet est mise en équation dans le paragraphe suivant.

Le nombre d'époques est à choisir afin d'éviter le sous-apprentissage (trop peu d'époques) et le sur-apprentissage (trop d'époques). Dans le premier cas, le réseau n'est pas assez entraîné, tandis que pour le sur-apprentissage, le réseau est trop spécifique aux données du jeu d'entraînement. Ces problèmes sont observés durant l'entraînement en comparant l'évolution au fur et à mesure des époques de deux grandeurs, la perte d'entraînement et la perte de validation. Le coût d'entraînement est mesurée en moyennant en fin d'époque la performance moyenne du NN de chaque paquet calculée séquentiellement. Le coût de validation correspond quant à lui à la performance moyenne du NN sur le jeu de validation $\mathcal{T}_{\text{val}}$ avec les paramètres mis à jour à la fin d'une époque. La figure 1.11 illustre les situations de sous-apprentissage et de sur-apprentissage, en fonction du comportement des coûts de validation et d'entraînement. Le phénomène de sur-apprentissage apparaît quand le coût de validation augmente de nouveau tandis que le coût d'entraînement continue de décroître. Dans cet exemple, il

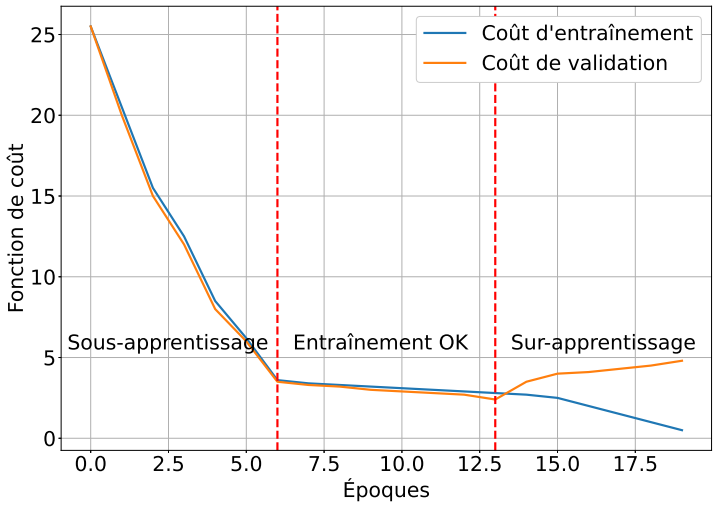


Figure 1.11 – Coût d'entraînement et de validation en fonction des époques

est ainsi nécessaire de limiter l'entraînement à 13 époques. De plus, la division de $\mathcal{T}_{\text{ent}}$ en plusieurs paquets est surtout mise en place afin d'accélérer le processus d'entraînement grâce à une parallélisation des calculs. La taille des paquets est donc un compromis entre vitesse d'entraînement et une optimisation correcte (nombre suffisant de retro-propagations), jouant aussi sur le nombre d'époques nécessaires.

De plus, il est important de préciser que toutes les opérations appliquées dans le réseau de neurones doivent être différentiables (ou substituables). Dans le cas contraire, les gradients ne pourront pas être calculés et le réseau ne pourra pas être entraîné. Il faut aussi être prudent sur le nombre d'étages cachés et le choix des fonctions d'activation afin d'éviter des problèmes d'évanouissement du gradient. En effet, lors de la dérivation en chaîne, certaines fonctions d'activation peuvent donner des niveaux de gradients très faibles (comme tanh) et empêcher ainsi les paramètres des premiers étages cachés de se mettre à jour correctement. Cet évanouissement de gradient se renforce donc d'autant plus que le réseau est profond. Le phénomène opposé, l'explosion du gradient, est aussi à surveiller dans le cas où des fonctions d'activations produiraient des gradients trop grands.

Il convient aussi de préciser que même si l'architecture est différente entre un RNN et un NN-FF, la manière d'entraîner reste similaire. Cependant, il est possible d'avoir accès aux sorties à chaque itération d'une cellule RNN. Certaines fonctions de coût sont alors conçues pour être calculées sur la sortie à chaque itération. Il suffit ensuite de sommer ces résultats pour obtenir la fonction de coût globale du paquet en cours. De telles fonctions de coût sont appelées fonctions de coût multiples. En outre, comme le jeu de poids et de biais d'une cellule RNN est le même pour toutes les itérations du RNN, il n'y aura que ce jeu à modifier par descente de gradient durant l'entraînement.

Enfin, l'entraînement est communément suivi d'une phase de test. Son but est de confir-

mer que le réseau a été entraîné correctement, en évaluant la performance du NN une fois l'optimisation des paramètres entraînables terminée. Le test s'effectue sur un jeu de donnée dit de test, noté $\mathcal{T}_{\text{test}}$. Il contient des données du même type que celles du jeu d'entraînement et de validation, mais il est construit de manière indépendante.

1.2.4.2 Formalisation de l'entraînement pour de la classification binaire supervisée

Dans ce manuscrit, nous nous concentrons en particulier sur des problèmes d'apprentissage supervisés pour de la classification binaire. Nous détaillons ainsi de manière plus formelle le processus d'entraînement. Pour ce type de problèmes, chaque entrée e de $\mathcal{T}_{\text{ent}}$ et $\mathcal{T}_{\text{val}}$ est associée à un vecteur $\mathbf{s} = [s_1, \ldots, s_R]$, où s_r indique la classe attendue de la r-ième sortie du NN lorsque l'entrée est e. Par conséquent, $s_r = 0$ (resp. $s_r = 1$) si la r-ième sortie du NN est censée appartenir à la classe 0 (resp. 1). Dans le cas du décodage, $\mathcal{T}_{\text{ent}}$ est constitué de mots bruités $\mathbf{y}$ et les vecteurs attendus sont les mots de code émis $\mathbf{c}$ correspondants. Le réseau doit alors apprendre une cartographie des entrées du jeu d'entraînement vers les sorties attendues. La qualité de la prédiction est évaluée avec la fonction de coût, en comparant les sorties calculées $\tilde{\mathbf{s}}$ avec les sorties attendues $\mathbf{s}$. La fonction de coût classiquement utilisé pour de la classification binaire supervisée est l'entropie croisée binaire (Binary Cross Entropy ou BCE, en anglais) [51], définie ci-dessous par l'équation (1.17) :

$$f_{\text{BCE}}(\tilde{\mathbf{s}}, \mathbf{s}) = -\frac{1}{R}\sum_{r=1}^{R}(1 - s_r)\log(1 - \tilde{s}_r) + s_r \log(\tilde{s}_r) \tag{1.17}$$

La fonction BCE est conçue pour mesurer la dissimilarité entre la sortie attendue et la sortie calculée par le NN. En d'autres termes, elle permet d'évaluer la distance entre la sortie calculée et la classe effectivement souhaitée. Une autre fonction de coût plus récemment introduite est la fonction de coût focal [52] (Focal Loss ou FL, en anglais) :

$$f_{\text{FL}}(\tilde{\mathbf{s}}, \mathbf{s}) = -\frac{1}{R}\sum_{r=1}^{R}(1 - s_r)\,(\tilde{s}_r)^{\gamma}\log(1 - \tilde{s}_r) + s_r(1 - \tilde{s}_r)^{\gamma}\log(\tilde{s}_r) \tag{1.18}$$

avec $\gamma \geq 0$ un hyper-paramètre ajustable appelé exposant focal. Son principe est le même que celui de la fonction BCE. D'ailleurs, pour $\gamma = 0$, $f_{\text{FL}} = f_{\text{BCE}}$. La fonction de coût focale permet cependant de focaliser l'apprentissage sur les éléments les plus durs à classifier, en leur affectant de plus grandes pénalités qu'aux autres éléments grâce aux facteurs focaux $(\tilde{s}_r)^{\gamma}$ et $(1 - \tilde{s}_r)^{\gamma}$. Cette fonction de coût sera employée en particulier dans le chapitre 4, dans lequel nous expliquerons plus en détail les motivations de son utilisation avec un cas d'usage concret.

Pour une classification binaire supervisée, le problème d'optimisation est donné par l'équation (1.19) :

$$\mathcal{W}^*, \mathcal{B}^* = \underset{\mathcal{W},\mathcal{B}}{\text{argmin}}\,\frac{1}{|\mathcal{T}_{\text{ent}}|}\sum_{\mathbf{e}\in\mathcal{T}_{\text{ent}}} f_L\Big(f_{\text{NN}}(\mathcal{W}, \mathcal{B}, \mathbf{e}), \mathbf{s}(\mathbf{e})\Big) \tag{1.19}$$

où $\mathcal{W}$ et $\mathcal{B}$ sont respectivement les ensembles des poids et des biais entraînables et f_{NN} la fonction représentant un NN. Elle prend en argument $\mathcal{W}$, $\mathcal{B}$ et une entrée $\mathbf{e}$ du même type que les données de $\mathcal{T}_{\text{ent}}$. $\mathbf{s}(\mathbf{e})$ est la sortie attendue pour l'entrée $\mathbf{e}$. f_L peut ici correspondre à f_{BCE}, f_{FL}, ou encore une variante de f_{BCE}.

Le processus d'entraînement reste le même que celui décrit dans le paragraphe 1.2.4.1. Pour de la classification binaire supervisée, la mise à jour séquentielle des paramètres entraînables par paquet $\mathcal{T}_{\text{ent},p}$ peut se traduire par les équations (1.20) et (1.21) :

$$\mathcal{W} \leftarrow \mathcal{W} - \lambda \nabla_{\mathcal{W}} f_L \left(\frac{1}{\left|\mathcal{T}_{\text{ent},p}\right|} \sum_{\mathbf{e} \in \mathcal{T}_{\text{ent}}} f_L\Big(f_{\text{NN}}(\mathcal{W}, \mathcal{B}, \mathbf{e}), \mathbf{s}(\mathbf{e})\Big) \right) \tag{1.20}$$

$$\mathcal{B} \leftarrow \mathcal{B} - \lambda \nabla_{\mathcal{B}} f_L \left(\frac{1}{\left|\mathcal{T}_{\text{ent},p}\right|} \sum_{\mathbf{e} \in \mathcal{T}_{\text{ent}}} f_L\Big(f_{\text{NN}}(\mathcal{W}, \mathcal{B}, \mathbf{e}), \mathbf{s}(\mathbf{e})\Big) \right) \tag{1.21}$$

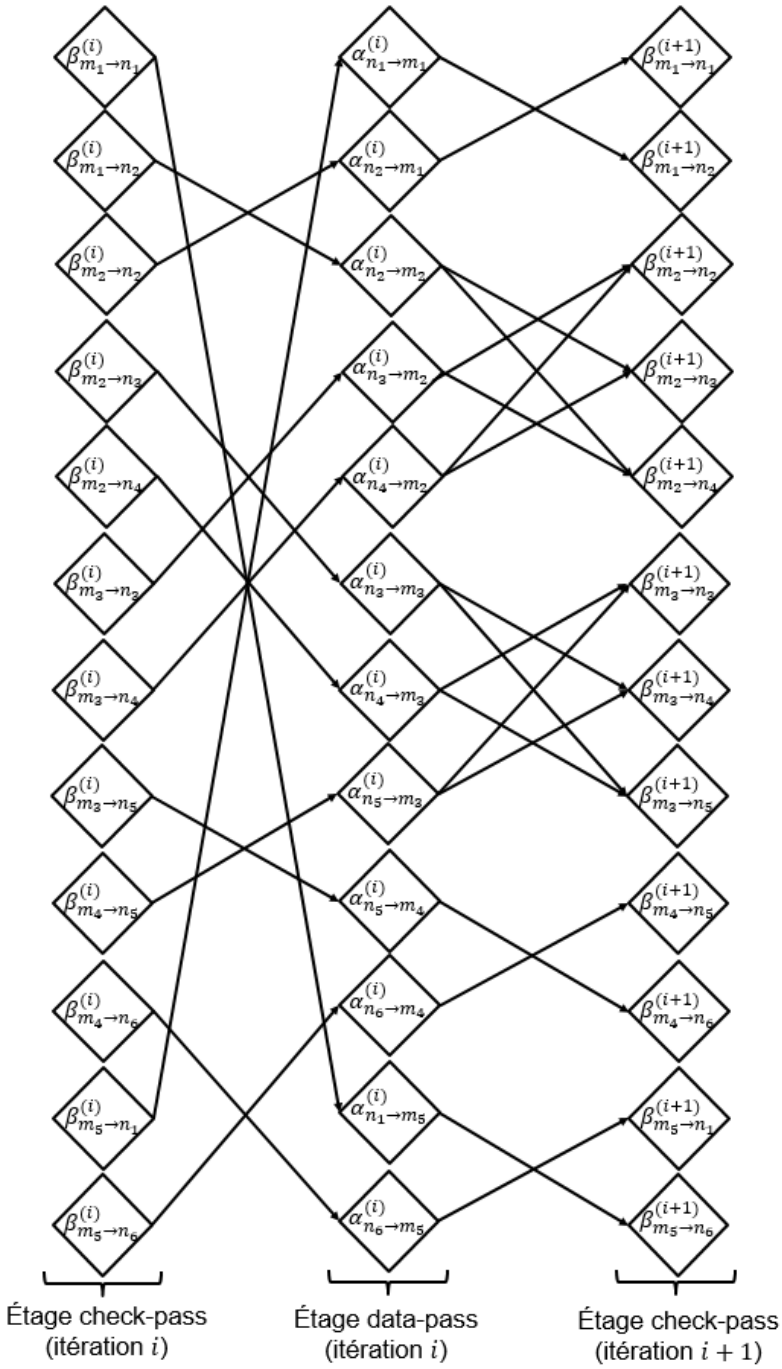
où λ est le pas d'adaptation du gradient et $\nabla_\theta f_L(\theta) = \left[\nabla_{\theta_1} f_L(\theta_1), \ldots, \nabla_{\theta_Q} f_L(\theta_Q)\right]$ est le gradient de f_L par rapport à θ, c'est à dire, le vecteur aux dérivées partielles de la fonction de coût par rapport à chaque composante θ_q du vecteur de paramètres entraînables $\theta \in \mathbb{R}^Q$ (correspondant donc à un vecteur de poids ou un vecteur de biais).

1.3 Décodage LDPC assisté par réseaux de neurones

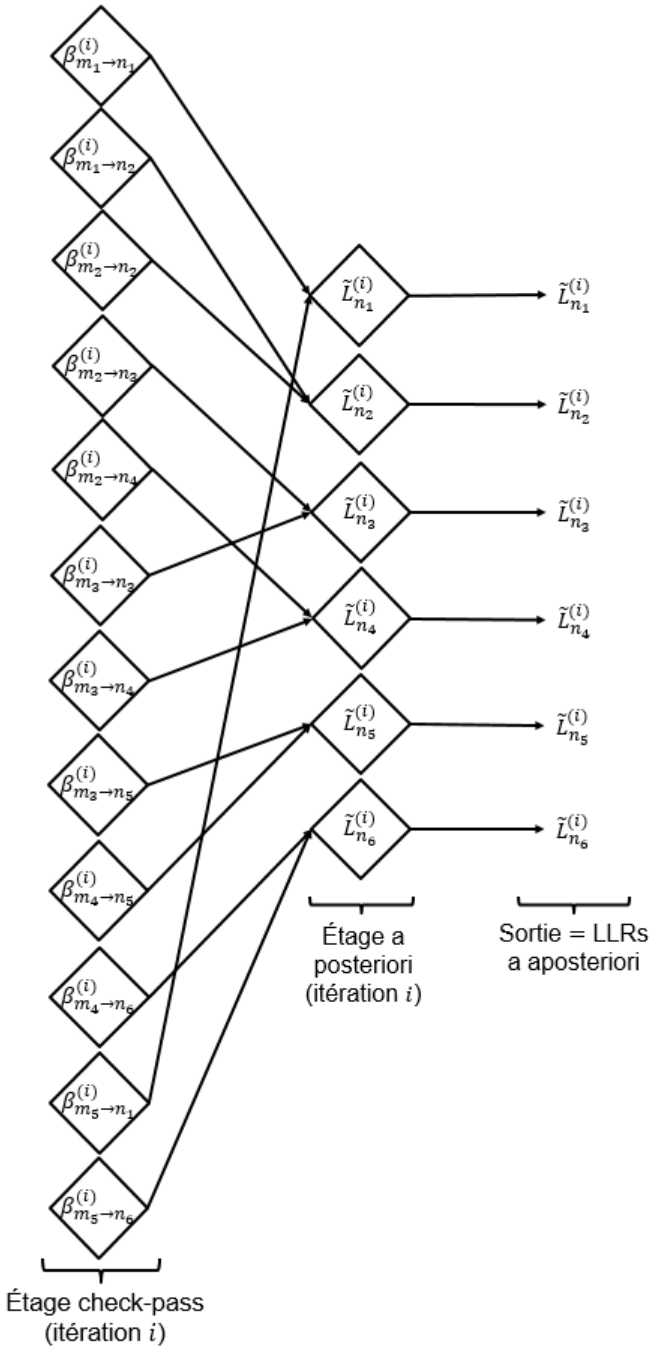
Les réseaux de neurones ayant été introduits, nous pouvons maintenant détailler la modélisation du BP par réseaux de neurones. Rappelons que le but de cette modélisation neuronale est d'améliorer la performance de décodage par rapport au BP en remédiant aux problèmes induits par la faible taille des mots de code. Nous consacrons ainsi cette section à expliquer le BP neuronal, ainsi que certaines approches de l'état de l'art se basant elles aussi sur le BP neuronal.

1.3.1 BP neuronal

La topologie du BP neuronal copie le processus de décodage BP, avec des itérations de décodage déroulées. Des poids sont alors introduits lors des étapes de décodage BP, afin de pondérer les LLRs sortant du canal et les messages circulant sur les arêtes du graphe de Tanner. De plus, le BP neuronal peut avoir une architecture NN-FF ou RNN, les décodeurs correspondants étant appelés BP-FF ou BP-RNN. Plus précisément, l'architecture neuronale sous-jacente (FF ou RNN) contient trois types d'étages neuronaux, chacun correspondant à une étape de l'algorithme BP :



(a) Connexions des étages check-pass et data-pass.



(b) Connexions des étages check-pass et a posteriori.

Figure 1.11 – Connexions entre les étages du BP neuronal du code figure 1.1.

- L'étage check-pass réalise le calcul des messages sortant des noeuds de parité. Cet étage contient un nombre de neurones égal au nombre d'arêtes du graphe de Tanner. Chaque neurone correspond ainsi au calcul d'un message allant d'un noeud de parité m vers un noeud de donnée n. Autrement dit, les neurones de l'étage check-pass du NN correspondent aux arêtes dirigées du graphe de Tanner, $m \longrightarrow n$.
- L'étage data-pass effectue le calcul des messages sortants des noeuds de donnée. Il possède lui aussi autant de neurones que d'arêtes dans le graphe de Tanner. Chaque neurone correspond ainsi à une arête dirigée $n \longrightarrow m$ du graphe et réalise le calcul du message circulant sur cette arête (allant du noeud de donnée n vers le noeud de parité m).
- L'étage a posteriori se compose de N neurones, calculant les LLRs a posteriori des N bits d'un mot transmis.

Les trois étages du NN sont connectées de sorte à ce qu'un étage check-pass, suivi d'un étage data-pass et d'un étage a posteriori forment une itération de décodage BP. Les connexions entre les étages neuronaux pour le code LDPC court présenté à la figure 1.1 sont affichées sur la figure 1.11. Remarquons que chaque neurone d'un étage check-pass est connecté à l'étage data-pass selon l'ensemble $\mathcal{N}(m) \setminus \{n\}$ associé. Par exemple, le neurone calculant $\beta_{m_2 \to n_2}^{(i+1)}$ est relié aux neurones de l'étage data-pass calculant $\alpha_{n_3 \to m_2}^{(i)}$ et $\alpha_{n_4 \to m_2}^{(i)}$ étant donné que $\mathcal{N}(m_2) \setminus \{n_2\}$ contient n_3 et n_4. De la même manière, les ensembles $\mathcal{M}(n) \setminus \{m\}$ sont utilisés pour établir les connexions d'un étage check-pass vers un étage data-pass et les ensembles $\mathcal{M}(n)$ permettent de connecter un étage check-pass à un étage a posteriori. Enfin, le décodage par BP neuronal s'arrête dès que pour une itération donnée $i \in [1, I]$, le mot estimé $\tilde{\mathbf{c}}^{(i)}$ vérifie le syndrome.

Les formules ci-dessous détaillent les équations appliquées au cours des itérations de décodage pour chaque neurone des étages check-pass, data-pass et a posteriori du BP-FF :

$$\beta_{m \to n}^{(i)} = 2 \tanh^{-1}\left(\prod_{n' \in \mathcal{N}(m) \setminus \{n\}} \tanh\left(\frac{\alpha_{n' \to m}^{(i-1)}}{2} \right) \right) \tag{1.22}$$

$$\alpha_{n \to m}^{(i)} = L_{\mathrm{ch},n} + \sum_{m' \in \mathcal{M}(n) \setminus \{m\}} w_{m' \to n \to m}^{(i)} \beta_{m' \to n}^{(i)} \tag{1.23}$$

$$\tilde{L}_n^{(I)} = L_{\mathrm{ch},n} + \sum_{m \in \mathcal{M}(n)} \tilde{w}_{m \to n}^{(I)} \beta_{m \to n}^{(I)} \tag{1.24}$$

Ces équations montrent tout d'abord que par rapport aux équations (1.7) et (1.8) du BP, des poids $(w, \tilde{w})$ sont introduits pour les étapes data-pass (1.23) et a posteriori (1.24). Il est important de remarquer aussi que les poids sont appliqués uniquement sur les messages provenant de l'étape check-pass. En effet, le but de la pondération introduite dans le BP

neuronal est d'atténuer les messages envoyés par les noeuds de parité possédant un grand nombre de petits cycles dans leurs voisinages locaux [10]. En conséquence, optimiser les poids du BP neuronal revient à réduire l'impact des cycles courts présents dans le graphe de Tanner. Précisons également qu'aucun biais n'est ajouté dans les équations (1.23) et (1.24). En effet, un biais n'agit pas directement sur les messages $\beta^{(i)}_{m\to n}$ dans ces formules et ne permet donc pas de réduire spécifiquement l'impact d'un noeud de parité impliqué dans des cycles courts. De plus, les poids sont fonction de l'itération i pour l'architecture FF. Plus précisément, pour l'itération i,

- les poids pondérant les $\beta^{(i)}_{m'\to n}$ dans (1.23) sont notés par $w^{(i)}_{m'\to n\to m}$, où l'indice indique à la fois le neurone correspondant $n \to m$ dans l'étage data-pass et l'arête neuronale entrante du neurone $m' \to n$ de l'étage check-pass.
- les poids pondérant les $\beta^{(i)}_{m\to n}$ dans (1.24) sont notés par $\tilde{w}^{(i)}_{m\to n}$, où l'indice indique le neurone correspondant n dans l'étage a posteriori à l'itération i et l'arête entrante du neurone $m \to n$ de l'étage check-pass à l'itération i.

Il est à noter que des poids peuvent aussi être appliqués spécifiquement sur les LLRs canal. Cependant, il a été observé lors de nos tests qu'un BP-FF entraîné avec ces poids sur les LLRs canal en plus ne présente pas de meilleurs performances de décodage qu'un BP-FF entraîné sans. Enfin, l'étage check-pass (1.22) correspond en réalité à l'équation (1.6) et n'est pas à proprement parlé un étage neuronal comme défini en sous-section 1.2.2. Néanmoins, il est intégré à un réseau de neurones et est ainsi qualifié d'étage neuronal par un léger abus de langage.

Il est aussi important de remarquer que la profondeur du BP-FF est égale au nombre maximal d'itérations de décodage I. Il en résulte que le nombre de paramètres à entraîner augmente aussi avec I pour le BP-FF, ce qui peut rendre son entraînement relativement long. De plus, des problèmes d'évanouissement du gradient sont d'autant plus probables que la profondeur du réseau est grande. Le nombre d'itérations I doit donc rester relativement faible pour le BP-FF [4]. Afin d'éviter ces deux désavantages, la contrainte d'une indépendance des poids selon les itérations de décodage est fixée pour le BP-RNN [10]. Le BP-RNN possède donc moins de paramètres à entraîner que le BP-FF et ainsi un temps d'entraînement plus court que le BP-FF pour un même nombre d'itérations I. Il est à noter que malgré un nombre réduit de paramètres, le BP-RNN présente des performances similaires au BP-FF. Par conséquent, il est plus avantageux d'utiliser un BP-RNN en termes d'entraînement et même d'implémentation matérielle (moins de poids à stocker).

Les équations de l'étage data-pass et de l'étage a posteriori pour le BP-RNN peuvent ainsi se réécrire :

$$\alpha^{(i)}_{n\to m} = L_{\mathrm{ch},n} + \sum_{m'\in\mathcal{M}(n)\setminus\{m\}} w_{m'\to n\to m}\beta^{(i)}_{m'\to n} \tag{1.25}$$

4. I est par exemple limité à 5 dans [10] et à 6 dans [11].

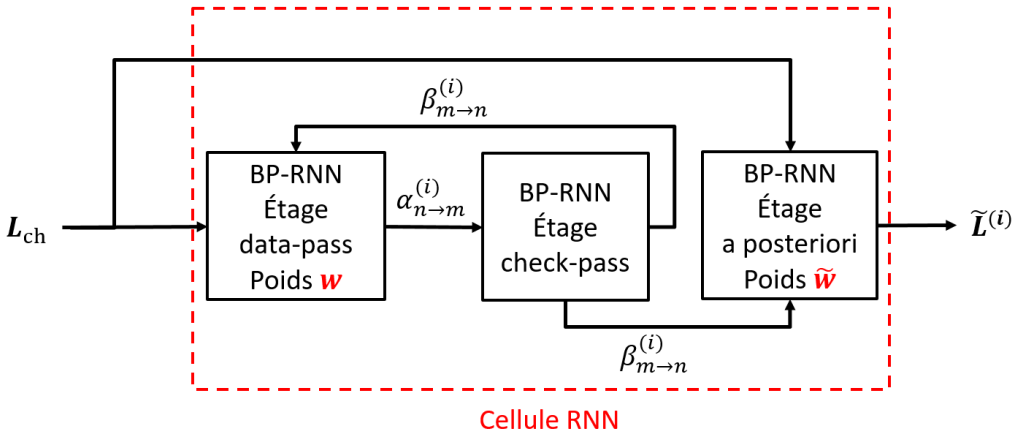


Figure 1.12 – BP-RNN non déroulée.

$$\tilde{L}_n^{(i)} = L_{\text{ch},n} + \sum_{m\in\mathcal{M}(n)} \tilde{w}_{m\to n}\beta^{(i)}_{m\to n} \tag{1.26}$$

De plus, une modélisation également suggérée dans [10] permet de réduire davantage le nombre de poids dans l'étage data-pass :

$$\alpha^{(i)}_{n\to m} = L_{\text{ch},n} + w_{n\to m} \sum_{m'\in\mathcal{M}(n)\setminus m} \beta^{(i)}_{m'\to n}, \tag{1.27}$$

où le poids appliqué $w_{n\to m}$ dépend uniquement du neurone $n \to m$ de l'étage data-pass. Cette simplification réduit la complexité de l'entraînement et permet d'utiliser des architectures de décodage conventionnelles du BP pour une implémentation matérielle efficace (voir la discussion dans la sous-section 1.1.2 et l'équation (1.10), concernant le calcul à moindre coût du data-pass après le calcul des LLRs a posteriori).

Dans les chapitres suivants, nous considérons le BP-RNN utilisant les équations (1.27), (1.22) et (1.26) pour les étages data-pass, check-pass et a posteriori respectivement. La figure 1.12 illustre l'architecture du BP-RNN, avec la configuration de poids choisie [5]. Elle explicite aussi que la cellule RNN correspond bien à une itération de décodage BP. Nous tenons néanmoins à préciser que le travail présenté peut être adapté pour d'autres configurations neuronales, telles que le Min-Sum modélisé par un RNN (MS-RNN).

5. La première utilisation de l'étage data-pass correspond à l'equation (1.5).

1.3.2 Entraînement standard du BP neuronal

1.3.2.1 Jeux d'entraînement et de validation

Le processus d'entraînement du BP neuronal (BP-FF ou BP-RNN) suit la procédure standard d'un réseaux de neurones présentée en sous-section 1.2.4. Il faut ainsi dans un premier temps concevoir les jeux d'entraînement et de validation. Les entrées du BP neuronal sont les LLRs observés. Pour les générer, nous considérons la transmission de mots de codes modulés en BPSK à travers un canal AWGN. Comme le canal AWGN et le BP neuronal sont tout deux symétriques [10], l'entraînement et la validation peuvent être réalisés avec le mot de code tout zéro. Le signal reçu pour un bit d'entrée de canal à +1 s'exprime donc sous la forme :

$$y_n = 1 + z_n, \quad z_n \sim \mathcal{N}(0, \sigma^2),\ n = 1, \ldots, N \tag{1.28}$$

où $\mathcal{N}(0, \sigma^2)$ est la distribution normale de moyenne nulle et de variance σ^2 du canal AWGN. Les LLRs observés sont ensuite calculés en utilisant l'équation (1.4). Les jeux d'entraînement $\mathcal{T}_{\text{ent}}$ et de validation $\mathcal{T}_{\text{val}}$ contiennent alors les LLRs bruités $\mathbf{L}_{\text{ch}}$ pour plusieurs réalisations de transmission du mot tout zéro.

Enfin, indiquons dès maintenant que le jeu de test $\mathcal{T}_{\text{test}}$ du BP neuronal est aussi construit avec (1.28), pour le même SNR que celui d'entraînement. La phase de test du BP neuronal consiste ensuite à calculer son taux d'erreur binaire (Binary Error Rate ou BER en anglais) ou son FER sur le jeu de test.

1.3.2.2 Fonction de coût et problème d'optimisation

Le but du décodage par BP neuronal est de mapper le signal reçu à un mot de code valide. Ce processus peut être interprété comme une tâche de classification, avec une classe correspondant à un mot de code du code linéaire considéré. Cependant, ceci est irréalisable en pratique en raison du nombre élevé de 2^K classes [13]. Une méthode généralement acceptée pour éviter cette difficulté est de remplacer la tâche de classification initiale par une tâche de classification binaire plus simple pour chaque bit du mot de code. La fonction de coût BCE définie en (1.17) est ainsi tout indiquée pour le problème d'optimisation du BP neuronal. Elle se spécifie comme suit :

$$f_{\text{BCE}}\left(\tilde{L}, \mathbf{c}\right) = -\frac{1}{N}\sum_{n=1}^{N}\left(1 - \hat{x}_n\right)\log(1 - \sigma(\tilde{L}_n)) + \hat{x}_n \log(\sigma(\tilde{L}_n)) \tag{1.29}$$

où $\mathbf{c}$ est le mot de code émis et $\sigma(x)$ est la fonction sigmoïde, convertissant la valeur LLR a posteriori $\tilde{L}_n$ en probabilité que le bit décodé soit égal à zéro. Aussi, la valeur de $\tilde{L}_n$ est relevée à la dernière itération de décodage du BP neuronal. En supposant le mot de code tout zéro

est transmis, soit $c_n = 0$ pour $n \in [1, N]$, (1.29) se simplifie en :

$$f_{\text{BCE}}(\tilde{L}) = -\frac{1}{N}\sum_{n=1}^{N}\log(\sigma(\tilde{L}_n)), \tag{1.30}$$

Le problème d'optimisation (1.19) peut aussi se réécrire comme suit [6] :

$$\mathcal{W}^* = \underset{\mathcal{W}}{\text{argmin}}\ \frac{1}{|\mathcal{T}_{\text{ent}}|}\sum_{\mathbf{L}_{\text{ch}}\in\mathcal{T}_{\text{ent}}} f_{\text{BCE}}\Big(f_{\text{BP-RNN}}(\mathcal{W}, \mathbf{L}_{\text{ch}})\Big) \tag{1.31}$$

Ce problème est résolu en minimisant la fonction de coût (1.30) pendant l'entraînement du BP-RNN. Cela revient ainsi à minimiser ainsi la probabilité moyenne d'erreur binaire, c'est à dire le BER.

Des fonctions de coût multiple peuvent aussi être considérées [10, 13]. Elles prennent en compte les LLRs a posteriori après chaque itération de décodage. Cela permet d'amplifier la contribution des premières itérations pour la retro-propagation. Dans [10], l'optimisation d'une fonction de coût multiple produit des gains avec le BP-FF et BP-RNN, pour des codes Bose-Chaudhuri-Hocquenghem (BCH) courts, possédant des matrices de parité de forte densité. Cependant, lorsque pour les mêmes codes des matrices de parité plus clairsemées sont choisis, le gain observé est réduit de manière significative, voire totalement. Dans le chapitre suivant, nous testerons également l'impact de la fonction de coût multiple sur la performance de codes LDPC courts construits avec l'algorithme PEG.

1.3.2.3 Méthode de descente de gradient

Ce paragraphe est dédié au choix de la méthode de descente de gradient employée lors de l'entraînement. Dans [10], la retro-propagation est effectuée avec RMSprop, une méthode basée sur une mise à jour du pas adaptatif [53]. Une moyenne des amplitudes précédentes et récentes du gradient est calculée afin de mettre à jour le pas. RMSprop arrive généralement à mieux entraîner un réseau qu'une simple descente à pas constant. En effet, elle permet tout d'abord d'avoir une stabilité numérique plus importante qu'une descente de gradient classique. Des risques de divergence sont ainsi évités. De plus, comme le pas est adaptatif, le réseau se rapproche plus efficacement de la solution optimale, tout en étant assez rapide en début de descente. Toutefois, il est nécessaire de bien choisir le pas d'initialisation. Il peut effectivement affecter la capacité de l'entraînement à bien converger, c'est à dire à minimiser au mieux la fonction de coût.

Une autre méthode avec pas adaptatif et testée avec le BP-RNN est Adam [54]. Cette méthode estime les moments d'ordre 1 et 2 du gradient pour modifier le pas d'adaptation. Les méthodes Adam et RMSprop nous ont donné des BP-RNNs obtenant les mêmes performances de décodage, pour différents pas d'adaptation initiaux. L'entraînement des BP-RNNs se fera par la suite avec RMSprop, avec un pas d'adaptation initial de 10^{-3}. Cette étude est expliquée plus en détail dans l'annexe A. Le choix de l'initialisation des poids du BP-RNN y est également évalué.

6. Le BP-RNN ne possède pas de biais, ils sont donc absents du problème d'optimisation.

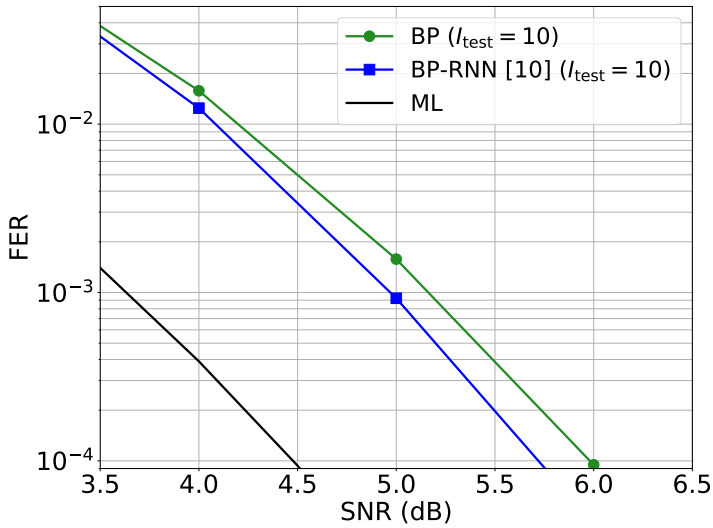
1.3.2.4 Nombre d'itérations de décodage du BP neuronal

Un paramètre important pour l'entraînement du BP neuronal est son nombre maximal d'itérations de décodage. Il impacte en effet les performances du BP neuronal et par conséquent, les valeurs prises par la fonction de coût. L'approche standard est à la fois d'entraîner et de tester le décodeur pour un même nombre maximal d'itérations de décodage, habituellement petits, comme 5 ou 10 itérations [9, 10, 13]. Rappelons que pour le modéle BP-FF, augmenter le nombre d'itérations pour l'entraînement augmente également le nombre de poids à entraîner, ce qui peut devenir très coûteux en termes de complexité calculatoire. Pour le modèle BP-RNN, ce problème est évité étant donné que les poids sont indépendants des itérations. Cependant, en supposant un test du décodeur BP-RNN pour un nombre accru d'itérations de décodage, une question naturelle est de savoir s'il est bénéfique d'utiliser le même nombre d'itérations de décodage pour l'entraînement. A première vue, cela pourrait être le cas. Toutefois, quand le décodage BP ne converge pas, il peut présenter un comportement erratique, en particulier lors des dernières itérations de décodage. Les cycles et les structures problématiques du graphe de Tanner (comme les ensembles absorbants) sont notamment responsables de ce phénomène, puisqu'ils ont tendance à désorienter le décodeur BP au fur et à mesure des itérations. De ce fait, augmenter le nombre d'itérations de décodage pour l'entraînement pourrait amplifier ce comportement et dégrader les résultats de l'optimisation. Lors des résultats présentés dans la suite, nous distinguerons ainsi le nombre maximum d'itérations pendant l'entraînement, noté I_{ent} et le nombre maximum d'itérations pendant le test, noté I_{test}. L'impact de ces paramètres sera notamment évalué.

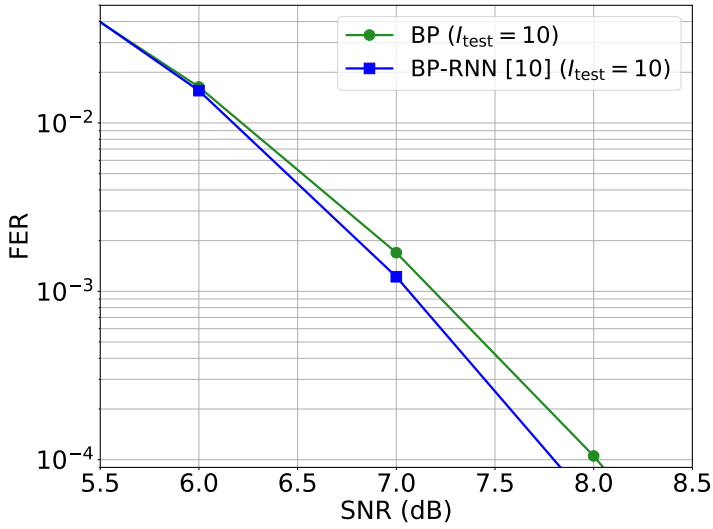
1.3.2.5 Performance FER du BP-RNN

Nous évaluons dans ce paragraphe la performance du BP-RNN [10] en termes de FER pour deux codes LDPC courts construits avec l'algorithme PEG. Le premier code est le Code-1 ($N = 64, K = 32$), explicité plus détail en sous-section 1.1.5. Le second code, le Code-3, a un plus haut rendement que le Code-1, avec $N = 64$ et $K = 46$. Les noeuds de donnée sont de degré $d_n = 3$, tandis que les noeuds de parité sont de degré $d_m = 10$ pour la première moitié et $d_m = 11$ pour la seconde. La circonférence g vaut 4 avec une multiplicité égale à 47. Du fait de ses caractéristiques, le Code-3 possède une plus faible capacité de correction que le Code-1. Le BP-RNN est entraîné comme expliqué précédemment. En outre, le nombre maximal d'itérations fixé pour l'entraînement et le test est égal à 10. Les résultats sont affichés à la figure 1.13 et les gains sont relevés pour un FER de 10^{-4}. Pour le Code-1, le performance de décodage du BP-RNN est modérément améliorée par rapport au BP, avec un gain de 0.25 dB. Ce gain se réduit à 0.18 dB pour le Code-3. Les gains observés restent donc limités pour ces codes LDPC courts construits avec l'algorithme PEG [7]. De nouvelles méthodes de décodage doivent donc être développées afin d'améliorer la capacité de correction des codes LDPC courts dans la région du *waterfall*.

7. À titre de comparaison, les gains pour les codes BCH testés dans [10] sont généralement de l'ordre d'1 dB.



(a) FER du Code-1.



(b) FER du Code-3.

Figure 1.13 – Résultats FER du BP et du BP-RNN standard, avec $I_{\text{test}} = 10$ itérations.

1.3.3 Autres méthodes de décodage par approche neuronale

Nous présentons dans cette section deux approches récentes basées sur le BP neuronal et ses variantes améliorant la capacité de correction des codes LDPC courts. Dans la première [17], une nouvelle méthode d'entraînement du MS neuronal est proposée afin d'obtenir une diversité de décodeurs itératifs à alphabet fini (Finite-Alphabet Iterative Decoders ou

FAID en anglais) neuronaux dans la région du plancher d'erreur. En particulier, les résultats numériques montrent que la notion de diversité se révèle intéressante pour obtenir de meilleures performances. Nous proposerons ainsi dans les chapitres suivants de créer une diversité de décodage afin d'améliorer la capacité de correction des codes LDPC courts dans la région du *waterfall*. La seconde [11] introduit une méthode d'élagage les noeuds de parité d'un BP neuronal afin d'obtenir une matrice de parité différente pour chaque itération de décodage. Cette seconde approche induit en particulier des gains par rapport au BP-FF de [10] pour le Code-2 et servira de référence pour les comparaisons de performances en termes de taux d'erreur paquet.

1.3.3.1 Réseau de neurones pour une diversité de décodeurs FAIDs

Comme expliqué en section 1.1.4, les ensembles piégeants et les ensembles absorbants sont responsables pour les codes LDPC du phénomène de plancher d'erreur. Afin de l'abaisser, les auteurs de [16, 17] ont proposé en 2020 et 2021 de combiner les réseaux de neurones avec la notion de diversité de FAIDs. Un FAID [55] est un décodeur par passage de messages avec une quantification des messages circulant sur les arêtes du graphe (d'où le terme alphabet fini). Une diversité de FAIDs est alors un ensemble de décodeurs dans lequel chaque FAID est capable de corriger des motifs d'erreurs différents, provenant par exemple des ensembles piégants. Cependant, la méthode initiale de conception d'une diversité de FAIDs repose sur une approche de force brute et il apparaît donc intéressant de développer des nouvelles méthodes moins complexes, avec l'aide de réseaux de neurones.

Dans [16], les règles de décodage des FAIDs sont ainsi basées sur celles du MS mis sous forme RNN (MS-RNN) et possèdent une opération de quantification. L'entraînement de ces FAIDs neuronaux permet alors d'apprendre de nouvelles règles de décodage pour des codes LDPC réguliers sur le canal binaire symétrique (Binary Symetric Channel ou BSC en anglais). Les outils nécessaires pour prendre en compte la quantification durant l'entraînement sont notamment expliqués. La notion de FAID est ensuite réutilisée par les mêmes auteurs dans [17]. Ici le modèle neuronal du FAID est plus simple que le MS-RNN, avec des poids uniquement sur les observations du BSC. De plus, afin d'atteindre une diversité de décodage des FAIDs, les différents jeux d'entraînement sont constitués d'erreurs ayant un support d'ensemble piégeant. Plus précisément, leur approche commence par concevoir un FAID initial, D_1, garantissant une bonne capacité de correction dans la région du *waterfall*. Les motifs les plus problématiques non corrigés par D_1 dont le support est un ensemble piégeant sont ensuite déterminés grâce à une méthode d'extension-contraction de sous-graphes [56]. Les autres FAIDs sont ensuite modélisés sous la forme de RNNs quantifiés et leur entraînement est fait de manière séquentielle. En effet, le FAID-RNN D_j, $(j > 1)$, est entraîné sur un jeu d'entraînement composé de tous les motifs d'erreurs n'ayant pas pu être décodés par les précédents FAIDs. Enfin, étant donné que les FAIDs sont entraînés à décoder des motifs d'erreurs, une fonction de coût basée sur une mesure du taux d'erreur paquet est aussi proposée. Il est alors prouvé que pour le code LDPC de Tanner $(N = 155, K = 64)$ [57], la diversité de FAIDs construite à l'aide de RNNs quantifiés possède un plancher d'erreur

abaissé par rapport à celui de la diversité de FAIDs de [55]. Le gain apporté par l'utilisation de plusieurs décodeurs FAIDs au lieu d'un unique FAID est également montré. En conclusion, la notion de diversité combiné avec le FAID neuronal permet d'améliorer les performances d'un code LDPC court dans la région du plancher d'erreur.

Toutefois, il convient de remarquer que la nature en cascade de l'entraînement des décodeurs FAIDs neuronaux peut provoquer un entraînement fastidieux, bien plus long que celui du BP-RNN standard par exemple. Afin de réduire le temps d'entraînement, une diversité de décodeurs neuronaux dans laquelle chaque décodeur est entraîné indépendamment des autres serait à envisager. Ceci permettrait en effet de paralléliser les entraînements. De plus, pour les codes courts, des ensembles de quelques noeuds de donnée seulement (piégeants ou absorbants par exemple) peuvent aussi avoir un impact dans la région du *waterfall*. Nous nous intéresserons ainsi à créer une diversité de décodeurs neuronaux traitant des motifs d'erreurs responsables des dégradations dans la région du *waterfall*.

1.3.3.2 Élagage des noeuds de parité d'un BP-FF

Bien que le BP neuronal est capable d'améliorer le décodage par rapport au BP conventionnel en taille courte [10], la performance reste limitée par la matrice de parité sous-jacente du code, du fait de sa construction sous optimale. Pour adresser ce problème, les auteurs de [11] proposent une méthode d'élagage des noeuds de parité d'un BP neuronal permettant d'obtenir une matrice de parité différente pour chaque itération de décodage.

Pour accomplir ceci, un décodage BP-FF est considéré sur le graphe d'une matrice de parité redondante, donc possédant un nombre de noeuds de parité plus grand que le rang de H. Ce BP-FF possède une configuration de poids différentes de celles présentées en sous-section 1.3.1, avec en particulier l'introduction de poids dans l'étage check-pass :

$$\beta_{m\to n}^{(i)} = 2w_m^{(i)} \tanh^{-1}\left(\prod_{n'\in\mathcal{N}(m)\setminus\{n\}} \tanh\left(\frac{\alpha_{n'\to m}^{(i)}}{2}\right)\right) \tag{1.32}$$

Le poids $w_m^{(i)}$ est appliqué sur la sortie du noeud de parité m utilisé à l'itération i du BP-FF. Il est vu comme une indication de l'importance du noeud de parité correspondant pour le décodage. Les noeuds de parité avec les poids les plus faibles sont ensuite élagués durant un processus d'entraînement itératif. En effet, le BP-FF commence par être initialisé avec des poids à 1. Il est ensuite entraîné de manière à minimiser le BER avec une fonction de coût multiple, prenant ainsi en compte le résultat de chaque itération de décodage. Une fois que l'optimisation a convergé, le poids $w_m^{(i)}$ le plus faible est fixé à 0. Le noeud de parité correspondant est de ce fait élagué. Le BP-FF est alors entraîné de nouveau, avec les poids précédemment optimisés, puis le noeud de parité avec le poids le plus faible est élagué. Ce processus est répété jusqu'à ce qu'une complexité cible soit atteinte (mesurée par le nombre de noeuds de parité), ou qu'une performance cible soit atteinte. Il est à remarquer que comme le poids le plus faible est recherché à travers toutes les itérations, chaque étage check-pass du BP-FF possède un ensemble de noeuds de parité différent.

Enfin, ce processus d'élagage est étendu à une version NN-FF d'une des variantes du BP, le MS avec offset. Un phénomène de quantification est aussi introduit sur les poids, les offsets, les LLRs du canal et les messages circulant dans le graphe de Tanner. Les performances du BP-FF élagué et du MS-FF avec offset élagué sont ensuite évaluées pour plusieurs codes, dont le Code-2 (code CCSDS). En particulier, le décodeur BP-FF neuronal élagué, noté PB-NBP D_1 dans [11], présente de meilleures performances que BF-FF ou le BP. Toutefois, PB-NBP D_1 se situe à 1.5 dB de la performance ML pour un FER de 10^{-4}. L'écart à la performance ML n'est donc pas entièrement comblé. De plus, la méthode d'entraînement de PB-NBP D_1 possède une complexité calculatoire très importante par définition, puisqu'une optimisation entière des paramètres est effectuée à l'élagage de chaque noeud de parité.

1.4 Conclusion

Ce chapitre a permis d'expliciter les notions fondamentales pour le décodage des codes LDPC courts par réseaux de neurones. Dans ce chapitre introductif, nous avons ainsi rappelé la description des codes LDPC par leur graphe de Tanner, puis nous avons présenté leur algorithme de décodage itératif par passage de messages extrinsèques, le BP. Il a été expliqué qu'avec un mot de code de faible taille, les graphes bipartis présentent inévitablement de nombreuses structures problématiques, telles que les cycles et les ensembles absorbants, dégradant fortement la performance du BP et l'empêchant de ce fait d'atteindre la performance ML. Dans le même temps, les structures classiques de réseaux de neurones FF et RNN ont été décrites. Le processus d'entraînement a aussi été détaillé, puis précisé pour un problème de classification binaire supervisée. Enfin la dernière section a été consacrée à la modélisation du BP sous la forme d'un NN-FF et d'un RNN, ainsi qu'à leur entraînement standard. Elle a aussi permis d'introduire deux autres approches neuronales récentes et intéressantes pour le décodage : la diversité de FAIDs neuronaux et le BP-FF résultant d'un élagage des noeuds de parité. Cependant, nous avons montré que dans la région du *waterfall*, la performance du BP-RNN reste peu améliorée par rapport au BP pour des codes LDPC courts construits avec l'algorithme PEG. De plus, le BP-FF élagué ne permet pas de combler entièrement l'écart à la performance ML. La diversité FAIDs est quant à elle spécifique au phénomène de plancher d'erreur et ne répond donc pas au problème d'amélioration des performances dans la région du *waterfall*.

Notre objectif dans la suite du manuscrit est par conséquent d'améliorer la performance du BP-RNN dans la région du *waterfall* et de se rapprocher de la performance ML. Pour réaliser ceci, nous proposons dans le chapitre suivant une première approche visant à créer une diversité de décodeurs BP-RNNs pour la région du *waterfall*.

Chapitre 2

Spécialisation du BP-RNN et diversité de décodage

Contenu du Chapitre 2

2.1 Motivation de la spécialisation 38
2.2 BP-RNNs spécialisés 39
2.2.1 Classification des supports des motifs d'erreurs 40
2.2.2 Jeu d'entraînement spécialisé 41
2.2.3 Architecture parallèle de BP-RNNs 42
2.2.4 Sélection optimale par complémentarité 43
2.3 Résultats FER 45
2.3.1 Paramètres de simulation 45
2.3.2 BP-RNNs spécialisés 46
2.3.3 MS-RNNs spécialisés 49
2.3.4 Impact de la configuration de poids 50
2.3.5 BP-RNNs entraînés avec une fonction de coût multiple 51
2.3.6 Analyse des classes sélectionnées 52
2.4 Conclusion 54

Le BP-RNN entraîné comme décrit obtient des gains importants pour des codes BCH courts, possédant des matrices de parité de forte densité [10]. Cependant, avec des codes LDPC courts (e.g., construits par l'algorithme PEG), le BP-RNN ne présente que des gains limités.

Afin de remédier au gain limité sur les codes LDPC courts, nous proposons dans ce chapitre de créer une diversité de décodeurs BP-RNNs pour la région du *waterfall*. Cette approche est ainsi motivée par [16, 17, 55], où les règles de décodage pour les décodeurs itératifs à alphabet fini FAIDs ont été conçues ou apprises pour corriger des motifs d'erreurs avec un support d'ensemble piégeant dans le cadre du canal symétrique binaire. Afin d'obtenir une diversité de BP-RNNs, les défauts du jeu d'entraînement standard du BP-RNN responsables des gains limités sont tout d'abord explicités. Nous présentons en conséquence une nouvelle méthode d'entraînement spécialisant le décodeur BP-RNN sur des motifs d'erreurs partageant des propriétés structurelles similaires. Cette approche est ensuite associée avec une nouvelle architecture de décodage composée de plusieurs BP-RNNs spécialisés (donc une diversité de décodage) mis en parallèle, dans laquelle chaque BP-RNN est entraîné à corriger un type différent de motif d'erreurs. Enfin, nous nous concentrons en particulier sur des petits motifs d'erreurs, ce qui rend l'approche proposée dans ce chapitre pertinente pour des codes LDPC courts corrigeant un petit nombre d'erreurs.

2.1 Motivation de la spécialisation

Les gains limités du BP-RNN sur les Code-1 et Code-3 peuvent s'expliquer par l'analyse du jeu d'entraînement standard du BP-RNN. Il contient en effet des mots de code bruités générés aléatoirement, ce qui implique une grande variété de motifs d'erreurs, en termes de taille, de difficulté de décodage et de structures dans les sous-graphes induits. En particulier, la probabilité d'obtenir un motif de E erreurs dans un mot de code bruité constitue une première métrique intéressante pour analyser le jeu d'entraînement standard. Le tableau 2.1 donne ainsi le calcul de $P_{\text{err}}(E)$ avec l'équation (1.12) pour un nombre d'erreurs E allant de 0 à 4, aux SNRs 6 dB et 8 dB. Notons que 6 dB (resp. 8 dB) correspond au SNR pour lequel le taux d'erreur paquet du BP du Code-1 (resp. Code-3) est de l'ordre de grandeur de 10^{-4}.

Tableau. 2.1 – $P_{\text{err}}(E)$ vs E.

E	$P_{\text{err}}(E)$ **à** SNR =6 dB	$P_{\text{err}}(E)$ **à** SNR =8 dB
0	0.2254	0.6801
1	0.3397	0.2629
2	0.2520	0.0500
3	0.1226	0.0062
4	0.0440	0.0005

Ce tableau montre tout d'abord que des mots bruités sans erreur ($E = 0$) peuvent être présent dans le jeu d'entraînement, avec une probabilité de 0.2254 (resp. 0.6801) à 6 dB (resp.

8 dB). Par définition, ces mots n'ont pas besoin de décodage et ils ne sont donc pas utiles pour l'entraînement du BP-RNN. Ensuite, même si des mots bruités possèdent effectivement des erreurs ($E > 0$), il n'est pas assuré qu'entraîner un BP-RNN à les décoder soit réellement nécessaire. En effet, le BP-RNN est initialisé avec le BP, étant donné que ses poids avant entraînement valent tous 1. Par conséquent, si le BP conventionnel arrive à corriger aisément un mot bruité contenant des erreurs, le BER sera minimisé et la fonction de coût associée (1.30) prendra ainsi une valeur très faible. La descente de gradient résultante ne modifiera donc quasiment pas les poids du BP-RNN. Autrement dit, le BP-RNN ne pourra rien apprendre de nouveau avec un mot bruité facilement décodé par le BP. Ce type de mot peut néanmoins constituer une partie non négligeable du jeu d'entraînement. Par exemple, les mots bruités contenant une seule erreur sont facilement décodés par le BP (sauf cas très peu probable, lorsque le bruit sur le bit erroné serait particulièrement élevé). Ils constituent pourtant environ 34% du jeu d'entraînement à 6 dB selon le tableau 2.1. Un premier problème du jeu d'entraînement standard vient donc d'une mauvaise représentation des motifs d'erreurs pour lesquels un apprentissage du décodage par BP-RNN est réellement nécessaire pour améliorer les performances dans la région du *waterfall*.

Un second problème du jeu d'entraînement standard relève de la variété des motifs d'erreurs en termes de sous-graphe induit. En effet, les motifs d'erreurs impactent différemment le décodage BP en fonction des propriétés structurelles de leur sous-graphe induit. Par exemple, le décodage de motifs avec un support d'ensemble absorbant [33] est réputé être compliqué à cause d'un nombre de noeuds de parité satisfaits plus important que ceux non satisfaits pour chaque noeud de donnée du sous-graphe induit. Une optimisation particulière du BP-RNN peut ainsi être nécessaire afin qu'il puisse apprendre à décoder des motifs possédant des propriétés structurelles spécifiques. Dans le cas contraire, si trop de types de sous-graphes sont présents dans le jeu d'entraînement, le BP-RNN risque de ne pas saisir les particularités de chaque type et n'apprendra donc pas à les décoder de manière optimale.

Nous proposons une nouvelle méthode d'entraînement motivée par une meilleure représentation des motifs d'erreurs dégradant effectivement les performances dans la *waterfall*, ainsi qu'une nécessité de spécialiser le BP-RNN en fonction des propriétés structurelles de leurs sous-graphes induits. Cette approche est présentée dans la section suivante.

2.2 BP-RNNs spécialisés

La section précédente a permis d'expliciter les motivations de notre première contribution à savoir la création d'une nouvelle méthode d'entraînement pour le BP-RNN. Son objectif est de spécialiser le décodeur BP-RNN sur des motifs d'erreurs de petite taille et partageant des propriétés structurelles similaires dans le sous-graphe induit. Cette section détaille ainsi les étapes menant à la spécialisation du BP-RNN. Ensuite, une architecture de décodage parallèle constituée de plusieurs BP-RNNs spécialisés est décrite. Enfin, pour limiter le nombre de décodeurs utilisés dans l'architecture parallèle, nous proposons une méthode de sélection, où les BP-RNNs sélectionnés sont les plus complémentaires entre eux

en termes des erreurs qu'ils arrivent à décoder.

2.2.1 Classification des supports des motifs d'erreurs

La première étape de la spécialisation consiste en une classification d'ensembles de noeuds de donnée, selon leur taille et leurs propriétés structurelles dans le sous-graphe de Tanner induit.

Pour réaliser ceci, nous définissons tout d'abord le type absorbant d'un ensemble de noeuds de donnée $\mathcal{V}$ comme ν-(ω, ε), avec ν le cardinal de $\mathcal{V}$, ω le nombre de noeuds de parité connectés un nombre impair de fois à $\mathcal{V}$ et ε le nombre de noeuds de parité connectés un nombre pair de fois à $\mathcal{V}$. De manière plus formelle, en utilisant les notations $O(\mathcal{V})$ et $\mathcal{E}(\mathcal{V})$ introduites dans la sous-section 1.1.4 :

$$\nu = \mathbf{card}(\mathcal{V}) \tag{2.1}$$

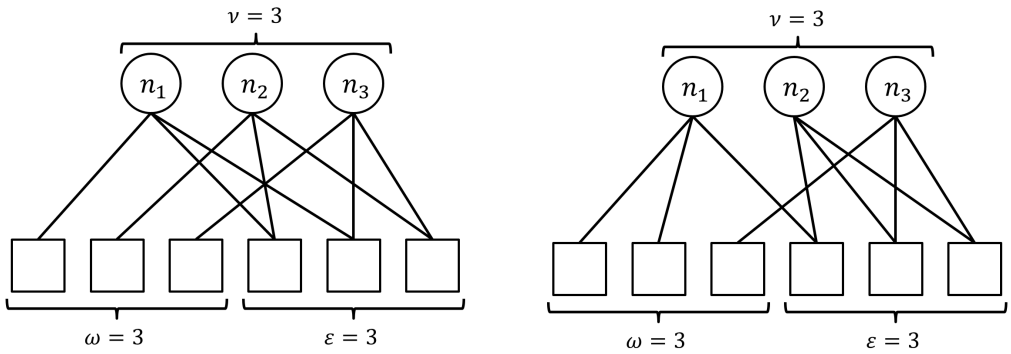
$$\omega = \mathbf{card}(O(\mathcal{V})) \tag{2.2}$$

$$\varepsilon = \mathbf{card}(\mathcal{E}(\mathcal{V})) \tag{2.3}$$

Nous pouvons remarquer que le type absorbant indique le nombre total de noeuds de parité (non satisfait, satisfait), dans le cas où les noeuds de donnée de $\mathcal{V}$ sont tous erronés. Notons également que le type absorbant est défini pour tout ensemble $\mathcal{V}$, qu'il soit un ensemble absorbant ou non.

Il est à noter que des ensembles $\mathcal{V}$ ayant un même type absorbant peuvent induire des sous-graphes de Tanner différents. Un tel exemple est illustré dans la figure 2.1, pour deux ensembles de noeuds de donnée de type absorbant 3-(3, 3). En effet, pour l'ensemble $\mathcal{V}$ du cas (b), nous pouvons observer que le noeud de donnée n_1 est connecté à deux noeuds de parité de degré impair et un de degré pair, tandis que sur le cas (a), ce même noeud n_1 est connecté à un noeud de parité de degré impair et deux de degré pair.

Par conséquent, nous proposons de caractériser davantage les graphes en calculant le profil de connexions de chaque noeud de donnée. Il est défini par $p_n := (m_{n,1}, m_{n,2}, \dots)$, où $m_{n,d}$ est le nombre le nombre de noeuds de parité connectés au noeud de donnée n et de degré d dans le sous-graphe induit. Il convient de préciser que la séquence $m_{n,d}$ est en réalité finie, de longueur égale au degré maximum des noeuds de parité dans le sous-graphe induit correspondant. Les p_n des noeuds d'un ensemble $\mathcal{V} = \{n_1, \dots, n_\nu\}$ sont ensuite utilisés pour former une liste de profils de connexion des noeuds de donnée, définie par $\mathfrak{P} := [p_{n_1}, \dots, p_{n_\nu}]$. La classe ν-$(\omega, \varepsilon, \mathfrak{P})$ contient alors tous les ensembles de noeuds de donnée $\mathcal{V}$ possèdant un type absorbant ν-(ω, ε) et une liste de profils de connexions égale à $\mathfrak{P}$ ou à l'une des permutations de $\mathfrak{P}$. Les ensembles de noeuds de donnée illustrés dans la Fig. 2.1 sont associés aux classes 3-(3, 3, [(1, 2), (1, 2), (1, 2)]) et 3-(3, 3, [(2, 1), (0, 3), (1, 2)]), respectivement.



(a) Graphe induit possible n°1 **(b)**

Figure 2.1 – Exemple de deux ensembles $\mathcal{V}$, avec $\mathbf{card}(\mathcal{O}(\mathcal{V})) = \mathbf{card}(\mathcal{E}(\mathcal{V})) = 3$.

D'un point de vue algorithmique, les ensembles $\mathcal{V}$ sont calculés exhaustivement jusqu'à une certaine taille fixée $\nu_{\max}$, puis classifiés séquentiellement selon leur type absorbant et leur liste de profils de connexions des noeuds de donnée. Plus précisément, pour classifier un ensemble $\mathcal{V}$, son type absorbant et les 2^ν permutations de sa liste de profils de connexion des noeuds de donnée sont calculés. Pour chaque classe ν-$(\omega, \varepsilon, \mathfrak{P})$ en cours de construction, nous comparons ensuite ν-(ω, ε) avec le type absorbant de $\mathcal{V}$, puis les 2^ν permutations avec $\mathfrak{P}$. Si $\mathcal{V}$ possède le type absorbant ν-(ω, ε) et que l'une des permutations est égale à $\mathfrak{P}$, alors $\mathcal{V}$ est ajouté à la classe ν-$(\omega, \varepsilon, \mathfrak{P})$. Dans le cas où $\mathcal{V}$ n'appartient à aucune classe, une nouvelle classe est créée, avec les caractéristiques de $\mathcal{V}$. La complexité de la classification provient essentiellement des tests de profils de connexion, avec à chaque test le calcul de 2^ν permutations et 2^ν comparaisons. Toutefois, comme des petits motifs d'erreurs ($\nu \leq 4$) sont traités, la complexité de cette procédure reste raisonnable.

2.2.2 *Jeu d'entraînement spécialisé*

Comme pour l'entraînement standard du BP-RNN, nous considérons un canal AWGN à entrée binaire, issue d'un alphabet BPSK ±1 et de variance fixée σ^2. La transmission du mot de code tout zéro est aussi supposée. De plus, pour un mot reçu $\mathbf{y}$, nous définissons son ensemble d'erreurs par $\mathcal{E}(\mathbf{y}) := \{n \mid y_n \leq 0\} \subset \{1, \ldots, N\}$. L'objectif est de construire un jeu d'entraînement spécifique, contenant uniquement des mots $\mathbf{y}$ tels que le motif d'erreurs défini par $\mathcal{E}(\mathbf{y})$ appartienne à une classe donnée ν-$(\omega, \varepsilon, \mathfrak{B})$. Étant donné qu'un réseau de neurones se spécialise pour traiter les propriétés majeures du jeu d'entraînement [58], nous nous attendons à ce que le BP-RNN entraîné soit spécialisé dans le décodage des mots $\mathbf{y}$ tels que $\mathcal{E}(\mathbf{y})$ est (ou contient) un motif d'erreurs de la classe donnée ν-$(\omega, \varepsilon, \mathfrak{B})$.

Une approche naive pour construire un tel jeu d'entraînement serait de générer aléatoirement les mots $\mathbf{y}$ avec (1.2), puis de sélectionner ceux pour lesquels $\mathcal{E}(\mathbf{y})$ appartienne à la

classe ν-$(\omega, \varepsilon, \mathfrak{B})$. Néanmoins, cette méthode serait plutôt fastidieuse et longue par définition. Nous proposons ici une procédure plus efficace produisant des erreurs uniquement sur les positions désirées grâce à notre connaissance des ensembles de noeuds de donnée appartenant à la classe ν-$(\omega, \varepsilon, \mathfrak{B})$ et à l'utilisation d'une distribution gaussienne tronquée. Plus précisément, nous choisissons tout d'abord un ensemble de noeuds de donnée $\mathcal{V}$ appartenant à la classe ν-$(\omega, \varepsilon, \mathfrak{B})$ traitée. Dans le cas du mot de code tout zéro modulé en BPSK, un bit est considéré erroné uniquement si son observation canal est inférieure à zéro. Par conséquent, $\mathcal{E}(\mathbf{y}) = \mathcal{V}$ si et seulement si pour chaque noeud de donnée $n \in \mathcal{V}$, l'amplitude de z_n est inférieure à -1. Nous générons ainsi un mot aléatoire $\mathbf{y} = [y_n = 1 + z_n]_{n=1,\dots,N}$, avec

$$z_n \sim \begin{cases} \mathcal{N}(0, \sigma^2, -\infty, -1), & \text{si } n \in \mathcal{V} \\ \mathcal{N}(0, \sigma^2, -1, \infty), & \text{sinon} \end{cases} \tag{2.4}$$

où $\mathcal{N}(0, \sigma^2, a, b)$ est la distribution normale tronquée de moyenne 0 et de variance σ^2, prenant valeur dans l'intervalle (a, b). Le jeu d'entraînement est obtenu en répétant la procédure ci-dessus plusieurs fois. De cette façon le jeu d'entraînement est représentatif de la classe ν-$(\omega, \varepsilon, \mathfrak{B})$. Par conséquent, un BP-RNN entraîné sur ce jeu devient spécialisé sur le décodage des motifs d'erreurs correspondant à la classe ν-$(\omega, \varepsilon, \mathfrak{B})$. Enfin, le jeu de validation, utilisé pour surveiller le bon déroulement de l'entraînement, est généré de la même façon que le jeu d'entraînement.

2.2.3 *Architecture parallèle de BP-RNNs*

En utilisant la procédure de la sous-section 2.2.2, nous entraînons un décodeur BP-RNN spécialisé pour chaque classe ν-$(\omega, \varepsilon, \mathfrak{B})$ [1], avec une taille maximale des motifs $\nu_{\max}$. Le choix de $\nu_{\max}$ sera notamment discuté dans la sous-section 2.3.1. Nous désignons par J le nombre total de décodeurs BP-RNNs spécialisés résultants. L'ensemble de ces J BP-RNNs spécialisés forment une diversité de décodage, puisque que chaque BP-RNN est entraîné à décoder un type spécifique de motifs d'erreurs. Cette diversité de décodage est appelée diversité de BP-RNNs en suivant une terminologie similaire à [16, 17, 55] et est notée par l'ensemble $\mathcal{D}_J$.

Pour organiser une diversité $\mathcal{D}_J$ de J BP-RNNs spécialisés, nous proposons une architecture de décodage parallèle, où l'ensemble des décodeurs sont exécutés en parallèle. Les BP-RNNs spécialisés de $\mathcal{D}_J$ sont notés par $D_1, \dots, D_J$, avec une numérotation arbitraire de 1 à J. La figure 2.2 présente l'architecture parallèle proposée, utilisant les J BP-RNNs spécialisés.

Dans l'architecture de décodage parallèle, chaque décodeur produit une estimation $\tilde{\mathbf{c}}_j = (\tilde{c}_{j,1}, \dots, \tilde{c}_{j,N}) \in \{0, 1\}^N$ du mot de code transmis $\mathbf{c}$. L'ensemble des $\tilde{\mathbf{c}}_j$ vérifiant le syndrome est défini par $\mathcal{S} := \{\tilde{\mathbf{c}}_j \mid \text{syndrome}(\tilde{\mathbf{c}}_j) = 0\}$. Si au moins l'un des mots estimés vérifie le syndrome ($\mathcal{S} \neq \emptyset$), un critère ML est utilisé pour déterminer le mot de code en sortie

1. Nous excluons uniquement les classes pour lesquelles $\omega = 0$. Elles correspondent en effet à des motifs dont le support est un mot de code non nul. De tels motifs d'erreurs ne peuvent pas être détectés ou corrigés.

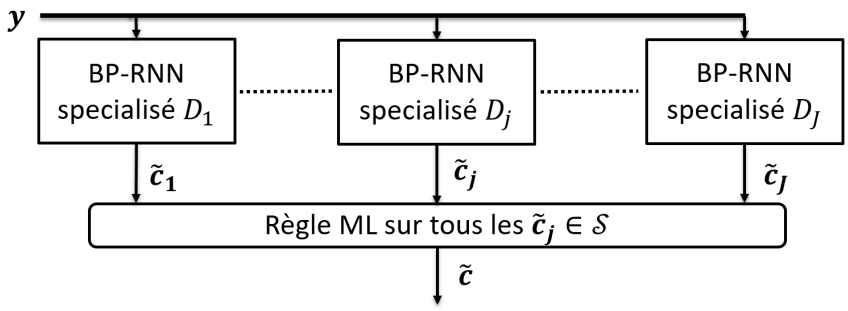


Figure 2.2 – Architecture de décodage parallèle.

l'architecture de décodage parallèle. Pour le canal AWGN à entrée binaire (avec des entrées ± 1), le critère ML s'écrit :

$$\tilde{\mathbf{c}} = \underset{\tilde{\mathbf{c}}_j \in \mathcal{S}}{\operatorname{argmax}}\ P(\tilde{\mathbf{c}}_j | y) = \underset{\tilde{\mathbf{c}}_j \in \mathcal{S}}{\operatorname{argmax}} \sum_{n=1}^{N} y_n \tilde{c}_{j,n}, \tag{2.5}$$

Le décodage est alors réussi si $\tilde{\mathbf{c}}$ est égal au mot de code transmis. Si aucun des décodeurs BP-RNNs ne converge vers un mot de code ($\mathcal{S} = \emptyset$), le décodage échoue. Dans un tel cas, le critère ML est appliqué sur les J mots estimés $\tilde{\mathbf{c}}_j$ afin de sélectionner celui minimisant le BER du décodeur

2.2.4 *Sélection optimale par complémentarité*

En pratique, il est désirable de réduire le nombre de décodeurs exécutés en parallèle afin d'éviter une trop grande complexité calculatoire et matérielle. En effet, chaque BP-RNN en parallèle possède un coût matériel dédié au processus itératif (calcul et instanciations en mémoire des LLRs a posteriori et des messages circulant sur le graphe), ainsi qu'un coût matériel lié au stockage et multiplications induits par les poids optimisés. De plus, nous voulons éviter des effets d'entraînement similaires entre les BP-RNNs, pouvant survenir par exemple si des motifs d'erreurs d'une taille donnée peuvent se subdiviser en motifs de plus petites tailles indépendants. La figure 2.3 illustre ce phénomène en prenant l'exemple de la classe $3 - (5, 2, [(1, 2), (1, 2), (3, 0)])$, dont les motifs associés de taille 3 sont en réalité composés de deux motifs indépendants. En particulier, n_3 n'est connecté qu'a des noeuds de parité de degré un. Ainsi, si $\{n_1, n_2, n_3\}$ sont en erreurs, le BP-RNN pourra assez aisément corriger n_3 et le motif d'erreurs se réduira à un ensemble absorbant de type $2 - (2, 2, [(1, 2), (1, 2)])$. Les BP-RNNs entraînés respectivement sur les classes $3 - (5, 2, [(1, 2), (1, 2), (3, 0)])$ et $2 - (2, 2, [(1, 2), (1, 2)])$ risquent ainsi de converger fréquemment vers les mêmes mots de code, ce qui rend les décodages effectués en parallèle redondants.

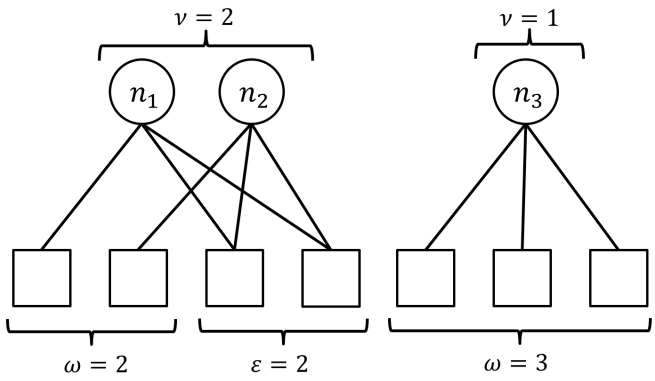


Figure 2.3 – Motifs d'erreurs de la classe $3 - (5, 2, [(1, 2), (1, 2), (3, 0)])$, composés de deux motifs indépendants de la classe $2 - (2, 2, [(1, 2), (1, 2)])$.

Pour réduire cette redondance, nous proposons ici une méthode pour sélectionner de manière optimale Z ($Z \in [1, J]$) BP-RNNs spécialisés parmi les J BP-RNNs de $\mathcal{D}_J$. Pour réaliser ceci, nous évaluons la complémentarité des décodeurs entraînés selon leurs probabilités conjointes d'échecs. Nous générons tout d'abord un jeu de test $\mathcal{T}_{\text{test}}$, contenant des mots bruités aléatoires $\mathbf{y}$ comme défini en (1.2). Ce jeu est utilisé pour estimer la capacité de correction individuelle de chaque décodeur entraîné. Nous notons par $\mathcal{F}_j \subset \mathcal{T}_{\text{test}}$ le sous-ensemble des mots bruités sur lesquels le BP-RNN D_j a échoué, avec $j = 1, \ldots, J$. Nous énumérons ensuite toutes les combinaisons d'indices de décodeurs $\mathcal{J}_Z$ de taille Z. La liste d'indices $\mathcal{J}_Z^{\text{opt}}$ telle que l'intersection des F_j correspondants soit la plus petite est alors déterminée avec la règle suivante :

$$\underset{\mathcal{J}_Z}{\operatorname{argmin}} \mathcal{J}_Z^{\text{opt}} = \underset{\mathcal{J}_Z}{\operatorname{argmin}} \left| \cap_{j \in \mathcal{J}_Z} \mathcal{F}_j \right| \tag{2.6}$$

Si l'argument minimum de (2.6) n'est pas unique, un choix arbitraire est fait parmi les combinaisons. La combinaison optimale de Z BP-RNNs $\mathcal{D}_Z^{\text{opt}}$ est alors définie comme suit :

$$\mathcal{D}_Z^{\text{opt}} := \left\{ D_j \mid j \in \mathcal{J}_Z^{\text{opt}} \right\} \tag{2.7}$$

L'équation (2.6) nous garantit que $\mathcal{D}_Z^{\text{opt}}$ est la sélection de Z décodeurs parmi les J BP-RNNs mis en parallèle donnant la meilleure performance en termes de FER (d'où le terme de sélection optimale). $\mathcal{D}_Z^{\text{opt}}$ est donc la combinaison des Z décodeurs les plus complémentaires entre eux, maximisant le nombre de mots bruités corrigés lorsqu'ils sont exécutés ensemble en parallèle. La valeur de Z peut être choisie pour obtenir des meilleurs compromis performance/complexité.

2.3 RÉSULTATS FER

Nous présentons dans cette section les résultats en termes de FER obtenus avec les BP-RNNs spécialisés mis en parallèle. De plus, des tests ont aussi été effectués avec des Min-Sum-RNNs (MS-RNNs) spécialisés, le principe restant le même que pour les BP-RNNs spécialisés. Enfin, l'impact de différentes configurations de poids pour le BP-RNN et l'utilisation d'une fonction de coût multiple sont également évalués.

2.3.1 Paramètres de simulation

Les codes LDPC courts utilisés pour nos simulations sont le Code-1, le Code-3, ainsi que le Code-4, construit lui aussi avec l'algorithme PEG. Les paramètres sont rappelés/détaillés dans le tableau 2.2, avec en particulier $n_{\text{4-cycles}}$ le nombre de cycle de taille 4.

Tableau. 2.2 – Paramètres des codes LDPC évalués.

	N	K	R_c	d_n	d_m	$\mathbf{n}_{\text{4-cycles}}$
Code-1	64	32	0.5	3	6	0
Code-3	64	46	0.71	3	10-11	47
Code-4	128	105	0.81	3	16-17	1130

Les nombres maximaux d'itérations I_{ent} et I_{test} ont été fixées à dix, pour l'ensemble des décodeurs. Au niveau de la classification des motifs d'erreurs, nous avons fixé $\nu = 2, 3, 4$ pour le Code-1 et $\nu = 2, 3$ pour le Code-3 et le Code-4. Ces choix de ν permettent de traiter des tailles d'événements d'erreurs adaptées aux capacités de correction des nos codes LDPC courts, sans rendre la classification trop complexe. [2]

La procédure de classification présentée en sous-section 2.2.1 a été appliquée pour les trois codes. Le nombre de classes ν-$(\omega, \varepsilon, \mathfrak{B})$ obtenues pour chaque code en fonction de ν est donné dans le tableau 2.3. Le nombre total de classes J indique le nombre de décodeurs BP-RNNs entraînés et utilisés dans l'architecture de décodage parallèle.

Un BP-RNN a été entraîné de manière indépendante pour chaque classe obtenue, avec la méthode de construction du jeu d'entraînement décrite dans la sous-section 2.2.2. Ceci nous assure que chaque BP-RNN puisse saisir les propriétés structurelles des motifs d'erreurs de la classe utilisée pour construire son jeu d'entraînement. Les J entraînements peuvent être effectués en parallèle, ce qui permet d'obtenir un temps d'entraînement minimal. Ceci diffère donc de la stratégie d'entraînement des FAIDs neuronaux, où un décodeur FAID D_j

2. À des fins de comparaisons, un code BCH avec $(N, K) = (63, 45)$ ou $(N, K) = (127, 106)$ possède une distance minimale de 7 et peut ainsi corriger 3 erreurs. Un code BCH($N = 63, K = 36$) peut quant à lui corriger 5 erreurs.

Tableau. 2.3 – Nombre de classes obtenues ν-$(\omega, \varepsilon, \mathfrak{B})$ selon ν.

ν	Code-1	Code-3	Code-4
2	2	3	3
3	5	10	11
4	10	N/A	N/A
J	17	13	14

Tableau. 2.4 – Paramètres Keras.

Paramètres	**Valeurs des paramètres**
Méthode de descente de gradient	RMSprop [53] (initialisé avec un pas d'adaptation de 10^{-3})
Nombre d'époques	10
Taille des paquets d'entraînement	8192
Nombre de paquets d'entraînement	37 à 122 (dépend du SNR)
Taille des paquets de test	16384
Nombre de paquets de test	19 à 61 (dépend du SNR)

est entraîné sur les échecs du précédent D_{j-1}. Enfin, nous avons entraîné un BP-RNN seul en suivant la procédure de [10], pour avoir une référence de comparaison.

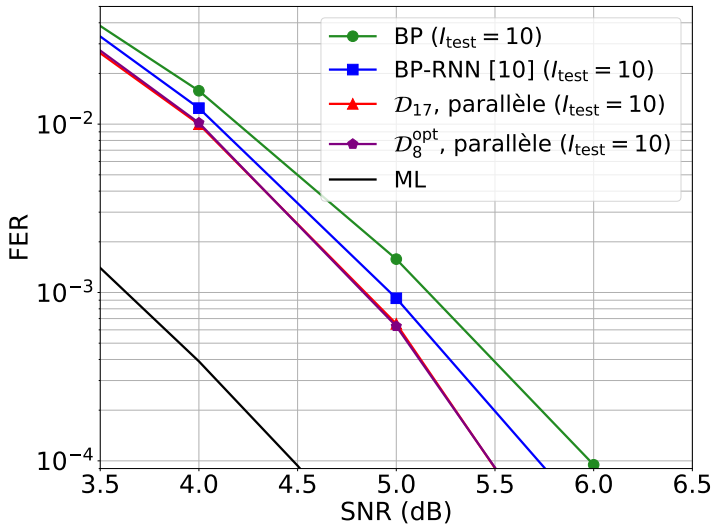
Pour entraîner les décodeurs BP-RNNs, nous avons utilisé la librairie Keras, avec les hyper-paramètres montrés dans le tableau 2.4. Pour le Code-1, tous les BP-RNNs sont entraînés pour chaque valeur de SNR comprise entre 1 dB et 7 dB, avec un pas de 1 dB, fournissant ainsi 7 ensembles de poids optimisés pour chaque décodeur. Pour le Code-3 et le Code-4, deux points supplémentaires à 8 dB et 9 dB sont considérés pour l'entraînement. Bien que cela augmente la complexité d'entraînement, la performance de décodage se retrouve également améliorée, par rapport au cas où uniquement un seul entraînement est effectué en mélangeant toutes les valeurs de SNRs [10].

Enfin, nous considérons un budget de $Z = 8$ maximum de huit BP-RNNs spécialisés et exécutés suivant une architecture parallèle quelque soit le code LDPC court. La sélection optimale $\mathcal{D}^{\text{opt}}_{Z=8}$ a été ainsi calculée pour chaque valeur de SNR de chaque code, en utilisant la méthode décrite en sous-section 2.2.4. Une nouvelle phase de test a été réalisées pour chaque code afin d'évaluer le taux d'erreur paquet pour uniquement huit décodeurs en parallèle.

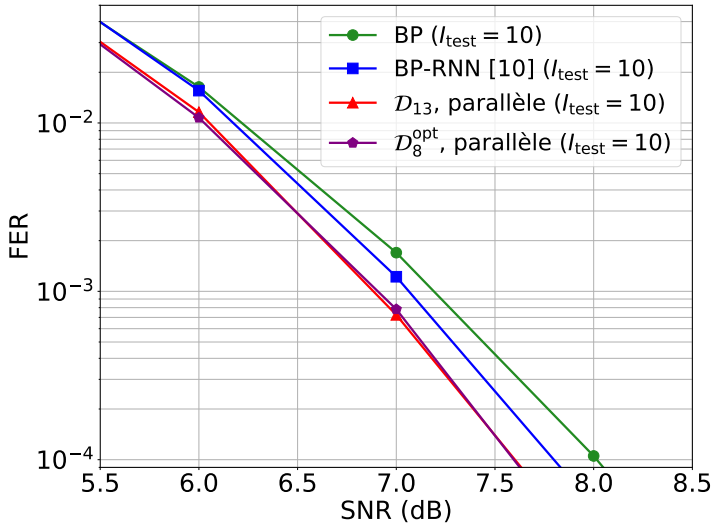
2.3.2 *BP-RNNs spécialisés*

Nous comparons ici la performance des différentes stratégies discutées dans la section précédente, en termes de FER. Les gains sont évalués à un FER de 10^{-4}.

Les résultats de simulation du Code-1 sont montrés sur la figure 2.3(a). En utilisant notre architecture parallèle de BP-RNNs spécialisés, le gain est accentué de 0.5 dB par rapport au BP($I_{\text{test}} = 10$). Le gain par rapport au BP-RNN de [10] est de 0.25 dB. De plus, il est important



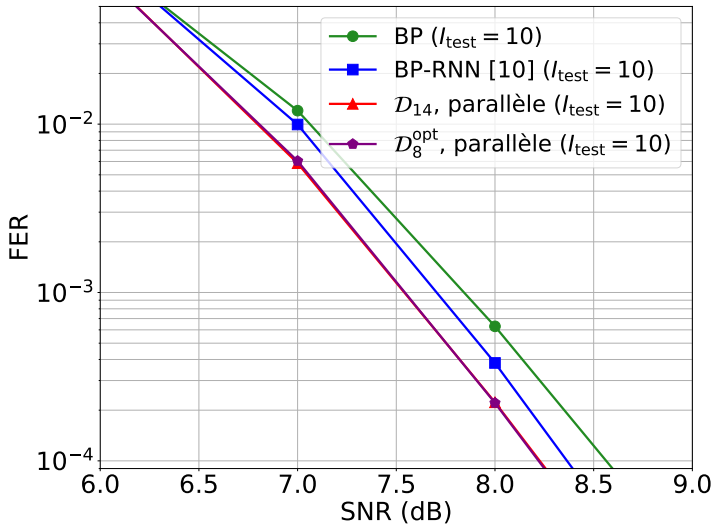
(a) FER du Code-1.



(b) FER du Code-3.

de remarquer que la sélection optimale $\mathcal{D}_8^{\text{opt}}$ permet d'atteindre les mêmes performances que l'architecture parallèle originale tout en réduisant d'un peu plus de la moitié la complexité calculatoire tant en termes du nombre moyen d'itérations effectuées à chaque décodage que du nombre d'itérations effectuées en cas d'échec de décodage. La complexité matérielle est également réduite d'un peu plus de la moitié. Enfin, notre nouvelle stratégie de décodage se situe à 1 dB de la performance ML.

Les résultats de simulation du Code-3 sont montrés dans la figure 2.3(b). Des gains similaires au code précédent sont observés. En effet, l'architecture proposée permet d'obtenir



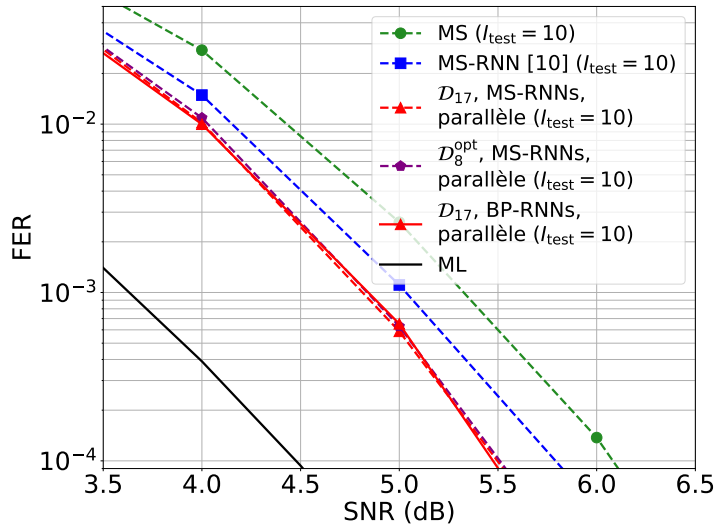
(c) FER du Code-4.

Figure 2.3 – Résultats FER des BP-RNNs spécialisés mis en parallèle.

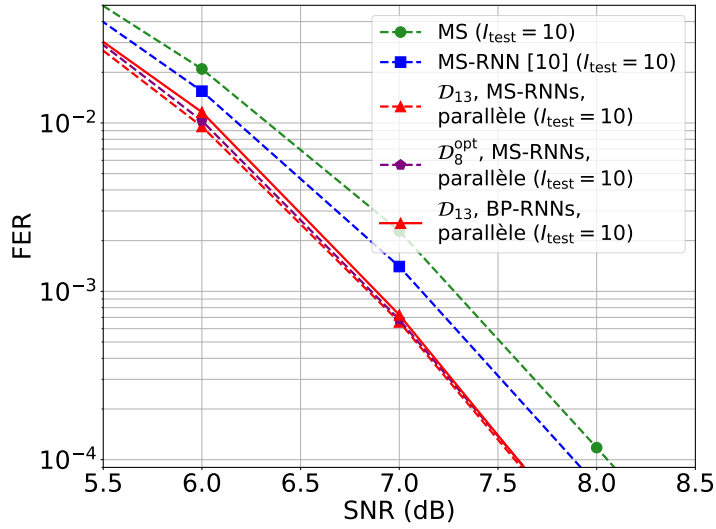
un gain de 0.42 dB par rapport au BP et de 0.22 dB par rapport au BP-RNN de [10]. Dans cet exemple aussi, les décodeurs de $\mathcal{D}_8^{\text{opt}}$ mis en parallèle obtiennent la même performance que J décodeurs mis en parallèle.

Enfin, les résultats de simulations du Code-4 sont affichés dans la figure 2.3(c). Malgré sa plus grande taille, ce code présente un nombre accru de cycles de longueur 4, en raison de son rendement de codage plus élevé. Cependant, les BP-RNNs spécialisés mis en parallèle fournissent aussi un gain similaire de 0.42 dB par rapport au BP et de 0.15 dB par rapport au BP-RNN de [10]. De plus, la figure 2.3c montre également la pertinence de la sélection optimale, permettant d'atteindre encore une fois les mêmes performances que les J décodeurs mis en parallèle.

Ainsi, pour les trois codes, les combinaisons de 8 décodeurs nous permettent de conserver des décodeurs se complétant les uns avec les autres, c'est à dire décodant des motifs d'erreurs différents. Notons également que chaque BP-RNN de $\mathcal{D}_8^{\text{opt}}$ utilise le data-pass défini en (1.27). Ils peuvent donc être implémentés avec des architectures matérielles conventionnelles du BP. Dans ce cas, le coût en mémoire supplémentaire pour un seul décodeur BP-RNN par rapport au décodeur BP provient uniquement du stockage des poids et le décodeur BP-RNN nécessite également une multiplication supplémentaire pour chaque poids dans (1.27) et (1.26). Enfin, comme le BP-RNN de [10] utilise la même configuration de poids que nos BP-RNNs spécialisés, $\mathcal{D}_8^{\text{opt}}$ nécessite un coût en mémoire et un nombre de multiplications 8 fois plus grand que ceux du BP-RNN de [10].



(a) FER du Code-1.



(b) FER du Code-3.

Figure 2.4 – Résultats FER des MS-RNNs spécialisés mis en parallèle.

2.3.3 MS-RNNs spécialisés

Dans cette sous-section, le BP-RNN est remplacé par le MS-RNN, le seul changement étant la substitution de la formule (1.22) par (1.11) pour le check-pass. Les décodeurs MS-RNNs sont entraînés sur les mêmes classes du tableau 2.3, en suivant la même procédure que les BP-RNNs. De plus, un MS-RNN est également entraîné en suivant la méthode de [10].

Les performances en termes de FER sont affichées sur la figure 2.4 pour le Code-1 (a) et le Code-3 (b) [3]. Pour le Code-1, nous observons tout d'abord un gain de 0.29 dB entre le MS et le MS-RNN entraîné comme dans [10]. Comparé au MS standard, notre architecture de MS-RNNs spécialisés obtient un gain de 0.58 dB, doublant ainsi le gain du MS-RNN [10]. Les $J = 17$ MS-RNNs spécialisés mis en parallèle atteignent la même performance que leurs homologues BP-RNNs spécialisés et mis en parallèle. Ceci donne un avantage en termes de complexité calculatoire aux MS-RNNs spécialisés, puisque que l'étape check-pass du MS-RNN est moins complexe que celle du BP-RNN. Cette conclusion reste valable pour le Code-3. Pour ce code, l'architecture parallèle de MS-RNNs spécialisés atteint un gain de 0.42 dB par rapport au MS standard et et 0.3 dB par rapport au MS-RNN de [10]. Enfin, comme pour les BP-RNNs spécialisés, la combinaison optimale de 8 MS-RNNs spécialisés permet d'atteindre la même performance que l'architecture parallèle des J MS-RNNs spécialisés pour les deux codes.

2.3.4 *Impact de la configuration de poids*

Nous proposons ici de comparer l'impact sur la performance de trois configurations différentes de poids du BP-RNN. La spécialisation sur les classes du tableau 2.3 est appliquée pour chacune des configurations. La configuration numéro 1 correspond à celle utilisée jusqu'à présent, donnée par les équations (1.27), (1.22) et (1.26). La seconde configuration de poids est celle introduite par les auteurs de [11]. En particulier, les poids dépendent des itérations de décodage. Cette configuration évaluent donc également si une dépendance des poids avec l'itération de décodage est bénéfique pour la performance. Des BP-FFs seront ainsi utilisés pour les simulations, la technique de spécialisation restant cependant la même qu'avec le BP-RNN. Les équations ci-dessous explicitent comment la pondération s'effectue :

$$\beta_{m\to n}^{(i)} = 2w_{m\to n}^{(i)} \tanh^{-1}\left(\prod_{n'\in\mathcal{N}(m)\setminus\{n\}} \tanh\left(\frac{\alpha_{n'\to m}^{(i-1)}}{2}\right)\right) \tag{2.8}$$

$$\alpha_{n\to m}^{(i)} = w_{n\to m}^{(i)}\left[w_{n,\mathrm{ch}}^{(i)} L_{\mathrm{ch},n} + \sum_{m'\in\mathcal{M}(n)\setminus\{m\}} \beta_{m'\to n}^{(i)}\right] \tag{2.9}$$

$$\tilde{L}_n^{(i)} = \tilde{w}_{n,\mathrm{ch}}^{(i)} L_{\mathrm{ch},n} + \sum_{m\in\mathcal{M}(n)} \beta_{m\to n}^{(i)} \tag{2.10}$$

La coloration des poids indiquent les différences avec la configuration 1. Notons que des poids sont introduits durant l'étape check-pass, comme l'illustre l'équation (2.8). Les LLRs

3. Les conclusions établies pour le Code-4 sont similaires.

canaux sont également pondérés dans les équations (2.9) et (2.10). De plus, un poids $w_{n\to m}^{(i)}$ est appliqué sur l'ensemble des messages provenant des noeuds de parité et sur le LLR canal dans (2.10). Cette configuration de poids est assez différente de la configuration 1 et est adaptée pour l'approche de [59], à savoir réaliser un l'élagage de noeuds de parité du BP-FF. Il est néanmoins intéressant de comparer ces deux configurations, afin de déterminer la meilleure dans notre cas d'utilisation.

Ensuite, nous considérons une troisième configuration de poids donnée par les équations suivantes :

$$\beta_{m\to n}^{(i)} = 2w_{m\to n} \tanh^{-1}\left(\prod_{n'\in\mathcal{N}(m)\setminus\{n\}} \tanh\left(\frac{\alpha_{n'\to m}^{(i-1)}}{2}\right)\right) \tag{2.11}$$

$$\alpha_{n\to m}^{(i)} = L_{\mathrm{ch},n} + \sum_{m'\in\mathcal{M}(n)\setminus\{m\}} w_{m'\to n\to m}\beta_{m'\to n}^{(i)} \tag{2.12}$$

$$\tilde{L}_n^{(i)} = L_{\mathrm{ch},n} + \sum_{m\in\mathcal{M}(n)} \tilde{w}_{m\to n}\beta_{m\to n}^{(i)} \tag{2.13}$$

Comparé à la configuration 1, des poids sont introduits dans l'étage check-pass, comme indiqué dans la formule (2.11). De plus, dans l'équation (2.12), un poids $w_{m'\to n\to m}$ est appliqué pour chaque message rentrant dans le neurone calculant $\alpha_{n\to m}$. La configuration 3 possède ainsi plus de poids que la configuration 1. La comparaison de ces deux configurations permet donc de savoir si l'introduction de davantage de poids entraînables induit une amélioration de la performance.

Les résultats en termes de FER des BP-RNNs spécialisés utilisant les différentes configurations de poids sont montrés dans la figure 2.5 pour le Code-3. Nous remarquons dans un premier temps que les BP-RNNs spécialisés utilisant la configuration de poids 1 ou 3 obtiennent les mêmes performances lorsqu'ils sont exécutés en parallèle. Ainsi, l'ajout de paramètres entraînables implique une complexité plus grande, sans pour autant fournir de gains. La configuration de poids 2 induit quant à elle une dégradation de 0.15 dB par rapport à la configuration de poids 1. En analysant les sorties des BP-RNNs spécialisés utilisant la configuration 2, ces derniers convergent généralement vers un mot de code, mais pas celui transmis. En conclusion, la configuration 1 nous permet d'obtenir le meilleur compromis performance/complexité. Les mêmes observations ont été faites pour les deux autres codes LDPC courts.

2.3.5 *BP-RNNs entraînés avec une fonction de coût multiple*

Comme expliqué dans le paragraphe 1.3.2.2, une fonction de coût multiple a été envisagée pour entraîner le BP neuronal dans [10] et [13]. Son intérêt est de prendre en compte

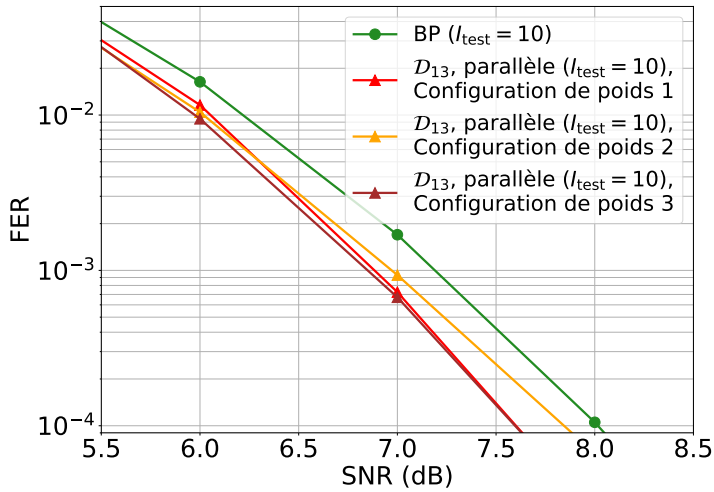


Figure 2.5 – Résultats FER du Code-3, pour différentes configurations de poids.

le comportement du décodeur neuronal au cours de ses itérations de décodage pour l'optimisation des poids. Il a ainsi été montré dans [10] qu'elle induit une meilleure performance du BP-RNN pour des codes BCH avec des matrices de parité de forte densité. Toutefois, pour des codes BCH courts avec une matrice de parité plus clairsemée (et possédant donc moins de cycles), les gains sont réduit significativement. Nous proposons de vérifier ceci avec nos codes LDPC courts construits par l'algorithme PEG et donc possédant un nombre significativement réduit de cycles courts. Une nouvelle spécialisation des BP-RNNs a ainsi été conduite, sur les mêmes classes, mais avec la fonction de coût suivante [10, 13] :

$$f_{\text{Multi-BCE}}(\tilde{L}) = -\frac{1}{I_{\text{ent}}} \sum_{i=1}^{I_{\text{ent}}} \frac{1}{N} \sum_{n=1}^{N} \log(\sigma(\tilde{L}_n^{(i)})), \tag{2.14}$$

Les résultats de ce test sont illustrés sur la figure 2.6 pour le Code-3. Nous constatons que la fonction de coût multiple n'engendre pas de meilleures performances par rapport à la fonction BCE standard. Ce test rejoint ainsi les conclusions sur le BP-RNN de [10] présentés dans le paragraphe 1.3.2.2. Dans la suite, nous n'utiliserons pas de fonction de coût multiple.

2.3.6 *Analyse des classes sélectionnées*

Nous proposons dans cette sous-section une analyse a posteriori des BP-RNNs sélectionnés dans $\mathcal{D}_Z^{\text{opt}}$ et de leurs classes ν-$(\omega, \varepsilon, \mathfrak{P})$ correspondantes. L'objectif de cette analyse est de déterminer les types de supports de motifs d'erreurs pour lesquels la spécialisation d'un BP-RNN apporte un gain en performance dans la diversité de décodage. Par souci de simplicité, nous considérons la diversité de décodage composée de 4 BP-RNNs spécialisés $\mathcal{D}_{Z=4}^{\text{opt}}$ du Code-1. La figure 2.7 affiche alors les sous-graphes induits des classes ν-$(\omega, \varepsilon, \mathfrak{P})$ sur

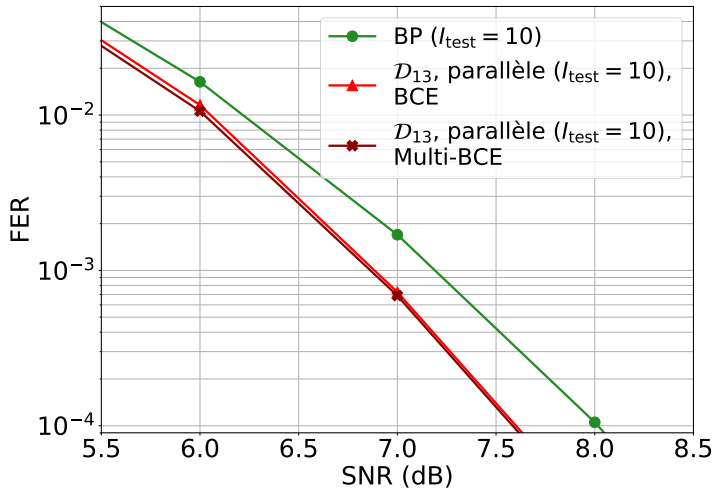
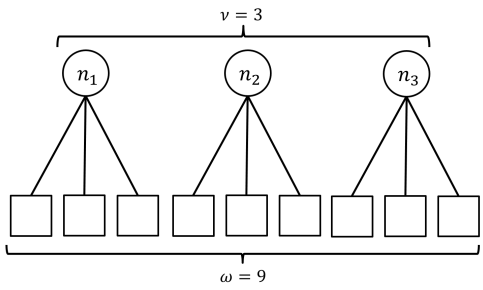


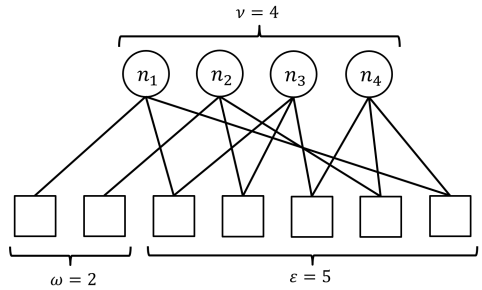
Figure 2.6 – Résultats FER du Code-3, BCE vs Multi-BCE.

lesquelles les BP-RNNs de $\mathcal{D}_4^{\text{opt}}$ ont été entraînés. Le cas (a) correspond à une classe de motifs d'erreurs relativement faciles pour le décodage pour le BP, chaque noeud de donnée étant connecté uniquement à des noeuds de parité de dégré impair. En revanche, dans les cas (b) et (c), chaque noeud de donnée est connecté à plus de noeuds de parité de degré pair que de degré impair. Par conséquent, les classes 4-$(4, 4, \mathfrak{P})$ et 4-$(2, 5, \mathfrak{P})$ contiennent uniquement des ensembles de noeuds données formant des ensembles absorbants. Enfin, le cas (d) n'est pas un ensemble absorbant, le noeud de donnée n_1 étant connecté à plus de noeud de parité de degré impair que de degré pair. Cependant, si n_1 est corrigé lors du décodage, le motif $\{n_2, n_3, n_4\}$ se restreint à un ensemble absorbant de la classe 3-$(3, 3, [(1, 2), (1, 2), (1, 2)])$. Les motifs de la classe 4-$(4, 3, \mathfrak{P})$ se rapprochent ainsi fortement de motifs avec un support d'ensemble absorbant. Nous constatons donc que trois des BP-RNNs de $\mathcal{D}_4^{\text{opt}}$ sont entraînés dans le décodage de motifs d'erreurs soit avec un support formant un ensemble absorbant, soit avec un support très proche d'un ensemble absorbant.

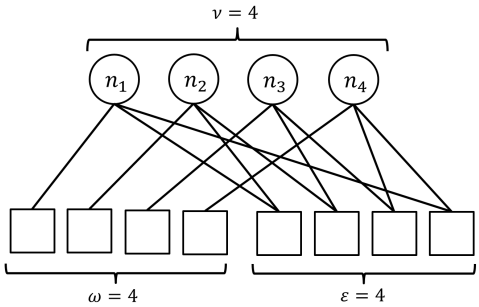
Ainsi, dans la figure 2.8, nous montrons l'importance des BP-RNNs entraînés sur des ensembles absorbants dans la performance observée. En effet, l'architecture parallèle de $\mathcal{D}_4^{\text{opt}}$ atteint quasiment la même performance que l'architecture parallèle de $\mathcal{D}_{17}$. De plus, le BP-RNN entraîné sur la classe 3-$(9, 0, \mathfrak{P})$ possède une performance similaire au BP-RNN de [10]. Ainsi, les gains observés par rapport à l'état de l'art résultent des BP-RNNs de $\mathcal{D}_4^{\text{opt}}$ spécialisés sur des ensembles soit absorbants (classes 4-$(2, 5, \mathfrak{P})$ et 4-$(4, 4, \mathfrak{P})$), soit très proches d'ensembles absorbants (classe 4-$(4, 3, \mathfrak{P})$). Enfin, des tests supplémentaires nous ont aussi permis d'identifier qu'une diversité de BP-RNNs entraînés uniquement sur les classes composées d'ensembles absorbants du Code-1 ($\nu \leq 4$) présente une performance proche de celle de $\mathcal{D}_{17}$. Des conclusions similaires peuvent être établies pour le Code-3 et le Code-4.



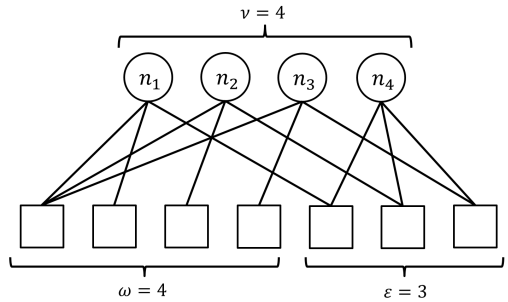
(a) Sous-graphe de la classe 3-$(9, 0, \mathfrak{P})$.



(b) Sous-graphe de la classe 4-$(2, 5, \mathfrak{P})$.



(c) Sous-graphe de la classe 4-$(4, 4, \mathfrak{P})$.



(d) Sous-graphe de la classe 4-$(4, 3, \mathfrak{P})$.

Figure 2.7 – Les sous-graphes des classes correspondants aux BP-RNNs de $\mathcal{D}_4^{\text{opt}}$ pour le Code-1. Les types absorbants de chaque classe étant différents, les profils de connexions $\mathfrak{P}$ ne sont pas détaillés pour simplifier la notation.

2.4 Conclusion

Ce chapitre s'est concentré sur le développement d'une diversité de décodage composée de plusieurs BP-RNNs, où chacun est spécialisé dans le décodage de petits motifs d'erreurs partageant des propriétés structurelles similaires. Nous avons tout d'abord montré que le jeu d'entraînement standard aléatoire ne représente pas correctement les motifs d'erreurs responsables des échecs dans la région du *waterfall*, empêchant par conséquent le BP-RNN d'apprendre à les traiter. De plus, nous avons mis en évidence que ce jeu standard possède une trop grande variété de motifs d'erreurs en termes de sous-graphe induit. Les caractéristiques propres à chaque type de sous-graphe induit ne sont ainsi pas apprises de manière adéquate par le BP-RNN.

Afin de remédier à ces problèmes, nous avons proposé dans un premier temps une nouvelle méthode de construction du jeu d'entraînement. Elle se base sur une classification exhaustive des supports de motifs d'erreurs selon leurs types absorbants et les profils de connexion de leurs noeuds de donnée. Ensuite, pour chaque classe déterminée, un jeu

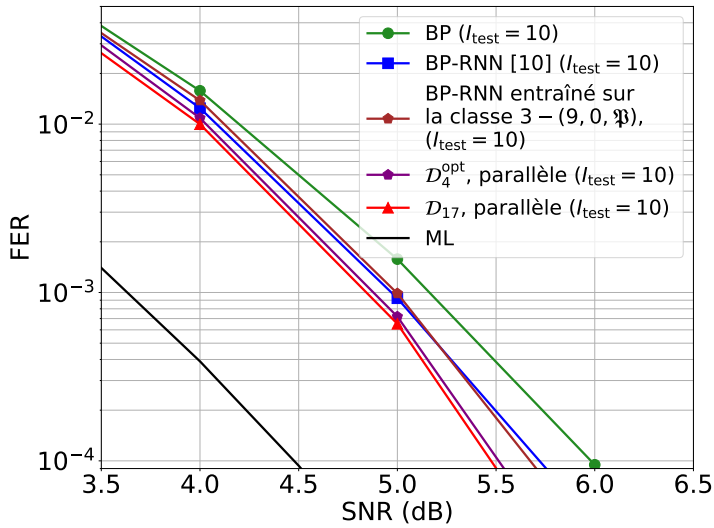


Figure 2.8 – Résultats FER du Code-1.

d'entraînement spécifique a été créé. Il contient des mots bruités tels que le motif d'erreurs appartienne à la classe correspondante. Les erreurs y sont notamment placées aux positions voulues grâce à une distribution gaussienne tronquée. Le jeu d'entraînement résultant permet ainsi de spécialiser le BP-RNN dans le décodage des motifs de la classe relative.

Dans un second temps, la procédure de spécialisation a été associée à la notion de diversité de décodage. En effet, pour chaque classe déterminée, un BP-RNN est entraîné spécifiquement pour décoder les motifs d'erreurs avec un support appartenant à la classe. Les BP-RNNs spécialisés de la diversité de décodage sont alors utilisés dans une architecture de décodage parallèle. Les résultats en termes de FER ont alors montré des performances améliorées par rapport à l'état de l'art. De plus, nous avons proposé une méthode sélectionnant les BP-RNNs les plus complémentaires de manière optimale, qui s'est avérée efficace pour réduire le nombre de décodeurs mis en parallèle sans perte de performance. Des tests supplémentaires ont aussi permis de confirmer le choix de la configuration de poids du BP-RNN et de la fonction de coût. Enfin, nous avons montré que les BP-RNNs spécialisés sur les ensembles absorbants sont en réalité responsables des gains observés par rapport au BP.

Néanmoins, à cause du caractère exhaustif de la classification, l'approche présentée dans ce chapitre reste limité pour des codes LDPC courts corrigeant peu d'erreurs. Le prochain chapitre se consacre en conséquence à adapter cette approche pour des codes LDPC courts avec une meilleure capacité de correction (codes plus long et/ou de rendement plus faible), en se concentrant notamment sur les motifs d'erreurs avec un ensemble absorbant comme support.

Cette première contribution a donné lieu a une publication dans la conférence International Symposium on Topics in Coding de 2021 [IC1].

Chapitre 3

Diversité de BP-RNNs spécialisés sur des ensembles absorbants

Contenu du Chapitre 3

3.1 Les ensembles absorbants 58
3.2 Diversité de BP-RNNs spécialisés sur des ensembles absorbants 59
3.2.1 Algorithme de recherche des ensembles absorbants 59
3.2.2 Classification des ensembles absorbants 63
3.2.3 Jeu d'entraînement spécialisé 64
3.3 Sélection des BP-RNNs et architecture de décodage 66
3.3.1 Sélection sous-optimale d'une diversité de BP-RNNs 66
3.3.2 Architecture série d'une diversité de BP-RNNs 68
3.3.3 Métriques de latence et de complexité 68
3.4 Résultats numériques 71
3.4.1 Paramètres de simulation 71
3.4.2 Nombre maximal d'itérations pour l'entraînement et le test 72
3.4.3 Impact du SNR d'entraînement 74
3.4.4 Sélection d'une diversité de BP-RNNs 75
3.4.5 Performances FER 77
3.4.6 Évaluation de la complexité et de la latence 81
3.5 Conclusion 84

Dans le chapitre précédent, plusieurs algorithmes par propagation de croyances (BP) modélisés par des réseaux de neurones récurrents (BP-RNNs) ont été entraînés en énumérant exhaustivement les supports de motifs d'erreurs de taille inférieure ou égale à 4, puis en les classant selon la structure du sous-graphe induit. L'architecture parallèle d'une diversité de décodage composée de ces BP-RNNs spécialisés a ensuite montré de meilleures performances par rapport à l'utilisation d'un seul décodeur BP-RNN entraîné comme dans [10], pour des codes LDPC courts avec une faible capacité de correction.

Dans ce chapitre, nous nous posons la question d'améliorer les performances pour des codes ayant une meilleure capacité de correction. Pour cela, nous allons explorer des structures spécifiques qui apparaissent dans les graphes, les ensembles absorbants. Ainsi, nous proposons dans un premier temps un algorithme énumérant les ensembles absorbants et une classification selon leurs propriétés structurelles. En suivant une méthode similaire au chapitre 2, les BP-RNNs sont ensuite spécialisés sur le décodage de motifs d'erreurs avec un support d'ensemble absorbant, grâce à des jeux d'entraînement spécifiques. Pour réduire le nombre de décodeurs BP-RNNs, nous proposons également une nouvelle procédure de sélection moins complexe que celle présentée en sous-section 2.2.4. Nous considérons ensuite deux architectures de décodage, dans lesquelles les BP-RNNs sélectionnés sont exécutés soit en parallèle soit en série, puis nous définissons les métriques appropriées pour évaluer leur complexité calculatoire ainsi que la latence de décodage associée.

Enfin, il convient de mentionner que deux autres approches de spécialisation ont été développées en parallèle de celle sur les ensembles absorbants, afin de traiter également des codes LDPC courts avec une meilleure capacité de correction. Elles sont présentées et comparées avec la méthode de spécialisation sur les ensembles absorbants dans l'annexe B.

3.1 Les ensembles absorbants

Comme décrit en sous-section 1.1.4, l'une des difficultés majeurs rencontrée par le BP est la présence de structures particulières dans le graphe, tels que les ensembles absorbants [43]. Ces derniers sont définis de manière indépendante du décodeur et du modèle de bruit. De ce fait, une spécialisation sur des ensembles absorbants peut aussi bien être réalisée pour des BP-RNNs, que pour des Min-Sum modélisés par des RNNs (MS-RNNs) par exemple. De plus, pour des petits mots de code, les ensembles absorbants sont de taille équivalente au nombre d'erreurs qu'un code peut corriger et sont responsables de dégradation significative dans la région du *waterfall*. Entraîner des réseaux de neurones à mieux les décoder permettrait donc de diminuer ces dégradations.

Une seconde motivation à explorer les ensembles absorbants provient de l'étude des décodeurs BP-RNNs sélectionnés dans le chapitre précédent. En effet, nous avons montré dans la sous-section 2.3.6 que les BP-RNNs sélectionnés sont en réalité majoritairement entraînés sur des classes contenant des supports soit d'ensembles absorbants, soit très proches d'ensembles absorbants. De plus, nous avons prouvé que les BP-RNNs spécialisés sur les ensembles absorbants sont responsables des gains engendrés par rapport au BP ou au BP-

RNN entraîné comme dans [10]. Il n'est donc pas nécessaire d'énumérer exhaustivement tous les supports de motifs d'erreurs pour créer une diversité de BP-RNNs, une spécialisation de l'entraînement sur les ensembles absorbants suffit.

Nous en déduisons l'intérêt de spécialiser l'entraînement sur des ensembles absorbants afin de traiter des codes LDPC courts avec une meilleure capacité de correction que ceux du chapitre 2. Cette approche est présentée dans la section suivante.

3.2 Diversité de BP-RNNs spécialisés sur des ensembles absorbants

La question que nous abordons maintenant est le choix des ensembles absorbants pour créer une diversité de BP-RNNs spécialisés dans le décodage des motifs d'erreurs.

Notre approche se divise en trois étapes. Nous proposons tout d'abord un algorithme pour énumérer de manière efficace tous les ensembles absorbants d'une taille donnée dans le graphe biparti. Il se base notamment sur une procédure de parcours en profondeur (Depth-First Search ou DFS, en anglais). Ensuite, une classification est réalisée sur les ensembles absorbants identifiés, selon le profil de degrés des noeuds de parité du sous-graphe induit. Enfin, de manière similaire au chapitre précédent, un BP-RNN est entraîné spécifiquement pour chaque classe d'ensembles absorbants.

3.2.1 Algorithme de recherche des ensembles absorbants

Un algorithme de recherche exhaustive énumérant tous les ensembles absorbants (ou même les ensembles piégeants) d'une taille donnée ν devrait explorer tous les $\binom{N}{\nu}$ candidats, ce qui peut devenir très vite trop complexe, même pour des valeurs relativement petites de N et ν. Plusieurs algorithmes ont ainsi été proposés dans la littérature pour rechercher différents types d'ensembles absorbants ou piégeants. Les ensembles piégeants élémentaires, dont les sous-graphes ne contiennent uniquement que des noeuds de parité de degré un ou deux, sont énumérés avec les procédures de [60–62]. Dans [63, 64], les ensembles piégeants et les ensembles absorbants sont listés pour des classes spécifiques de codes LDPC. Des travaux supplémentaires se concentrent quant à eux sur des ensembles piégeants dominants [65], responsables de la majorité des échecs dans la région du plancher d'erreur, ou sur des ensembles piégeants de petite taille [66]. Enfin, des ensembles entièrement absorbants (c.-à-d., lorsque la condition d'ensemble absorbant est satisfaite par tous les nœuds de donnée dans le graphe) sont énumérés dans [67, 68]. Plusieurs de ces procédures d'énumération [62, 67, 68] reposent notamment sur une approche de programmation linéaire basée sur de la séparation-et-évaluation.

Notons également que l'algorithme proposé par [64] exploite les propriétés liées au type spécifique de codes LDPC traités pour énumérer les ensembles absorbants. De plus, les approches décrites dans [67, 68] utilisent les contraintes induites par les ensembles entièrement absorbants pour les rechercher. Les méthodes de l'état de l'art ne permettent donc pas de lister sans contrainte tous les ensembles absorbants d'un code LDPC court quelconque.

Ici, nous proposons un algorithme basé sur de la recherche de graphes, qui est à notre connaissance le premier spécifiquement développé pour une énumération exhaustive des ensembles absorbants, sans aucune contrainte sur la structure du graphe de Tanner de ces ensembles ou sur le type de codes LDPC considéré. L'algorithme proposé est essentiellement un algorithme de retour-arrière, construisant progressivement des ensembles absorbants candidats et abandonnant un candidat dès qu'il détermine que ce candidat ne peut pas être complété en un ensemble absorbant. Les candidats sont construits de manière incrémentale en traversant le graphe biparti avec une approche de parcours en profondeur (Depth First Search ou DFS en anglais) [69]. En effet, en partant d'un noeud de donnée choisi comme racine, nous explorons au plus profondément le graphe avant de faire un retour-arrière. La différence majeure avec le DFS standard de [69] est que notre algorithme ne visite pas seulement un, mais un sous-ensemble de nœuds de donnée à chaque niveau de profondeur pour incrémenter la solution candidate.

Nous proposons l'algorithme de recherche d'ensembles absorbants basé sur du parcours en profondeur (Absorbing Sets-Depth First Search ou AS-DFS en anglais) qui suit le déroulé suivant.

Soit $n \in \{1, \ldots, N\}$ un noeud de donnée fixé, que nous appellerons le noeud racine. Notre algorithme énumère tous les ensembles absorbants $\mathcal{A} \subset \{1, \ldots, N\}$ contenant n, de taille $|\mathcal{A}| = v$. Pour éviter d'énumérer plusieurs fois le même ensemble absorbant en partant de différents noeuds racines, nous imposons de plus que n soit le plus petit élément de $\mathcal{A}$. Autrement dit, $\mathcal{A} \subset \{n, \ldots, N\}$.

Le graphe biparti partant du noeud racine n est dans un premier temps étendu, c'est-à-dire, les noeuds de parité et les noeuds de donnée sont ajoutés progressivement à une distance croissante de n jusqu'à ce que le noeud le plus éloigné dans le graphe biparti soit atteint. Comme pour l'algorithme par croissance d'arêtes progressives (PEG), la distance entre deux noeuds est la longueur (*e.g*, le nombre d'arêtes) du plus court chemin les connectant. Cette expansion produit un graphe biparti enraciné, noté par $\mathcal{G}_n$. Pour $\ell \geq 0$, nous notons par $\mathcal{V}^{(\ell)}$ l'ensemble des noeuds de donnée à une distance 2ℓ de n (donc $\mathcal{V}^{(0)} = \{n\}$) et par $C^{(\ell)}$ l'ensemble des noeuds de parité à une distance $2\ell + 1$ de n.

Soit $\mathcal{A}$, un ensemble quelconque contenant n et $C_{\mathcal{A}}$ l'ensemble des noeuds de parité connectés à $\mathcal{A}$ au moins une fois. Pour $\ell \geq 0$, nous définissons $\mathcal{A}^{(\ell)} := \mathcal{A} \cap \mathcal{V}^{(\ell)}$ et $C_{\mathcal{A}}^{(\ell)} := C_{\mathcal{A}} \cap C^{(\ell)}$. Si chaque noeud de donnée de $\mathcal{A}^{(\ell)}$ possède plus de voisins dans $\mathcal{E}(\mathcal{A})$ (pair) que dans $\mathcal{O}(\mathcal{A})$ (impair), alors $\mathcal{A}^{(\ell)}$ satisfait la condition d'ensemble absorbant. Puisque $\mathcal{A}^{(\ell)}$ est un sous-ensemble de $\mathcal{V}^{(\ell)}$, les voisins de ses noeuds de donnée sont des noeuds de parité appartenant soit à $C_{\mathcal{A}}^{(\ell-1)}$, soit à $C_{\mathcal{A}}^{(\ell)}$. Par conséquent, les degrés de ces noeuds de parité dans le sous-graphe induit par $\mathcal{A}$ dépendent uniquement de $\mathcal{A}^{(\ell-1)}$ (pour $\ell > 0$), de $\mathcal{A}^{(\ell)}$ et de $\mathcal{A}^{(\ell+1)}$ (pour ℓ inférieur à la profondeur maximale de $\mathcal{A}$). Ainsi, la condition d'ensemble absorbant pour $\mathcal{A}$ peut se vérifier de manière incrémentale en la vérifiant pour chaque sous-ensemble $\mathcal{A}^{(\ell)}$, ce qui nécessite toutefois la connaissance du sous-ensemble de niveau suivant $\mathcal{A}^{(\ell+1)}$ (sauf pour le dernier niveau).

L'algorithme proposé construit alors progressivement un ensemble absorbant $\mathcal{A}$, en

Algorithme 2 AS-DFS $\left(\mathcal{G}_n, \ell, \mathcal{A}^{(0)}, \ldots, \mathcal{A}^{(\ell)}\right)$

si $\ell > 0$ et AS_check $\left(\mathcal{A}^{(\ell-1)}\right)$ = faux **alors**
  retourner // retour-arrière
fin si
$\bar{v}_\ell \leftarrow \left|\mathcal{A}^{(0)}\right| + \cdots + \left|\mathcal{A}^{(\ell)}\right|$
si $\bar{v}_\ell = v$ **alors**
  si AS_check $\left(\mathcal{A}^{(\ell)}\right)$ = vrai **alors**
    Ajouter $\mathcal{A} := \mathcal{A}^{(0)} \cup \cdots \cup \mathcal{A}^{(\ell)}$ à la liste des ensembles absorbants
  fin si
  retourner // retour-arrière
fin si
$\mathfrak{C}\left(\mathcal{A}^{(0)}, \ldots, \mathcal{A}^{(\ell)}\right) \leftarrow$ Ensemble des complétions possibles
pour tout les $\mathcal{A}^{(\ell+1)}$ **dans** $\mathfrak{C}(A_0, \ldots, A_\ell)$ **faire**
  AS_DFS $\left(\mathcal{G}_n, \ell+1, \mathcal{A}^{(0)}, \ldots, \mathcal{A}^{(\ell)}, \mathcal{A}^{(\ell+1)}\right)$ // appel récursif
fin pour

choisissant des sous-ensembles $\mathcal{A}^{(\ell)}$, pour $\ell \geq 0$ et en vérifiant la condition d'ensemble absorbant pour déterminer si le choix courant pourrait éventuellement être complété en un ensemble absorbant valide. Pour $\ell = 0$, nous avons $\mathcal{A}^{(0)} = \{n\}$. Considérons un choix de sous-ensembles $\mathcal{A}^{(0)}, \ldots, \mathcal{A}^{(\ell)}$, avec $\ell \geq 0$, tel que $\bar{v}_\ell := \sum_{t=0}^{\ell} \left|\mathcal{A}^{(t)}\right| \leq v$. Nous définissons alors l'ensemble des complétions possibles $\mathfrak{C}\left(\mathcal{A}^{(0)}, \ldots, \mathcal{A}^{(\ell)}\right)$ comme suit :

- Si $\bar{v}_\ell = v$, alors $\mathfrak{C}\left(\mathcal{A}^{(0)}, \ldots, \mathcal{A}^{(\ell)}\right) = \emptyset$ (la taille des ensembles absorbants recherchés v est atteinte, il n'y a donc pas besoin de complétion).
- Si $\bar{v}_\ell < v$, alors
  (1) Nous déterminons le sous-ensemble $\mathcal{N}^{(\ell+1)} \subset \mathcal{V}^{(\ell+1)}$ contenant les descendants des noeuds de donnée dans $\mathcal{A}^{(\ell)}$.
  (2) Nous déterminons $\mathcal{N}_{\geq n}^{(\ell+1)} := \mathcal{N}^{(\ell+1)} \cap \{n, \ldots, N\}$, puis nous complétons

$$\mathfrak{C}\left(\mathcal{A}^{(0)}, \ldots, \mathcal{A}^{(\ell)}\right) := \left\{\mathcal{A}^{(\ell+1)} \subseteq \mathcal{N}_{\geq n}^{(\ell+1)} \;\middle|\; \left|\mathcal{A}^{(\ell+1)}\right| \leq v - \bar{v}_\ell\right\}. \tag{3.1}$$

Il est à noter que $\mathfrak{C}\left(\mathcal{A}^{(0)}, \ldots, \mathcal{A}^{(\ell)}\right)$ est un ensemble contenant des sous-ensembles $\mathcal{A}^{(\ell+1)}$ [1], pouvant être parcourus dans n'importe quel ordre. En pratique, ce parcours itératif s'effectue de manière croissante avec la taille des sous-ensembles $\mathcal{A}^{(\ell+1)}$.

L'algorithme AS-DFS est présentée dans l'algorithme 2. En considérant un noeud de donnée racine n, la procédure est simplement appelée par le programme principale avec les arguments $\left(\mathcal{G}_e, \ell = 0, \mathcal{A}^{(0)} = \{n\}\right)$.

Le déroulement de l'algorithme AS-DFS est illustré par un exemple avec le graphe biparti enraciné de la figure 3.1, avec le noeud de donnée n_1 comme noeud racine et $\{n_2, n_3, n_4, n_5, n_6\} \subset$

1. Ces sous ensembles peuvent être vide si $\bar{v}_\ell = v$, ou si $\mathcal{N}_{\geq n}^{(\ell+1)} = \emptyset$.

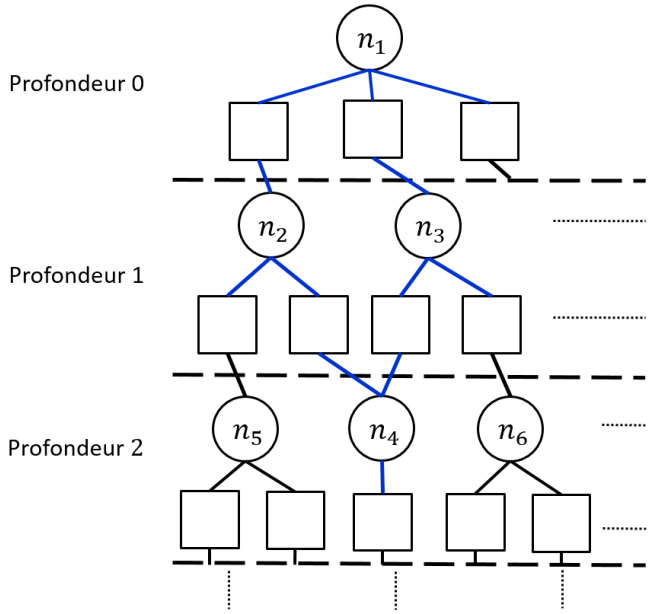
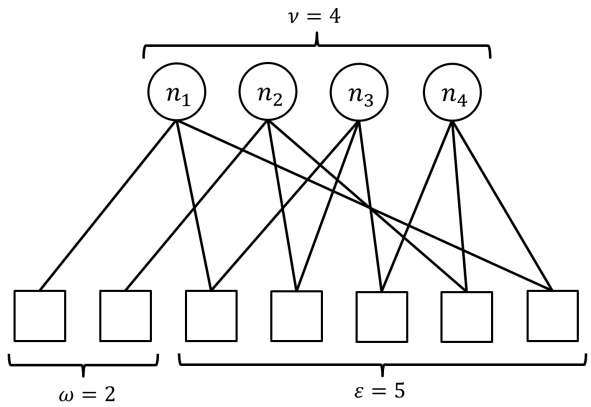


Figure 3.1 – Exemple de graphe biparti enraciné $\mathcal{G}_n$. Les connexions prises en comptes pour le choix $\mathcal{A} = \{n_1, n_2, n_3, n_4\}$ sont indiquées en bleu.

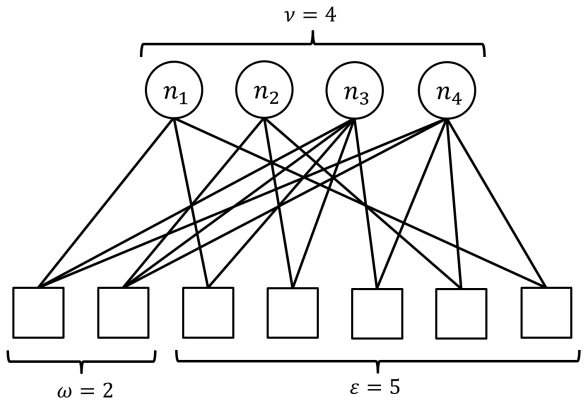
$\{n_1 + 1, \ldots, N\}$. La taille v est fixée à 4 et les noeuds de donnée sont de degré $d_n = 3$. Les complétions sont tout d'abord déterminées pour $l = 0$ et $\bar{v}_\ell = 1$. Examinons le choix $\mathcal{A}^{(1)} = \{n_2, n_3\}$ et l'appel récursif correspondant AS-DFS $\left(\mathcal{G}_n, \ell = 1, \mathcal{A}^{(0)}, \mathcal{A}^{(1)}\right)$. La condition d'ensemble absorbant est vérifiée pour $\mathcal{A}^{(0)}$, n_1 étant connecté à deux noeuds de parité de degré pair et un noeud de degré impair dans le sous-graphe induit par $\mathcal{A} = \mathcal{A}^{(0)} \cup \mathcal{A}^{(1)}$. De ce fait, $\bar{v}_\ell$ est mis à jour à 3, puis les complétions de taille $v - \bar{v}_\ell = 1$ sont calculées. Si la procédure est appliquée pour $\ell = 2$ et la complétion $\mathcal{A}^{(2)} = \{n_3\}$, la condition d'ensemble absorbant s'avère vraie pour $\mathcal{A}^{(1)}$. En effet, n_2 et n_3 sont chacun connectés à deux noeuds de parité de degré pair et un noeud de parité de degré impair dans le sous-graphe induit par $\mathcal{A} = \{n_1, n_2, n_3, n_4\}$. La variable $\bar{v}_\ell$ est ensuite actualisée à 4 et le test $\bar{v}_\ell = v$ devient vrai. Il est alors vérifié que $\mathcal{A} = \{n_1, n_2, n_3, n_4\}$ est bien un ensemble absorbant de taille 4, de sous-graphe induit par ailleurs isomorphe à celui de la figure 2.7c. Par contre, si $\mathcal{A}^{(2)} = \{n_5\}$, la condition d'ensemble absorbant n'est plus vérifiée pour $\mathcal{A}^{(1)}$, n_3 étant maintenant connecté à deux noeuds de parité de degré impair contre un seul de degré pair dans le sous graphe induit correspondant. Le choix de sous-ensembles $\mathcal{A}^{(0)}, \mathcal{A}^{(1)}, \mathcal{A}^{(2)} = \{n_5\}$ est donc abandonné. Il en est de même pour le choix $\mathcal{A}^{(0)}, \mathcal{A}^{(1)}, \mathcal{A}^{(2)} = \{n_6\}$. À ce stade, les complétions de 1 noeud de donnée de $\{n_1, n_2, n_3\}$ sont toutes explorées. Le retour-arrière s'effectue donc et d'autres ensembles $\mathcal{A}^{(1)}$ sont considérés pour compléter $\mathcal{A}^{(0)} = \{n_1\}$.

3.2.2 Classification des ensembles absorbants

Une fois les ensembles absorbants identifiés et dénombrés, nous voulons les classifier selon leurs propriétés structurelles dans le sous-graphe induit. Une limitation pratique de la classification par type absorbant proposée en sous-section 2.2.1 est sa complexité calculatoire. En effet, pour chaque ensemble de noeuds de donnée de taille ν, la classification par type absorbant et par liste des profils de connexions $\mathfrak{B}$ requiert 2^ν permutations et 2^ν comparaisons. Cette approche peut ainsi vite engendrer un nombre important de calculs lorsque ν augmente. Par exemple, pour le Code-1 $(N = 64, K = 32)$, notre algorithme de recherche énumèrent 450340 ensembles absorbants de taille $\nu = 7$. Un total d'environ $450340 \times 2^7 \times 2 \approx 115$ millions d'opérations est donc nécessaire pour classifier ces ensembles absorbants avec la procédure du chapitre 2.



(a) $P_c = (2, 5)$.



(b) $P_c = (0, 5, 2)$.

Figure 3.2 – Exemple d'ensembles absorbants de type 4-$(2, 5)$.

Tableau. 3.1 – Nombre de types étendus (TE) et d'ensembles absorbants (EA).

	Code-1		Code-2	
ν	Nombre de TE	Nombre d'EA	Nombre de TE	Nombre d'EA
3	1	164	1	32
4	2	1 452	6	944
5	3	9 413	12	11 504
6	9	64 813*	32	152 824
7	16	450 340	69	2 124 928
8	24	2 994 834*	157	28 670 736

*Le Code-1 contient un mot de code de poids 6 et 37 mots de code de poids 8, correspondant à des ensembles absorbants avec $\omega = 0$

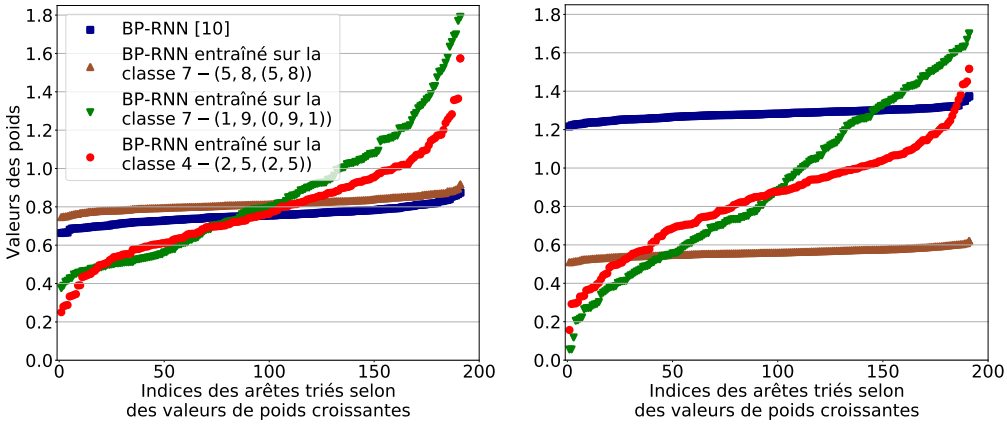
Pour limiter la complexité, nous caractérisons plus simplement un ensemble absorbant $\mathcal{A}$ en définissant son profil de connexion des noeuds de parité $P_c := (m_1, m_2, \dots)$, où m_d est le nombre de noeuds de parité connectés à exactement d noeuds de donnée dans $\mathcal{A}$. Nous introduisons alors le type étendu de $\mathcal{A}$ comme ν-$(\omega, \varepsilon, P_c)$, avec $\nu = \mathbf{card}(\mathcal{V})$, $\omega = \mathbf{card}(\mathcal{O}(\mathcal{V}))$, $\varepsilon = \mathbf{card}(\mathcal{E}(\mathcal{V}))$ et P_c le profil de connexions de noeuds de parité. Cette classification par type étendu permet de séparer finement des ensembles absorbants possédant des sous-graphes induits différents. Un exemple est donné en figure 3.2 pour deux types absorbants 4-$(2, 5)$, où les noeuds de donnée sont respectivement de degré 3 dans le cas (a) et de degré 3 et 5 dans le cas (b).

Pour illustrer plus en détail la capacité de notre algorithme AS-DFS de recherche d'ensembles absorbants, ainsi que la classification par type étendu, nous considérons le Code-1 et le Code-2 de paramètres $(N = 128, K = 64)$, explicités en sous-section 1.1.5. Nous fournissons dans le tableau 3.1 le nombre total de types étendus (TE) différents et le nombre total d'ensembles absorbants pour ces deux codes, avec une taille d'ensemble absorbant $\nu \leq 8$. Précisons que pour le Code-2 et $\nu = 8$, un algorithme de recherche par force brute devrait explorer $\binom{128}{8} \approx 2^{40}$ candidats.

3.2.3 *Jeu d'entraînement spécialisé*

Nous voulons maintenant spécialiser l'entraînement sur les classes d'ensembles absorbants déterminées précédemment. Nous proposons de réutiliser les mêmes conditions que la sous-section 2.2.2 pour la génération du bruit, à savoir le canal AWGN à entrée binaire de variance σ^2 et de moyenne nulle et la transmission du mot de code tout zéro uniquement [2]. L'objectif est alors de construire un jeu d'entraînement spécifique, contenant uniquement des mots reçus $\mathbf{y}$ tels que l'ensemble d'erreurs $\mathcal{E}(\mathbf{y})$ correspond à un motif d'erreurs avec un support d'ensemble absorbant d'une classe ν-$(\omega, \varepsilon, P_c)$ donnée. Pour réaliser ceci, un ensemble $\mathcal{A}$ parmi la classe ν-$(\omega, \varepsilon, P_c)$ traitée est choisi aléatoirement, puis la distribution

2. D'autres mots de code peuvent être considérés pour l'entraînement, à condition d'utiliser l'entropie croisée binaire non simplifiée de l'équation (1.29).



(a) Étage data-pass.

(b) Étage a posteriori.

Figure 3.3 – Profils de poids pour différents décodeurs BP-RNNs (Code-1, SNR = 5 dB).

normale tronquée est utilisée de nouveau comme détaillé dans l'équation (2.4).

Le jeu d'entraînement est alors obtenu en répétant cette procédure jusqu'à atteindre la taille souhaitée. Ainsi, le jeu d'entraînement est représentatif de la classe d'ensembles absorbants ν-$(\omega, \varepsilon, P_c)$ considérée. Le BP-RNN entraîné avec ce jeu sera spécialisé en conséquence dans le décodage des motifs d'erreurs avec un support d'ensemble absorbant appartenant à la classe ν-$(\omega, \varepsilon, P_c)$.

Pour illustrer la spécialisation du décodeur entraîné, la figure 3.3 montre les valeurs des poids pour trois décodeurs BP-RNNs, entraînés chacun sur une classe ν-$(\omega, \varepsilon, P_c)$ différente. Le décodeur BP-RNN non spécialisé entraîné comme dans [10] est aussi présent. Pour éviter l'encombrement, les valeurs de poids sont triées dans un ordre croissant et nous appelons par "profil de poids" le jeu ordonné des poids. Notons que des profils de poids différents indiquent des jeux de valeurs de poids différents, tandis que des profils similaires indiquent des jeux de valeurs de poids similaires. Néanmoins, dans le second cas, des poids similaires peuvent être appliqués sur des arêtes différentes.

Nous considérons le Code-1 et nous montrons les profils de poids pour les étages data-pass et a posteriori. L'étage data-pass est implémenté avec l'équation (1.27), avec un poids par neurone appliqué sur toutes ses arêtes rentrantes. De ce fait, le nombre de poids est le même pour les étages data-pass et a posteriori. Il est égal au nombre d'arêtes dans le graphe de Tanner, soit 192 pour le Code-1. Les paramètres d'entraînement restent similaires à ceux de la sous-section 2.3.1 et seront décrits plus en détail dans la sous-section 3.4.1. Les profils de poids dans la figure 3.3 indiquent clairement que l'optimisation des poids répond différemment aux jeux d'entraînements correspondants aux différentes classes ν-$(\omega, \varepsilon, P_c)$ considérées. En effet, les profils de poids de chaque BP-RNN spécialisé sur les ensembles absorbants sont différents deux à deux pour les étages data-pass et a posteriori. Ceci illustre la diversité induite par l'entraînement de plusieurs BP-RNNs spécialisés. Notons également

que le BP-RNN entraîné sur la classe 7-(5, 8, (5, 8)) possède un profil de poids proche de celui du BP-RNN entraîné comme dans [10] pour l'étage data-pass, mais différent pour l'étage a posteriori. L'entraînement sur une classe d'ensembles absorbants permet donc d'obtenir une optimisation des poids différente de celle sur un jeu d'entraînement standard.

3.3 Sélection des BP-RNNs et architecture de décodage

Grâce à la procédure de la sous-section 3.2.3, un décodeur BP-RNN spécialisé est entraîné pour chaque classe ν-$(\omega, \varepsilon, P_c)$ (exceptées celles avec $\omega = 0$), avec une taille d'ensemble absorbants $\nu \leq \nu_{\max}$. Le choix de $\nu_{\max}$ sera détaillé dans la sous-section 3.4.1. Il convient toutefois de préciser dès maintenant que le paramètre $\nu_{\max}$ prend des valeurs plus grandes que dans le chapitre précédent afin de traiter des codes LDPC courts avec une meilleure capacité de correction que le Code-3 ou le Code-4 par exemple.

De manière similaire au chapitre 2, la problématique est maintenant d'organiser la diversité de BP-RNNs spécialisés obtenue en une architecture de décodage. Pour réduire le nombre de décodeurs BP-RNNs considérés dans une architecture, nous proposons tout d'abord une procédure de sélection sous-optimale, moins complexe que la sélection optimale de la sous-section 2.2.4. La procédure de sélection optimale devient en effet irréalisable en raison de l'augmentation du nombre de classes (et donc de décodeurs spécialisés). Nous proposons ensuite deux architectures de décodage, dans lesquelles les BP-RNNs sélectionnés sont exécutés soit en parallèle, soit en série. Enfin, des métriques évaluant la complexité calculatoire et la latence de décodage sont définies pour les deux architectures.

3.3.1 Sélection sous-optimale d'une diversité de BP-RNNs

Pour les mêmes raisons de réduction de la complexité et d'effets d'entraînement de décodeur similaires qu'en sous-section 2.2.4, il est souhaitable de sélectionner un nombre réduit de décodeurs parmi les BP-RNNs initialement entraînés. Nous désignons de nouveau par J le nombre total de décodeurs BP-RNNs initialement entraînés et par $\mathcal{D}_J$ la diversité de décodage correspondante.

Notons que le nombre total J de décodeurs BP-RNNs entraînés dans ce chapitre est plus grand que dans le chapitre précédent. En effet, nous considérons ici des valeurs de $\nu_{\max}$ plus importantes que dans le chapitre 2 afin de traiter des motifs d'erreurs plus grands, correspondant à la capacité de correction du code LDPC court traité. Par exemple pour le Code-1, $\nu_{\max}$ était fixé à 4 dans le chapitre précédent et $J = 17$ BP-RNNs ont été entraînés. Dans ce chapitre, $\nu_{\max}$ est fixé à 8 et $J = 52$ BP-RNNs sont entraînés (d'après le tableau 3.1) [3].

3. Précisons toutefois que le nombre total de classes (de types étendus) dans le tableau 3.1 est en réalité 55, mais trois d'entre elles correspondent à des supports de mot de codes autre que le mot tout zéro, de tailles $\nu = 6$ ou $\nu = 8$. L'entraînement n'est donc pas réalisé sur ces trois classes.

Le calcul de $\binom{J}{Z}$ combinaisons d'indices pour appliquer la règle de sélection (2.6) et obtenir $\mathcal{D}_Z^{\text{opt}}$ devient ainsi impossible en pratique.

Pour remédier à ce problème de complexité, nous proposons une procédure de sélection sous-optimale des décodeurs BP-RNNs. Comme la sélection optimale, cette approche repose sur l'évaluation de la complémentarité des BP-RNNs entraînés, selon leurs probabilités conjointes d'échecs. Nous proposons de manière itérative de trier une liste de décodeurs choisis pour minimiser le taux d'erreur paquet et maximiser la diversité avec les décodeurs précédemment choisis. Ainsi, un jeu de test $\mathcal{T}_{\text{test}}$, contenant des mots bruités aléatoirement construit à l'aide de (1.2), est créé dans un premier temps. Les ensembles [4] $\mathcal{F}_j \subset \mathcal{T}_{\text{test}}$ sont alors calculés pour chaque BP-RNN D_j, avec $j = 1, \ldots, J$. Nous construisons ensuite une liste ordonnée des indices de décodeurs, notée par $\mathfrak{J}$. Nous commençons par initialiser $\mathfrak{J}$ avec l'ensemble vide, $\mathfrak{J} = \emptyset$. Pour ajouter un nouvel indice j_{new} à $\mathfrak{J}$, nous utilisons la règle suivante :

$$j_{\text{new}} = \underset{j \in \{1,\ldots,J\} \setminus \mathfrak{J}}{\operatorname{argmin}} \left| \mathcal{F}_{\mathfrak{J}} \cap \mathcal{F}_j \right|, \tag{3.2}$$

où $\mathcal{F}_{\mathfrak{J}} := \mathcal{T}_{\text{test}}$, si $\mathfrak{J} = \emptyset$ et $\mathcal{F}_{\mathfrak{J}} := \cap_{j \in \mathfrak{J}} \mathcal{F}_j$, sinon. La règle ci-dessus est appliquée récursivement J fois, jusqu'à ce que $\mathfrak{J}$ contienne tous les indices de décodeurs $j = 1, \ldots, J$, dans un ordre trié. Si l'argument minimum de (3.2) n'est pas unique, un choix arbitraire est effectué.

Précisons que quand $\mathfrak{J} = \emptyset$, la règle (3.2) se réécrit en $j_{\text{new}} = \operatorname{argmin}_{j \in \{1,\ldots,J\} \setminus \mathfrak{J}} \left| \mathcal{F}_j \right|$. Ainsi, le premier indice ajouté à la liste correspond au décodeur minimisant le taux d'erreur paquet. Quand $\mathfrak{J} \neq \emptyset$, le nouvel indice ajouté à la liste représente le décodeur le plus complémentaire avec ceux dont les indices appartiennent déjà à $\mathfrak{J}$, dans le sens où ce nouveau décodeur minimise le nombre de mots sur lesquels tous les décodeurs indexés par $\mathfrak{J} \cup \{j_{\text{new}}\}$ échouent.

Pour $Z \leq J$, nous indiquons la sous-liste des Z premiers indices de $\mathfrak{J}$ par $\mathfrak{J}(1{:}Z) \subset \mathfrak{J}$. La diversité (sous-optimale) $\mathcal{D}_Z$ de BP-RNNs spécialisés est alors définie par :

$$\mathcal{D}_Z := \left\{ D_j \mid j \in \mathfrak{J}(1{:}Z) \right\} \tag{3.3}$$

Il convient de remarquer que $\mathcal{D}_Z$ est une liste ordonnée de décodeurs de taille Z. Pour le Code-1, les profils de poids affichés dans la figure 3.3 correspondent aux décodeurs BP-RNNs de $\mathcal{D}_{Z=3}$ pour un SNR = 5 dB. Les différences observées entre les profils de poids de ces 3 BP-RNNs coïncident donc avec leur complémentarité en termes des probabilités conjointes d'échecs.

Les Z BP-RNNs de $\mathcal{D}_Z$ peuvent ensuite être soit utilisés dans une architecture parallèle, comme décrit en sous-section 2.2.3, soit dans une architecture série (qui sera discutée dans la sous-section suivante). Enfin, la valeur de Z est généralement dictée par un compromis complexité/performance.

4. Rappelons que $\mathcal{F}_j$ est le sous-ensemble des mots bruités sur lesquels le BP-RNN D_j a échoué.

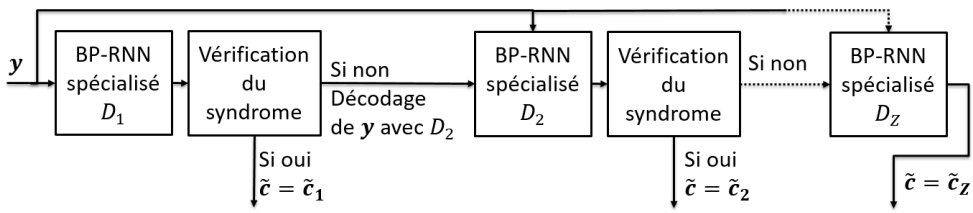


Figure 3.4 – Architecture série.

3.3.2 *Architecture série d'une diversité de BP-RNNs*

Nous considérons une diversité de BP-RNNs $\mathcal{D}_Z$, comprenant Z BP-RNNs spécialisés. Par souci de simplicité, nous indexons les décodeurs BP-RNNs dans $\mathcal{D}_Z$ de 1 à Z, soit $\mathcal{D}_Z = \{D_1, \ldots, D_Z\}$. Pour organiser la diversité $\mathcal{D}_Z$ en une architecture de décodage, une architecture parallèle telle que montrée en figure 2.2 peut tout d'abord être envisagée. De plus, étant donné que $\mathcal{D}_Z$ est une liste ordonnée de décodeurs, nous proposons une architecture de décodage série pour organiser les BP-RNNs de $\mathcal{D}_Z$ afin de réduire la complexité et ne pas devoir utiliser les Z décodeurs en parallèle.

Dans l'architecture série, illustrée dans la figure 3.4, les décodeurs BP-RNNs constituants sont exécutés séquentiellement selon l'ordre obtenu par la procédure de tri de la sous-section 3.3.1. Le décodage s'arrête dès qu'un décodeur BP-RNN D_j converge vers un mot de code $\tilde{\mathbf{c}}_j$ (syndrome$(\tilde{\mathbf{c}}_j) = 0$) et la sortie $\tilde{\mathbf{c}}$ de l'architecture sérié est égale à $\tilde{\mathbf{c}}_j$. Le décodage est ainsi réussi si $\tilde{c}$ est égal au mot de code transmis. Si aucun décodeur BP-RNN ne produit de mot de code, le décodage échoue. Par simplicité, nous prenons $\tilde{\mathbf{c}}_Z$ comme la sortie de l'architecture série dans ce second cas.

3.3.3 *Métriques de latence et de complexité*

Nous détaillons dans cette sous-section les différentes métriques de complexité calculatoire et de latence de décodage des architectures série et parallèle d'une diversité $\mathcal{D}_Z$ de BP-RNNs, dans laquelle tous les décodeurs constituants sont configurés avec un nombre maximum d'itérations de décodage I_{test}. Nous proposons tout d'abord d'évaluer leur complexité calculatoire moyenne en termes du nombre moyen d'itérations de décodage effectuées, puis d'utiliser cette métrique pour mesurer la complexité calculatoire moyenne en termes de mises à jour du check-pass. Le calcul de la complexité calculatoire maximale est également décrit. Les métriques de latence de décodage moyenne et maximale sont ensuite explicitées. Enfin, une évaluation haut niveau de la complexité matérielle est fournie.

3.3.3.1 Complexité moyenne en termes du nombre d'itérations

Pour un BP-RNN D_j, nous notons par $I_{D_j,\mathbf{y}} \leq I_{\text{test}}$ le nombre d'itérations effectuées par D_j pour décoder le signal reçu $\mathbf{y}$. Nous définissons ensuite $\overline{I}_{D_j} := \mathbb{E}_{\mathbf{y}}(I_{D_j,\mathbf{y}})$ le nombre moyen d'itérations du décodeur D_j, où $\mathbb{E}$ est l'opérateur espérance. En pratique, $\overline{I}_{D_j}$ est estimé en moyennant le nombre d'itérations effectuées pour décoder les mots bruités d'un jeu de test $\mathcal{T}_{\text{test}}$. Le nombre moyen d'itérations d'une architecture de décodage (parallèle ou série) utilisant $\mathcal{D}_Z$ est alors défini comme suit :

$$\overline{I}(\mathcal{D}_Z) = \sum_{D_j \in \mathcal{D}_Z} \overline{I}_{D_j}. \tag{3.4}$$

Il convient de remarquer que pour l'architecture série, $I_{D_j,\mathbf{y}}$ peut être égal à zéro dans le cas où le processus de décodage s'arrête avant d'atteindre D_j. Ainsi, $\overline{I}_{D_j}$ peut être proche de 0 si le décodeur D_j est rarement utilisé. Ceci diffère de l'architecture parallèle, qui utilise tous les BP-RNNs à chaque décodage. Enfin, puisque la complexité calculatoire croît linéairement avec le nombre d'itérations de décodage effectuées par le décodeur, nous utilisons le nombre moyen d'itérations de décodage $\overline{I}(\mathcal{D}_Z)$ comme une première mesure de la complexité calculatoire moyenne pour l'architecture de décodage basée sur $\mathcal{D}_Z$.

3.3.3.2 Complexité en termes du nombre de mises à jour du check-pass

Comme expliqué dans le paragraphe 1.1.2.1, les mises à jour des messages passant par les noeuds de parité correspondent aux opérations les plus coûteuses lors du décodage BP. Compter le nombre de ces mises à jour check-pass est donc également pertinent pour évaluer la complexité d'un décodeur basé sur le BP [36].

Nous proposons ainsi d'évaluer les complexités moyenne et maximale de nos architectures de décodage en termes du nombre de mises à jour du check-pass. Pour un BP-RNN D_j, le nombre moyen de mises à jour du check-pass est simplement donné par le nombre d'arêtes dans le graphe de Tanner, multiplié par le nombre moyen d'itérations $\overline{I}_{D_j}$:

$$\overline{C}_{D_j} = \overline{I}_{D_j} \sum_{m=1}^{M} d_m \tag{3.5}$$

La complexité moyenne d'une architecture de décodage utilisant $\mathcal{D}_Z$ est alors calculée comme suit :

$$\overline{C}(\mathcal{D}_Z) = \sum_{D_j \in \mathcal{D}_Z} \overline{C}_{D_j} = \overline{I}(\mathcal{D}_Z) \sum_{m=1}^{M} d_m \tag{3.6}$$

la deuxième égalité provenant de l'équation (3.4) et du fait que les BP-RNNs de $\mathcal{D}_Z$ se basent tous sur le même graphe de Tanner.

Détaillons maintenant notre métrique de complexité calculatoire maximale. Pour un unique BP-RNN D_j, la complexité calculatoire maximale est atteinte lorsque les I_{test} itérations de décodage sont effectuées. Elle s'évalue donc comme suit :

$$C_{D_j}^{\max} = I_{\text{test}} \sum_{m=1}^{M} d_m \tag{3.7}$$

La complexité calculatoire maximale d'une architecture de décodage (parallèle ou série) basée sur une diversité de BP-RNNs $\mathcal{D}_Z$ s'obtient alors en multipliant $C_{D_j}^{\max}$ par Z, soit :

$$C^{\max}(\mathcal{D}_Z) = ZI_{\text{test}} \sum_{m=1}^{M} d_m \tag{3.8}$$

Il convient de noter que nous utiliserons les métriques de cette section uniquement à des fins de comparaison, en termes de complexité calculatoire, avec le décodeur BP neuronal basé sur de l'élagage des noeuds de parité (Pruning Based Neural BP ou PB-NBP en anglais) [11].

3.3.3.3 Latence en termes du nombre d'itérations

Pour l'architecture série, $\overline{I}(\mathcal{D}_Z)$ peut aussi être vu comme une mesure de la latence de décodage moyenne. En effet, les itérations de décodage sont réalisées séquentiellement pour chaque décodeur constituant et les décodeurs composant l'architecture série sont aussi exécutés séquentiellement. Toutefois, ce n'est pas le cas pour l'architecture parallèle. Ainsi, nous définissons la latence de décodage pour l'architecture parallèle pour un mot bruité $\mathbf{y}$ donné comme

$$L_{\mathcal{D}_Z,\mathbf{y}} = \max_{D_j \in \mathcal{D}_Z} I_{D_j,\mathbf{y}}, \tag{3.9}$$

$L_{\mathcal{D}_Z,\mathbf{y}}$ correspond donc au nombre maximum d'itérations effectuées par les décodeurs constituants pour décoder $\mathbf{y}$. La latence de décodage moyenne de l'architecture parallèle est alors définie par :

$$\overline{L}(\mathcal{D}_Z) := \mathbb{E}_{\mathbf{y}}(L_{\mathcal{D}_Z,\mathbf{y}}), \tag{3.10}$$

et est estimée en moyennant sur l'ensemble des mots bruités $\mathbf{y}$ du jeu de test.

En cas de non convergence vers un mot de code, ZI_{test} itérations sont effectuées séquentiellement avec l'architecture série de $\mathcal{D}_Z$. Sa latence de décodage maximale est donc égale à ZI_{test}. Pour l'architecture parallèle, chacun des Z décodeurs effectue I_{test} itérations indépendamment. La latence de décodage maximale de l'architecture parallèle est donc égale à I_{test}. L'architecture parallèle donne donc une latence de décodage maximale réduite par rapport à l'architecture série.

3.3.3.4 Complexité matérielle

Nous comparons ici nos architectures de décodages composée de BP-RNNs avec le BP en termes de quantité mémoire et de nombre d'opérations. Comme nos décodeurs BP-RNNs utilisent le data-pass défini en (1.27), ils peuvent être implémentés en matériel avec des architectures conventionnelles du BP. Dans ce cas, le coût en mémoire supplémentaire pour un seul décodeur BP-RNN par rapport au décodeur BP provient uniquement du stockage des poids. Le décodeur BP-RNN nécessite également une multiplication supplémentaire pour chaque poids dans (1.27) et (1.26), ce qui induit également un coût supplémentaire dans l'implémentation matérielle du BP-RNN. Une architecture de décodage composée de Z BP-RNNs induit donc Z multiplications supplémentaires pour chaque poids de (1.27) et (1.26), ainsi que Z instanciations en mémoire supplémentaire pour les poids par rapport au BP.

Enfin, les Z BP-RNNs de l'architecture parallèle doivent être instanciés en matériel afin de paralléliser les Z décodages dans l'implémentation matérielle. Pour l'architecture série, seulement un unique décodeur peut être instancié, puis réutilisé pour exécuter séquentiellement les Z BP-RNNs (tout en mettant à jour l'ensemble de poids correspondant). L'architecture série présente donc une complexité matérielle plus faible que l'architecture parallèle.

3.4 Résultats numériques

Cette section est dédiée à l'évaluation par simulation des architectures de décodage série et parallèle d'une diversité de BP-RNNs spécialisés sur les ensembles absorbants. Nous évaluons ainsi les performances en termes de taux d'erreur paquet, complexité calculatoire et latence de décodage pour les deux architectures.

3.4.1 Paramètres de simulation

Pour nos simulations, nous utilisons les deux codes LDPC de rendement de codage $R_c = 1/2$ et de longueur $N = 64$ (Code-1), ou $N = 128$ (Code-2) bits, détaillés dans la section 1.1.5. Nous entraînons un décodeur BP-RNN spécialisé pour chaque classe d'ensembles absorbants ν-$(\omega, \varepsilon, P_c)$, avec une taille $\nu \leq \nu_{\max}$:

- Code-1 : $\nu_{\max} = 8$ est choisi, ce qui donne un total de $J = 52$ classes, d'après le tableau 3.1 (en éliminant les trois classes correspondant à des mots de code, c.-à-d., avec $\omega = 0$).
- Code-2 : $\nu_{\max} = 7$ est choisi, ce qui correspond à un total de $J = 120$ classes, d'après le tableau 3.1.

Pour le Code-2, le choix de $\nu_{\max} = 7$ est lié à des raisons de complexité, afin de limiter le nombre de décodeurs à entraîner. Cependant, nous notons que pour un SNR = 4 dB dans

Tableau. 3.2 – Paramètres Keras.

Paramètres	Valeurs des paramètres
Méthode de descente de gradient	RMSprop [53] (initialisé avec un pas d'adaptation de 10^{-3})
Nombre d'époques	10
Taille des paquets d'entraînement	8192
Nombre de paquets d'entraînement	37 à 122 (dépend du SNR)
Taille des paquets de test	16384
Nombre de paquets de test	19 à 61 (dépend du SNR)

la région du *waterfall* du Code-2, le nombre moyen d'erreurs est $Np = 7.2$, où $N = 128$ et $p = Q(1/\sigma) = 0.0565$ est la probabilité d'erreur binaire définie par l'équation (1.13) pour le canal AWGN à entrée binaire de variance σ^2. Le choix $v_{\max} = 7$ permet ainsi de spécialiser le BP-RNN sur une partie des ensembles absorbants dégradant la performance du BP pour un SNR ≥ 4 dB. Pour le Code-1 et le même SNR = 4 dB, la taille d'un ensemble d'erreurs aléatoire est inférieure ou égale à $v_{\max} = 8$, avec une probabilité légèrement plus grande que 0.99. Les ensembles absorbants dégradant la performance du BP pour un SNR ≥ 4 dB sont donc tous pris en compte pour la spécialisation du BP-RNN.

Les architectures de décodage évaluées dans cette section sont composées soit de BP-RNNs définis avec les équations (1.27), (1.22) et (1.26), soit des MS-RNNs définis avec les équations (1.27), (1.11) et (1.26).

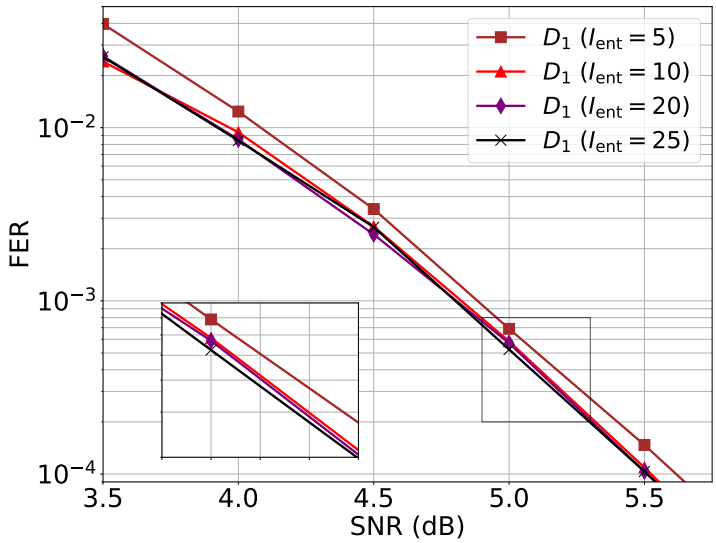
Chaque BP-RNN/MS-RNN spécialisé est entraîné indépendamment, avec un jeu d'entraînement construit en suivant la procédure de la sous-section 3.2.3. De plus, nous entraînons un décodeur BP-RNN non spécialisé selon la méthode décrite dans [10], afin de fournir une référence pour les résultats présentés. Ceci est répété avec un MS-RNN non spécialisé. Le choix du nombre maximal d'itérations de décodage pendant les phases d'entraînement et de test, I_{ent} et I_{test} (cf le paragraphe 1.3.2.4), sera détaillé dans la sous-section suivante. Le choix du SNR d'entraînement des BP-RNNs sera quant à lui discuté dans la sous-section 3.4.3.

Enfin, la librairie Keras a été de nouveau utilisée pour l'entraînement. Les paramètres d'entraînement correspondants sont montrés dans le tableau 3.2 et sont par ailleurs égaux à ceux utilisés dans le tableau 2.4.

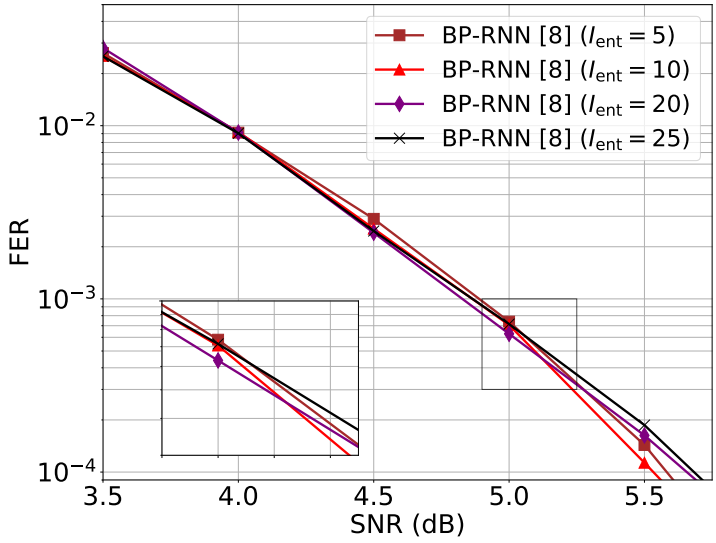
3.4.2 Nombre maximal d'itérations pour l'entraînement et le test

Nous étudions ici l'impact du nombre maximum d'itérations de décodage utilisé pour la phase d'entraînement I_{ent} sur la performance des BP-RNNs spécialisés. Le nombre maximum d'itérations de décodage pour le test I_{test} est fixé à 25.

Nous considérons le Code-1 et nous entraînons tous les décodeurs BP-RNNs pour $I_{\text{ent}} \in \{5, 10, 20, 25\}$. Il convient de préciser que tous les BP-RNNs sont entraînés pour chaque valeur de SNR comprise entre 1 dB et 6 dB, avec un pas de 0.5 dB. De plus, nous



(a) BP-RNN D_1, entraîné sur la classe $7 - (5, 8, (5, 8))$.

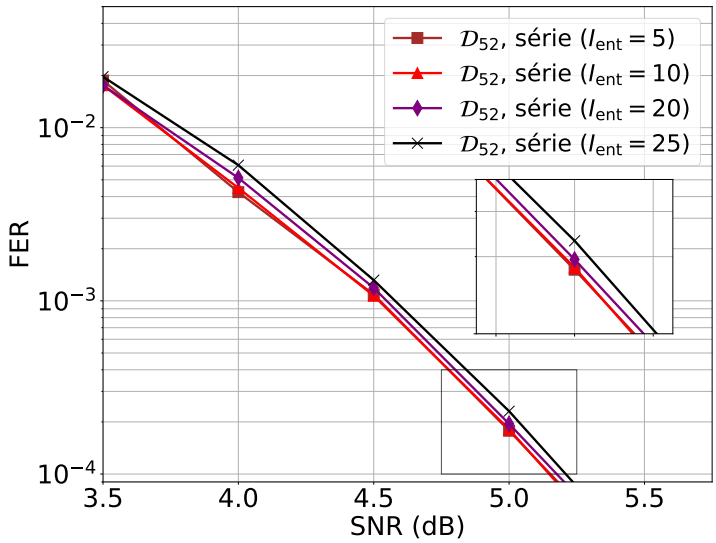


(b) BP-RNN [10].

utilisons le même SNR pour l'entraînement et le test. La figure 3.4 montre les résultats en termes de FER, avec $I_{test} = 25$, pour :

(a) Le décodeur BP-RNN spécialisé sur la classe 7-$(5, 8, (5, 8))$, indiqué par D_1 dans la légende. Ce décodeur est en effet le premier sélectionné par la règle (3.2) pour un SNR $= 5$ dB [5]. Il sert à titre d'exemple pour l'impact de I_{ent} sur l'entraînement d'un

5. Voir aussi le tableau 3.3 et la discussion correspondante de la sous-section suivante.



(c) Tous les 52 BP-RNNs spécialisés, architecture série.

Figure 3.4 – Impact du paramètre d'entraînement I_{ent} sur la performance en termes de FER, pour les décodeurs BP-RNNs utilisant $I_{test} = 25$ (Code-1).

BP-RNN sur des motifs d'erreurs avec un support d'ensemble absorbant.

(b) Le décodeur BP-RNN non spécialisé de [10].

(c) L'architecture série utilisant les 52 décodeurs BP-RNNs spécialisés.

Nous n'observons aucune différence notable dans la performance en termes de taux d'erreur paquet, excepté dans la figure 3.4(a), où la performance en termes de FER pour $I_{ent} = 5$ est légèrement dégradé par rapport à $I_{ent} \in \{10, 20, 25\}$. Dans la suite, $I_{ent} = 10$ est choisi, ce qui permet un entraînement plus rapide. Des résultats similaires ont été obtenus pour le Code-2.

3.4.3 *Impact du SNR d'entraînement*

Le choix du même SNR pour l'entraînement et le test résulte des travaux de [10], dans lesquels il est démontré qu'un jeu entraînement mélangeant plusieurs valeurs de SNRs ne correspondant pas aux SNRs de test implique une dégradation des performances. Nous revenons ici sur ce choix, en proposant d'entraîner les décodeurs BP-RNNs pour un SNR donné de la région du *waterfall* du BP, puis de les tester sur différentes valeurs de SNRs.

Nous considérons le Code-1 et les BP-RNNs entraînés sur $J = 52$ classes pour un SNR $\in \{2, 3, 4, 5, 6\}$ dB. Pour chacun de ces SNRs d'entraînement, un ensemble de $J = 52$ poids optimisés est obtenu. Nous testons ensuite ces BP-RNNs pour chaque valeur de SNR comprise entre 1 dB et 6 dB, avec un pas de 0.5 dB. Les résultats correspondants sont illustrés

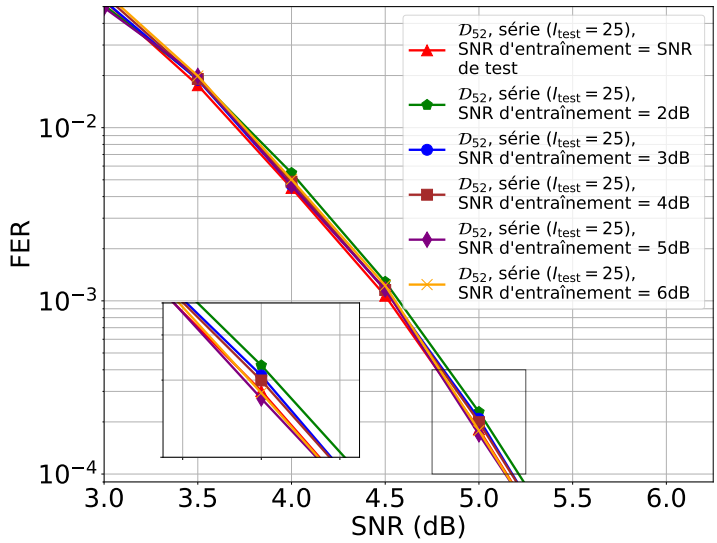


Figure 3.5 – Résultats FER du Code-1, pour différents SNRs d'entraînement.

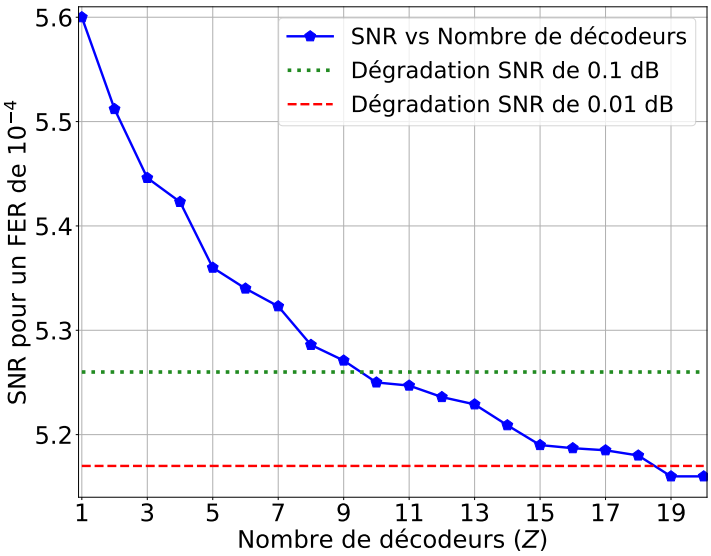
dans la figure 3.5 pour $I_{\text{test}} = 25$. De plus, la performance en termes de FER des BP-RNNs entraînés au SNR de chaque valeur de test est aussi affichée à des fins de comparaison. Nos observons des performances similaires entre les différentes combinaisons SNR d'entraînement/SNR de test. En conséquence, entraîner les BP-RNNs à un unique SNR de la région du *waterfall* suffit ici pour couvrir tout les SNRs de test. Les mêmes observations ont été retrouvées pour le Code-2. Dans la suite, le SNR d'entraînement est fixé à 5 dB pour les deux codes [6].

3.4.4 *Sélection d'une diversité de BP-RNNs*

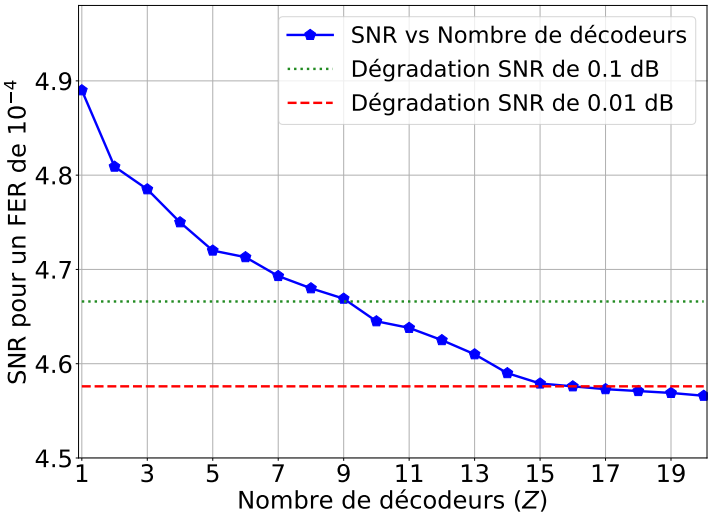
Nous appliquons maintenant la procédure de sélection sous-optimale, pour le Code-1 et le Code-2. Pour chaque code, nous considérons les J BP-RNNs entraînés sur les J classes ν-$(\omega, \varepsilon, P_c)$, avec un SNR d'entraînement fixé à 5 dB et $I_{\text{ent}} = 10$. Un jeu de test $\mathcal{T}_{\text{test}}$ contenant 10^8 mots de code bruités (SNR $=$ 5 dB) est ensuite créé pour évaluer la performance de correction d'erreurs individuelle de chaque BP-RNN spécialisé. Nous les ordonnons alors avec la règle de diversité (3.2), puis nous sélectionnons une diversité $\mathcal{D}_Z$ de décodeurs BP-RNNs selon (3.3), correspondant à Z classes ν-$(\omega, \varepsilon, P_c)$ différentes. Par la suite, nous ne considérons que les décodeurs BP-RNNs spécialisés sur les classes sélectionnées.

La figure 3.6 montre le SNR requis pour obtenir un FER cible de 10^{-4}, en fonction du nombre de décodeurs BP-RNNs sélectionnés Z, pour le Code-1 (a) et le Code-2 (b) . Les résultats sont affichés uniquement pour $Z = 1, \ldots, 20$, car nous n'observons aucune autre

6. Des SNRs d'entraînement fixés à 2 dB, 3 dB, 4 dB et 6 dB ont conduit à des résultats similaires.



(a) Code-1.



(b) Code-2.

Figure 3.6 – SNR pour une cible FER $= 10^{-4}$, en fonction de Z.

amélioration du taux d'erreur paquet pour des valeurs de Z plus élevées.

Pour le Code-1, nous choisissons $Z = 10$, ce qui correspond à une dégradation de la performance inférieure à 0.1 dB par rapport au cas où les J décodeurs BP-RNNs sont utilisés. La même procédure est effectuée pour sélectionner un nombre de décodeurs BP-RNNs pour le Code-2. Ici aussi, la valeur $Z = 10$ permet d'obtenir une dégradation de la performance inférieure à 0.1 dB. Les $Z = 10$ décodeurs BP-RNNs sélectionnés et leurs classes

Tableau. 3.3 – Classes ν-$(\omega, \varepsilon, P_c)$ sélectionnées et leurs décodeurs BP-RNNs associés.

Code-1		Code-2	
Dec.	Classe ν-$(\omega, \varepsilon, P_c)$	Dec.	Classe ν-$(\omega, \varepsilon, P_c)$
D_1	7-(5, 8, (5, 8))	D_1	7-(7, 11, (6, 11, 1))
D_2	7-(1, 9, (0, 9, 1))	D_2	5-(7, 9, (7, 9))
D_3	4-(2, 5, (2, 5))	D_3	6-(4, 10, (4, 10))
D_4	7-(3, 7, (1, 7, 2))	D_4	7-(5, 9, (5, 9))
D_5	8-(2, 9, (1, 8, 1, 1))	D_5	6-(8, 10, (8, 10))
D_6	6-(2, 7, (2, 6, 0, 1))	D_6	7-(5, 11, (4, 11, 1))
D_7	5-(1, 7, (1, 7))	D_7	7-(5, 8, (5, 8))
D_8	6-(2, 7, (1, 7, 1))	D_8	7-(7, 7, (7, 7))
D_9	7-(1, 9, (1, 8, 0, 1))	D_9	7-(3, 11, (3, 11))
D_{10}	3-(3, 3, (3, 3))	D_{10}	7-(7, 13, (7, 13))

correspondantes sont affichés dans le tableau 3.3 pour les deux codes. Enfin, pour les deux codes, les MS-RNNs spécialisés sont entraînés sur les mêmes classes que les BP-RNNs afin de comparer les performances obtenues pour une même sélection de classes.

3.4.5 Performances FER

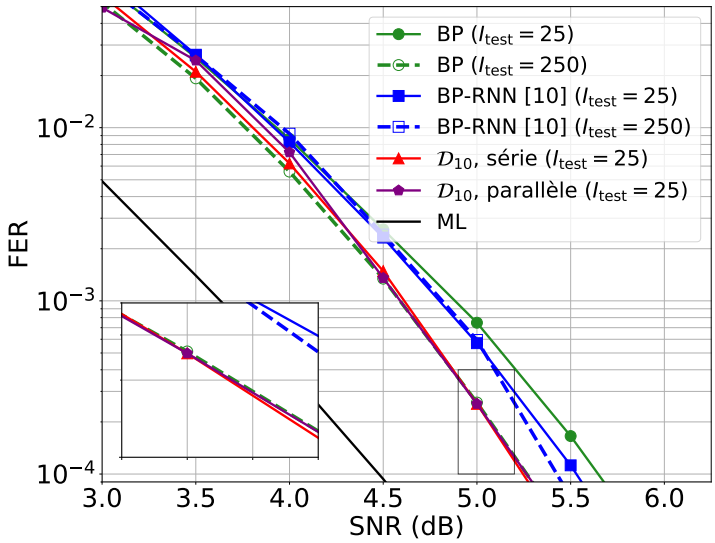
3.4.5.1 Diversité de BP-RNNs spécialisés

Nous considérons la diversité $\mathcal{D}_{10}$, composée des $Z = 10$ décodeurs BP-RNNs issues de sélection présentée dans la sous-section précédente et nous évaluons la performances en termes de FER pour les architectures série et parallèle. La figure 3.7 montre les résultats de taux d'erreur paquet pour le Code-1 (a) et pour le Code-2 (b). En guise de références, nous montrons également les taux d'erreur paquets du décodeur BP et du décodeur BP-RNN de [10], avec $I_{\text{test}} = 25$ et $I_{\text{test}} = 250$ itérations. Cette deuxième valeur de I_{test} correspond au nombre maximal d'itérations effectuées par l'ensemble des décodeurs BP-RNNs de la diversité $\mathcal{D}_{10}$. De plus, pour le Code-2, les résultats en termes de FER du décodeur PN-NBP [11] sont affichés à titre de comparaison pour $I_{\text{test}} = 6$ [7]. Précisons que plusieurs variantes du décodeur PB-NBP sont présentées dans [11] et nous ne considérons ici que le décodeur PB-NBP D_1 puisqu'il obtient la meilleure performance. Toutefois, toutes les variantes reposent bien sur l'approche présentée en sous-section 1.3.3.2. Les gains sont évalués pour un taux d'erreur paquet de 10^{-4}.

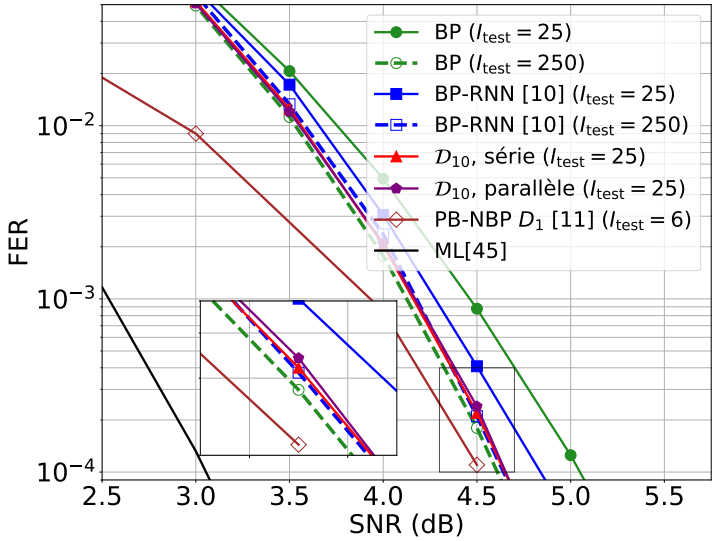
Pour le Code-1, nous observons en premier lieu que les architectures série et parallèle présentent pratiquement les mêmes performances [8]. Comparée au décodage BP conventionnel, la diversité de BP-RNNs $\mathcal{D}_{10}$ produit un gain SNR de 0.4 dB par rapport à BP($I_{\text{test}} = 25$).

7. Cette valeur et la performance en terme de FER du décodeur PB-NBP ont directement été relevées dans [11].

8. Ceci a d'ailleurs été vérifié pour différentes valeurs de Z.



(a) Résultats FER pour le Code-1.



(b) Résultats FER pour le Code-2.

Figure 3.7 – Résultats FER, BP-RNNs spécialisés.

Cette comparaison est particulièrement pertinente pour les applications avec des exigences de latence strictes, étant donné que le BP($I_{\text{test}} = 25$) et l'architecture parallèle de $\mathcal{D}_{10}$ ont la même latence de décodage maximale. Si la contrainte de latence de décodage maximale est relâchée, nous pouvons observer que le décodeur BP($I_{\text{test}} = 250$) atteint le même taux d'erreur paquet que la diversité de BP-RNNs $\mathcal{D}_{10}$. Augmenter le nombre d'itérations de

$I_{\text{test}} = 25$ à $I_{\text{test}} = 250$ n'améliore cependant pas la performance du BP-RNN entraîné comme dans [10] (ou seulement légèrement dans la région à très faible FER). De plus, nous notons qu'utiliser uniquement le premier décodeur D_1 de nos BP-RNNs spécialisés occasionne une performance similaire à celle du BP-RNN [10] non spécialisé pour $I_{\text{test}} = 25$. Bien que ceci ne soit pas illustrée sur la figure 3.7 pour des raisons d'encombrement, la comparaison des résultats en termes de FER des figures 3.4(a) et 3.4(b) permet de s'en apercevoir. Enfin, la performance ML se situe à 0.77 dB de la performance de $\mathcal{D}_{10}$.

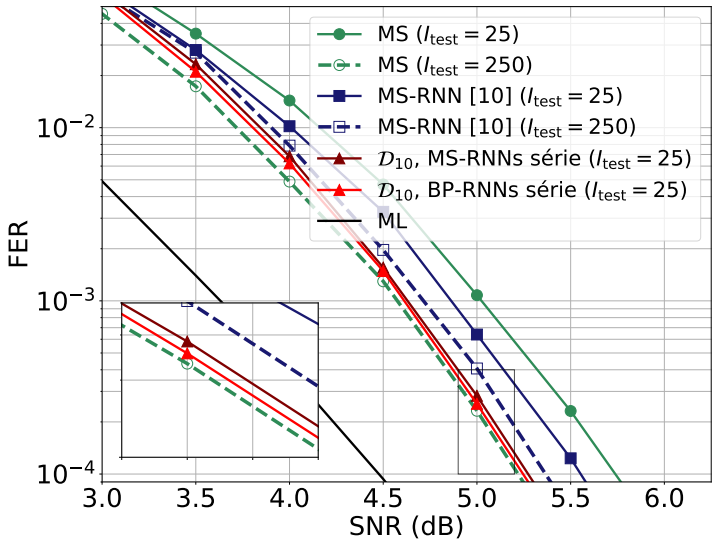
Pour le Code-2, nous relevons également un gain de 0.4 dB pour la diversité de BP-RNNs $\mathcal{D}_{10}$ (parallèle ou série) par rapport au décodeur BP($I_{\text{test}} = 25$). À titre de comparaison, un gain similaire par rapport au BP conventionnel a été rapporté dans [70, Fig. 6], en utilisant une approche de décodage par automorphisme d'ensembles (Automorphism Ensemble Decoding ou AED en anglais) avec 16 décodeurs BP fonctionnant en parallèle [9]. Nous remarquons aussi pour ce code que le décodeur BP($I_{\text{test}} = 250$) parvient à la même performance que la diversité de BP-RNNs $\mathcal{D}_{10}$. Contrairement au Code-1, cette comparaison reste valable pour le décodeur BP-RNN($I_{\text{test}} = 250$) entraîné comme dans [10]. De plus, l'approche par élagage proposée avec PB-NBP D_1 induit une meilleure performance que la diversité $\mathcal{D}_{10}$, en particulier pour des SNRs ≤ 4 dB. Nous montrerons toutefois dans la sous-section 3.4.6 que la complexité calculatoire moyenne et la complexité matérielle sont plus importantes pour le décodeur PB-NBP D_1 par rapport à l'architecture série ou au décodeur BP. Enfin, l'écart à la performance ML est de 1.59 dB pour $\mathcal{D}_{10}$.

3.4.5.2 Diversité de MS-RNNs spécialisés

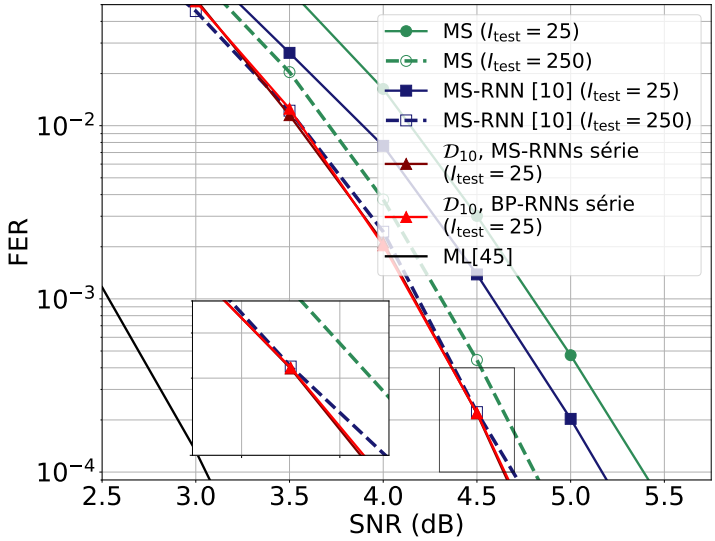
Nous effectuons ici la même étude de performances que dans la sous-section précédente, mais avec des décodeurs basés sur le MS. Une diversité $\mathcal{D}_{10}$ de décodeurs MS-RNNs spécialisés sur les classes du tableau 3.3 est dans un premier temps construite. Nous comparons ensuite son taux d'erreur paquet avec ceux du MS standard et du MS-RNN de [10], pour $I_{\text{test}} = 25$ et $I_{\text{test}} = 250$ itérations. L'architecture de décodage des $Z = 10$ MS-RNNs spécialisés peut être soit série, soit parallèle, les performances étant identiques. La figure 3.8 montre les résultats correspondants.

Pour le Code-1, nous observons tout d'abord que la diversité de MS-RNNs $\mathcal{D}_{10}$ fournit un gain de 0.47 dB par rapport au décodeur MS($I_{\text{test}} = 25$). De plus, la diversité de MS-RNNs $\mathcal{D}_{10}$ obtient une performance proche de son homologue composé de BP-RNNs spécialisés. Les MS-RNNs présentent également une complexité calculatoire plus faible que les BP-RNNs étant donné que la mise à jour des $\beta_{m\to n}^{(i)}$ s'effectue avec l'équation (1.11) au lieu de l'équation (1.22). En conséquence, pour le Code-1, une diversité de MS-RNNs spécialisés sur les ensembles absorbants est à privilégier par rapport à la diversité de BP-RNNs spécialisés. En augmentant le nombre d'itérations à $I_{\text{test}} = 250$, le décodeur MS atteint la même performance que l'architecture série des 10 MS-RNNs spécialisés. Le décodeur MS-RNN [10]($I_{\text{test}} = 250$) possède quant à lui une performance dégradé de 0.1 dB par rapport à la diversité $\mathcal{D}_{10}$ de

9. La courbe AED-16 de [70] n'a pas été incluse dans la figure 3.7b, pour éviter l'encombrement. Le gain rapporté dans [70] a été observé pour 32 itérations de décodage.



(a) Résultats FER pour le Code-1.



(b) Résultats FER pour le Code-2.

Figure 3.8 – Résultats FER, MS-RNNs spécialisés.

MS-RNNs.

Les MS-RNNs spécialisés conservent aussi un avantage net à être utilisés par rapport aux BP-RNNs spécialisés pour le Code-2. En effet, les deux diversités atteignent la même performance. Il convient de remarquer que pour ce code, le MS($I_{test} = 250$) n'obtient pas la performance produite par la diversité $\mathcal{D}_{10}$, avec un écart de 0.16 dB. Enfin le gain induit par

les décodeurs MS-RNNs de $\mathcal{D}_{10}$ par rapport au décodeur MS($I_{\text{test}} = 25$) est de 0.74 dB.

3.4.6 *Évaluation de la complexité et de la latence*

Nous détaillons dans cette sous-section notre étude sur la complexité calculatoire et la latence de décodage des architectures série et parallèle utilisant la diversité de BP-RNNs spécialisés $\mathcal{D}_{10}$.

3.4.6.1 Complexité moyenne en termes du nombre d'itérations

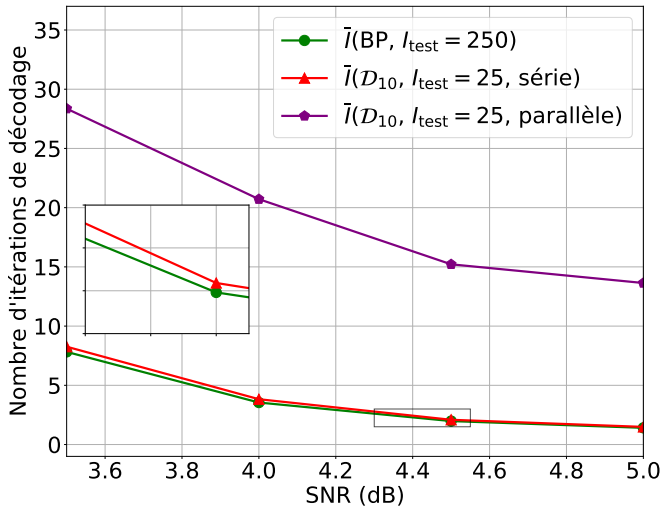
La figure 3.9 affiche les résultats associés à la complexité calculatoire moyenne évaluée avec l'équation (3.4), pour le Code-1 (a) et le Code-2 (b). Nous comparons le BP($I_{\text{test}} = 250$) et la diversité de BP-RNNs $\mathcal{D}_{10}$, organisée en une architecture soit série, soit parallèle. Nous observons pour les deux codes que l'architecture série fournit une complexité calculatoire proche du BP($I_{\text{test}} = 250$). L'architecture parallèle possède quant à elle une complexité calculatoire moyenne plus élevée, étant donné que chaque BP-RNN effectue au moins une itération à chaque décodage. Des observations similaires ont été réalisées pour les architectures série et parallèle composées de MS-RNNs spécialisés.

3.4.6.2 Complexité en termes du nombre de mises à jour du check-pass

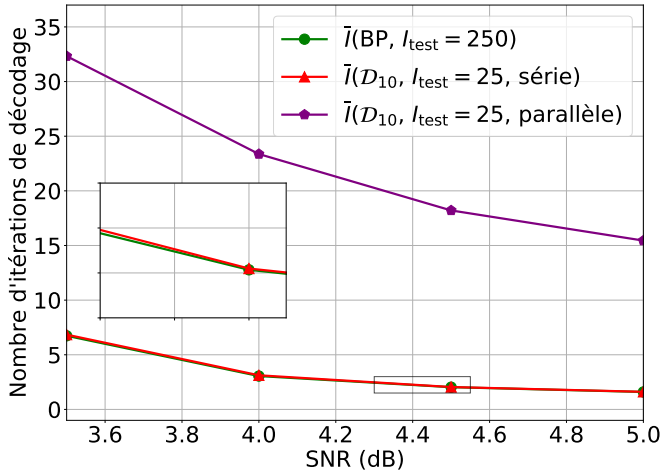
Les résultats correspondants à la complexité moyenne et maximale en termes du nombre de mises à jour du check-pass sont donnés dans le tableau 3.4 pour le Code-2. Nous comparons ici le BP, le décodeur PB-NBP D_1 [11], l'architectures série de $\mathcal{D}_{10}$ et l'architecture parallèle de $\mathcal{D}_{10}$.

Nous remarquons tout d'abord que les architectures série et parallèle de $\mathcal{D}_{10}$ ont la même complexité maximale $C^{\max}$ que BP($I_{\text{test}} = 250$) et qu'elle est 4.85 fois plus grande que celle de PB-NBP D_1. Toutefois, notre architecture série possède une meilleure complexité moyenne que le décodeur PB-NBP D_1. En effet, le décodeur PB-NBP D_1 est pénalisé en termes de complexité moyenne par le fait qu'il utilise un nombre élevé de noeuds de parité lors de la première itération (dont certains peuvent être élagués lors des itérations suivantes). Plus précisément, sa première itération contient 640 noeuds de parité de degré supérieur ou égal à 8 (cf [59, Fig. 4b]), ce qui donne un nombre moyen de mises à jour du check-pass supérieur ou égal à 5120. De plus, dans la figure 3.9(b), nous constatons que le nombre moyen d'itérations de décodage $\bar{I}(\mathcal{D}_{10})$ est égal à 10 pour l'architecture série à un SNR de 3.37 dB. Par conséquent, nous concluons que l'architecture série utilisant $\mathcal{D}_{10}$ possède une complexité moyenne inférieure à celle de PB-NBP D_1 à partir de 3.37 dB.

La complexité calculatoire moyenne de l'architecture parallèle de $\mathcal{D}_{10}$ est 1.82 fois supérieure ou égale à celle de PB-NBP D_1 pour un SNR de 4.5 dB. PB-NBP D_1 permet donc d'obtenir une performance légèrement meilleure que nos 10 BP-RNNs spécialisés mis en parallèle pour des complexités calculatoires moyenne et maximale plus faibles. Si aucune



(a) Code-1.



(b) Code-2.

Figure 3.9 – Complexité moyenne de décodage en termes du nombre moyen d'itérations.

contrainte sur la complexité de l'entraînement n'est imposée, il apparaît donc avantageux d'utiliser le décodeur PB-NBP D_1 par rapport à l'architecture parallèle utilisant $\mathcal{D}_{10}$.

3.4.6.3 Latence de décodage moyenne

La figure 3.10 montre les latences moyennes de décodage du décodeur BP($I_{\text{test}} = 250$) et de la diversité $\mathcal{D}_{10}$ organisée dans une architecture soit série, soit parallèle, pour le Code-1 (b) et le Code-2 (b). Pour le décodeur BP et l'architecture série de $\mathcal{D}_{10}$, le nombre moyen

Tableau. 3.4 – Complexité de décodage (en termes du nombre de mises à jour du check-pass) pour le Code-2.

Architecture de décodage	Complexité de décodage (nombre de mises à jour du check-pass [36])	
	$C^{\max}$	$\overline{C}^{(*)}$
BP($I_{\text{test}} = 25$)	12800	983
BP($I_{\text{test}} = 250$)	128000	1044
PB-NBP D_1 [11] ($I_{\text{test}} = 6$)	25920	Supérieur ou égale à 5120
$\mathcal{D}_{10}$, série ($I_{\text{test}} = 25$)	128000	1044
$\mathcal{D}_{10}$, parallèle ($I_{\text{test}} = 25$)	128000	9324

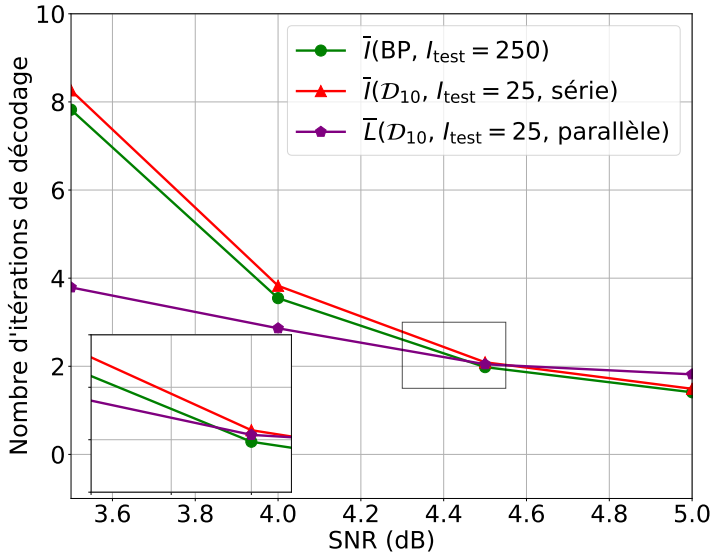
$^{(*)}$ $\overline{C}$ est évaluée en mesurant $\bar{I}$ à SNR = 4.5 dB.

d'itérations de décodage ($\bar{I}$) est une mesure à la fois de la complexité calculatoire moyenne et de la latence de décodage moyenne. Pour l'architecture parallèle de $\mathcal{D}_{10}$, $\bar{I}$ mesure seulement la complexité calculatoire moyenne, tandis que la latence de décodage moyenne est évaluée avec $\bar{L}$ (équation (3.10)).

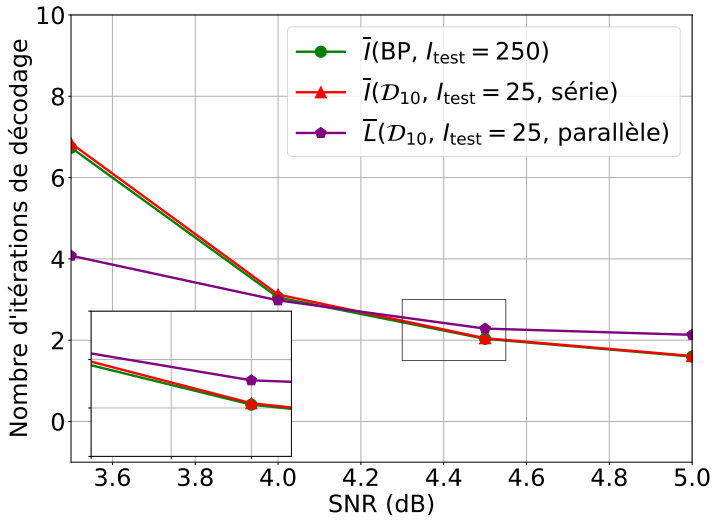
Bien que l'architecture parallèle de $\mathcal{D}_{10}$ présente la latence de décodage maximale la plus faible, nous observons qu'elle montre également un latence de décodage moyenne réduite dans la première partie de région du *waterfall* (jusqu'à 4 dB) pour les deux codes. L'architecture série de $\mathcal{D}_{10}$ peut être vu quant à elle comme une alternative au décodeur BP($I_{\text{test}} = 250$), fournissant non seulement des performances de décodage similaires, mais aussi une complexité calculatoire et une latence de décodage moyennes/maximales similaires. Néanmoins, les architectures parallèle et série de $\mathcal{D}_{10}$ conservent l'avantage de la diversité de décodage, qui sera en particulier exploitée avec une étape de post-traitement de décodage par statistiques ordonnées [28] (Ordered Statistics Decoding ou OSD en anglais) dans le prochain chapitre. Des conclusions similaires peuvent être établies avec une diversité composée de MS-RNNs spécialisés.

3.4.6.4 Complexité matérielle

Le tableau 3.5 détaille le nombre de poids optimisés à stocker en mémoire pour le décodeur PB-NBP D_1 ainsi que pour nos architectures série et parallèle utilisant $\mathcal{D}_{10}$. Nous constatons que PB-NBP D_1 possède 2.75 fois plus de poids que $\mathcal{D}_{10}$ (série ou parallèle) à cause de son architecture neuronale. PB-NBP D_1 requiert ainsi un coût de stockage des poids plus grand que $\mathcal{D}_{10}$. De plus, les connexions entre les noeuds de parité et de donnée diffèrent entre chaque itération de décodage de PB-NBP D_1. Ce décodeur neuronal nécessite donc une architecture matérielle spécifique et très différente de celles traditionnellement employées pour le BP, contrairement à l'architecture de décodage série ou parallèle de $\mathcal{D}_{10}$.



(a) Code-1.



(b) Code-2.

Figure 3.10 – Latence moyenne de décodage en termes du nombre moyen d'itérations.

3.5 Conclusion

Dans ce chapitre, nous avons proposé d'adapter la construction d'une diversité de BP-RNNs pour des codes LDPC courts avec une meilleure capacité de correction que ceux du chapitre 2, comme le code CCSDS (Code-2). À cette fin, nous avons montré que les

Tableau. 3.5 – Nombre de poids des différentes architectures de décodage évaluées.

Architectures de décodage	Nombre de poids
BP($I_{\text{test}} = 25$)	0
BP($I_{\text{test}} = 250$)	0
PB-NBP D_1 [11] ($I_{\text{test}} = 6$)	28416
$\mathcal{D}_{10}$, série ($I_{\text{test}} = 25$)	10240
$\mathcal{D}_{10}$, parallèle ($I_{\text{test}} = 25$)	10240

gains observés dans le chapitre 2 étaient dû aux BP-RNNs spécialisés dans le décodage de motifs d'erreurs avec un support d'ensembles absorbants. En conséquence, nous avons tout d'abord développé un algorithme permettant d'énumérer efficacement et exhaustivement tous les ensembles absorbants d'une taille donnée. Ces derniers sont alors classifiés avec une méthode moins complexe que celle présentée dans le chapitre précédent, puis un BP-RNN est entraîné sur chaque classe grâce à un jeu d'entraînement spécifique. Une nouvelle diversité de décodage de BP-RNNs est ainsi créée, dans laquelle chaque BP-RNN est spécialisé sur le décodage des motifs d'erreurs d'une classe d'ensembles absorbants.

Nous avons ensuite abordé la question de l'organisation d'une diversité de BP-RNNs spécialisés sur les ensembles absorbants en une architecture de décodage. Une nouvelle procédure de sélection des BP-RNNs spécialisés a tout d'abord été proposée afin de réduire leur nombre. Moins complexe que celle du chapitre précédent, elle permet également d'ordonner les BP-RNNs selon leur complémentarité. Deux architectures de décodage ont alors été considérées. La première correspond à l'architecture parallèle des BP-RNNs sélectionnés. La seconde consiste en une exécution séquentielle des BP-RNNs sélectionnés selon l'ordre donné par la procédure de sélection. Dans cette architecture série, un BP-RNN décode uniquement si les BP-RNNs avant ce dernier n'ont pas convergé vers un mot de code.

Les résultats de simulations ont montré que les BP-RNNs spécialisés sur les ensembles absorbants avec l'architecture parallèle améliorent la capacité de correction, sans augmenter la latence de décodage maximale. Nous avons également montré que l'architecture série d'une diversité de BP-RNNs spécialisés est équivalente au décodeur BP avec un même nombre maximal d'itérations de décodage. Ces deux décodeurs présentent en effet des performances similaires, ainsi qu'une complexité calculatoire moyenne et une latence de décodage moyenne similaires. Enfin, le décodeur neuronal basé sur de l'élagage des noeuds de parité obtient une performance améliorée par rapport à l'architecture série (ou au décodeur BP) pour le Code CCSDS, au prix toutefois d'une complexité calculatoire moyenne et matérielle plus importantes. De plus, les architectures série et parallèle conservent l'avantage de la diversité de décodage par rapport au BP ou au décodeur neuronal basé sur de l'élagage des noeuds de parité.

Ces résultats ont donné lieu à une publication dans le journal international IEEE Transactions on Communications en 2022 [J1].

Dans le chapitre suivant, nous voulons nous rapprocher davantage des performances du décodage ML qui sont encore à 0.77 dB et 1.59 dB des performances obtenues avec la

diversité de BP-RNNs spécialisés pour le Code-1 et le Code-2 respectivement. Pour cela, nous allons considérer un post-traitement OSD exploitant mieux la diversité des BP-RNNs que la simple combinaison série ou parallèle. De plus, la combinaison du décodeur BP standard avec un post-traitement OSD est également explorée afin d'en améliorer la capacité de décodage.

Chapitre 4

Post-traitement OSD pour le décodage BP/BP-RNN

Contenu du Chapitre 4

4.1 OSD 88
4.1.1 Principe de l'OSD 88
4.1.2 OSD intégré dans le BP 90
4.1.3 Post-traitement OSD du BP 91
4.1.4 Réduction de complexité 93
4.2 Post-traitement OSD d'une diversité de BP-RNNs spécialisés 96
4.2.1 Motivations et proposition d'un post-traitement OSD des BP-RNNs 96
4.2.2 Résultats en termes de FER et de complexité 97
4.3 BP-RNNs entraînés avec une fonction de coût focale 102
4.3.1 Motivations et principe 103
4.3.2 Résultats en termes FER 104
4.4 Amélioration du post-traitement OSD du BP 106
4.4.1 Motivations et solutions proposées 106
4.4.2 LLR accumulé et optimisé pour le post-traitement OSD 108
4.4.3 Post-traitement OSD multiples 110
4.4.4 Résultats en termes de FER 111
4.5 Conclusion 118

Les résultats du chapitre précédent nous ont montré que l'architecture parallèle d'une diversité de BP modélisés par des réseaux de neurones récurrents (BP-RNNs) et spécialisés sur les ensembles absorbants améliore la performance de décodage des codes LDPC courts sans augmenter la latence de décodage maximale par rapport au BP. Toutefois, cette architecture parallèle ne permet pas de combler entièrement l'écart jusqu'à la performance du décodage par maximum de vraisemblance (ML). Si la contrainte de latence de décodage maximale peut être relâchée, nous proposons de chercher à nous rapprocher de la performance du décodage ML à l'aide d'un traitement supplémentaire.

Nous choisissons dans ce manuscrit l'approche de décodage par statistiques ordonnées (Ordered Statistics Decoding ou OSD en anglais) [28] en tant que traitement supplémentaire. Le décodeur OSD est en effet connu pour être capable d'estimer la performance du décodage ML. Il requiert néanmoins une complexité calculatoire élevée pour accomplir ceci. Afin de remédier à ce problème, l'OSD peut aussi être utilisé comme un outil de post-traitement, exploitant la sortie souple d'un décodeur itératif (tel que le BP) lorsque ce dernier ne converge pas vers un mot de code [29–32].

Le travail présenté dans ce chapitre étudie ainsi la concaténation d'un décodeur itératif basé sur le BP avec un post-traitement OSD. Il se divise en deux contributions principales. Dans la première, nous proposons une étape de post-traitement OSD afin d'exploiter la diversité de décodage apportée par l'utilisation de plusieurs BP-RNNs spécialisés sur les ensembles absorbants tels que présentés dans le chapitre 3. Notre seconde contribution se concentre, quant à elle, sur la concaténation du décodeur BP conventionnel avec le post-traitement OSD. En plus d'améliorer la performance de cette combinaison, notre objectif est aussi de fournir un compromis complexité/performance alternatif à celui donné par la diversité de BP-RNNs associée avec le post-traitement OSD. Pour réaliser ceci, nous proposons une nouvelle entrée optimisée pour le post-traitement OSD, ainsi qu'une stratégie de décodage pour un post-traitement OSD multiples.

4.1 OSD

Cette section est dédiée à l'introduction de l'OSD et des différentes approches de l'état de l'art le combinant avec le décodeur BP. De plus, il est montré que le calcul des mots de code candidats lors du décodage par OSD tend à engendrer une forte complexité calculatoire, en particulier pour des codes LDPC de longueur moyenne ou grande. Nous proposons en conséquence une méthode limitant le nombre de mots de code candidats calculés par l'OSD.

4.1.1 Principe de l'OSD

L'OSD a été introduit pour la première fois dans [28], comme une méthode de décodage capable de s'approcher de la performance ML pour des codes linéaires de longueur moyenne, avec une complexité polynomiale. Il peut être utilisé comme un décodeur à part entière,

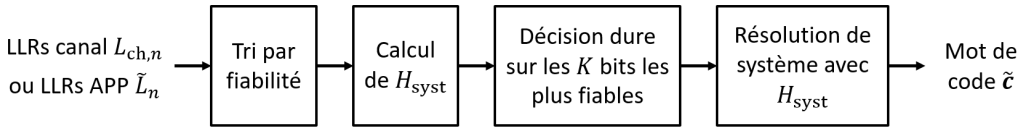


Figure 4.1 – OSD-0.

exploitant la sortie souple du canal ($\mathbf{L}_{\mathrm{ch}}$), ou comme une étape de décodage supplémentaire, exploitant les sorties souples ($\tilde{L}_n$) d'un décodeur à décision souple tel que le BP.

Dans l'OSD, un ensemble de K bits qualifiés de plus fiables est tout d'abord calculé à partir de la sortie souple correspondante. Les décisions dures sur ces K bits sont ensuite prises, puis utilisées pour retrouver les $N-K$ autres bits moins fiables. Les étapes du décodage OSD sont illustrées plus en détail dans la figure 4.1. Pour un vecteur de LLRs $\mathbf{L} = [L_1, \ldots, L_N]$ soit canal, soit a posteriori, la fiabilité d'un bit n est donnée par la valeur absolue de son LLR associé, $|L_n|$ ($n \in [1, N]$). Le décodage OSD commence par trier de manière décroissante les noeuds de donnée selon leur fiabilité. Le vecteur résultant est défini par $\mathbf{L}_{\mathrm{tri}} := \left[L_{\mathrm{tri},1}, \ldots, L_{\mathrm{tri},N}\right]$, avec $|L_{\mathrm{tri},1}| > \cdots > |L_{\mathrm{tri},N}|$. Les K premières composantes de $\mathbf{L}_{\mathrm{tri}}$ correspondent aux K bits dont les LLRs associés sont les plus grands en valeurs absolues. Ces K bits sont donc les bits les plus fiables de la séquence reçue. La matrice de parité H du code est ensuite mise sous une forme systématique, notée H_{syst}, avec la méthode du pivot de Gauss. Par simplicité, nous supposons que H est de rang $M = N - K$. Nous avons alors $H_{\mathrm{syst}} = [A \mid I]$, avec A une matrice de dimensions $M \times K$ et I la matrice identité de dimensions $M \times M$. La mise sous forme systématique de H se fait de manière à ce que les K colonnes de A correspondent aux K noeuds de donnée les plus fiables, en prenant en compte que des échanges de colonnes peuvent être nécessaires si les M colonnes les moins fiables de H ne sont pas linéairement indépendantes. Nous nous référons simplement aux nœuds de donnée correspondant aux colonnes de A comme étant les plus fiables et aux noeuds de donnée restants comme étant les moins fiables. Par souci de simplicité, nous notons de nouveau par $\mathbf{L}_{\mathrm{tri}}$ le vecteur de LLRs ordonnés par fiabilité après des échanges éventuels des colonnes de H. La décision dure est ensuite prise sur les K noeuds de donnée les plus fiables :

$$c_{\mathrm{tri},n} = \begin{cases} 0 & \text{si } L_{\mathrm{tri},n} > 0 \\ 1 & \text{si } L_{\mathrm{tri},n} \leq 0 \end{cases} \quad \text{avec } 1 \leq n \leq K \tag{4.1}$$

Les K premières composantes du vecteur $\mathbf{c}_{\mathrm{tri}} := \left[c_{\mathrm{tri},1}, \ldots, c_{\mathrm{tri},N}\right]$ sont par conséquent connus et correspondent aux noeuds de donnée les plus fiables. Les $N-K$ dernières composantes de $\mathbf{c}_{\mathrm{tri}}$ (bits les moins fiables) sont alors déterminés en résolvant le système linéaire donné par $H_{\mathrm{syst}}\mathbf{c}_{\mathrm{tri}}^T = \mathbf{0}_N$. Enfin, le mot de code estimé $\tilde{\mathbf{c}}$ est obtenu en appliquant la permutation inverse du tri par fiabilité sur $\mathbf{c}_{\mathrm{tri}}$.

Il convient de préciser que le processus décrit ci-dessus est en réalité appelé OSD d'ordre $p = 0$ (ou OSD-0) et réussit uniquement si les bits les plus fiables sont exempts d'erreur. Pour traiter les cas où les noeuds de donnée sélectionnés contiennent des erreurs, l'OSD-p ($p > 0$) considère toutes les combinaisons possibles d'au plus p erreurs parmi les noeuds de donnée

Algorithme 3 OSD-p.

Entrée : $\mathbf{L} = \mathbf{L}_{\text{ch}}$ **(ou** $\mathbf{L} = \tilde{\mathbf{L}}$**),** p
$\mathcal{S} = \emptyset$ // Initialisation de $\mathcal{S}$
$\mathbf{L}_{\text{tri}} \leftarrow$ Tri par fiabilité de $\mathbf{L}$
Calcul de H_{syst}
Mise à jour de $\mathbf{L}_{\text{tri}}$ (si échanges de colonnes)
$\mathbf{c}_{\text{tri}} \leftarrow$ Décision dure sur les K premiers éléments de $\mathbf{L}_{\text{tri}}$ et création de variables inconnues pour les autres
pour $q = 0$ à p **faire**
  Calcul des $\binom{K}{q}$ combinaisons $\{n_1, \ldots, n_q\}$
  pour chaque combinaison $\{n_1, \ldots, n_q\}$ **faire**
    Inversion des bits de $\{n_1, \ldots, n_q\}$ dans $\mathbf{c}_{\text{tri}}$
    Résolution du système $H_{\text{syst}}\mathbf{c}_{\text{tri}}^T = \mathbf{0}_N$
    Ajout du mot de code $\tilde{\mathbf{c}}$ correspondant à $\mathcal{S}$
  fin pour
fin pour
Retourner $\mathcal{S}$

les plus fiables. Pour chaque combinaison, la décision dure initiale des noeuds de donnée correspondants dans $\mathbf{c}_{\text{tri}}$ est inversée et les bits les moins fiables sont déterminés à nouveau en résolvant le système linéaire donné par $H_{\text{syst}}\mathbf{c}_{\text{tri}}^T = \mathbf{0}_N$. Cette procédure produit un ensemble $\mathcal{S}$ de $\sum_{q=0}^{p}\binom{K}{q}$ mots de code, à partir duquel une règle ML telle que (2.5) est utilisée pour sélectionner le mot de code le plus probable. L'algorithme 3 donne le pseudo-code résumant le processus de l'OSD-p ($p \geq 0$). Enfin, précisons que l'OSD-p peut approcher de près la performance du décodage ML, en supposant que la valeur p est suffisamment grande [28].

4.1.2 *OSD intégré dans le BP*

Afin d'améliorer la capacité de correction d'erreurs des codes linéaires courts, le BP peut être associé à l'OSD. Une première stratégie de combinaison, introduite par les auteurs [29] et employée dans [71], consiste à appliquer un OSD d'ordre faible ($p \leq 2$) entre chaque itération du BP. Plus précisément, nous considérons le vecteur de LLRs a posteriori $\tilde{\mathbf{L}}^{(i)}$, calculé à l'itération $i \in [1, I_{\text{test}}]$ du BP pour un vecteur de LLRs canal $\mathbf{L}_{\text{ch}}$ donnée. Un OSD-p est ensuite appliqué avec $\tilde{\mathbf{L}}^{(i)}$ comme entrée et un mot de code $\tilde{\mathbf{c}}^{(i)}$ est obtenu. Une autre itération du BP est alors réalisée et le processus ci-dessus est répété. La règle ML (2.5) entre les $\tilde{\mathbf{c}}^{(i)}$, $i \in [1, I_{\text{test}}]$, est alors appliquée pour obtenir le mot de code final $\tilde{\mathbf{c}}$.

Précisons que la version la plus simple de ce procédé itératif consiste à réaliser I_{test} itérations de BP et donc à appliquer I_{test} OSDs. Cette première version de l'OSD intégré dans le BP permet donc d'explorer une grande variété de mots de code candidats, chacun dépendant d'une entrée $\tilde{\mathbf{L}}^{(i)}$ différente. Cependant, l'OSD étant effectué I_{test} fois, une com-

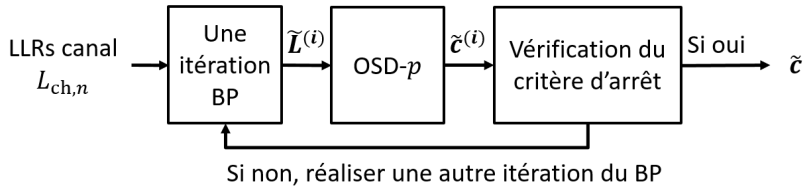


Figure 4.2 – OSD intégré dans le BP.

plexité calculatoire importante est également engendrée. Un critère d'arrêt, comme illustré à la figure 4.2, a donc été mis en place dans [29] afin de réduire la complexité (et donc le nombre d'OSD effectués). Son principe est le suivant. Si le mot de code le plus probable $\tilde{\mathbf{c}}^{(i)}$ trouvé à une itération donnée i est également obtenu au cours de u itérations consécutives, alors le décodage s'arrête et $\tilde{\mathbf{c}} = \tilde{\mathbf{c}}^{(i)}$. Ce test repose sur le fait que lorsque la quasi-convergence n'est pas atteinte par le BP, les valeurs de fiabilité des bits varient suffisamment au cours des itérations au point de fournir différents ensembles de noeuds de donnée les plus fiables pour l'OSD.

4.1.3 Post-traitement OSD du BP

Bien que les performances soient effectivement améliorées avec l'OSD intégré dans le BP, l'utilisation de l'OSD à chaque itération de décodage augmente la complexité de manière très significative. Pour limiter cette complexité, l'OSD peut aussi être utilisé comme une étape de post-traitement, exploitant la sortie souple du BP uniquement lorsque ce dernier n'arrive pas à converger vers un mot de code [29, Section VI.A]. La figure 4.3 détaille les étapes de ce processus. La première étape consiste à réaliser un décodage BP. Si pour une itération donnée $i \in [1, I_{\text{test}}]$ le BP estime un mot $\tilde{\mathbf{c}}^{(i)}$ vérifiant le syndrome, le décodage s'arrête. La sortie du décodage est alors $\tilde{\mathbf{c}} = \tilde{\mathbf{c}}^{(i)}$ et le post-traitement OSD n'est pas nécessaire. Si le BP ne converge pas vers un mot de code après I_{test} itérations de décodage, la fiabilité des bits est calculée à partir des LLRs a posteriori du BP, puis un décodage OSD-p est effectué pour obtenir le mot de code le plus probable $\tilde{\mathbf{c}}$ correspondant.

Il convient de remarquer que la fiabilité des bits est traditionnellement déterminée avec $\tilde{\mathbf{L}}^{(I_{\text{test}})}$, le vecteur de LLRs a posteriori calculé par le BP à la dernière itération de décodage. Cependant, il a été montré dans [72] qu'en cas d'échec du BP, les valeurs des LLRs a-posteriori de certains bits ont tendance à osciller et à changer régulièrement de signe tout au long du décodage. Ce phénomène, appelé phénomène d'oscillation, est notamment dû aux cycles et aux ensembles absorbants, qui sont enclins à désorienter le décodeur BP au fur et à mesure des itérations. La figure 4.4 illustre le phénomène d'oscillation, en affichant la valeur de $\tilde{\mathbf{L}}^{(i)}$ en fonction de $i \in [0, I_{\text{test}}]$ [1] pour deux bits donnés du Code-2. Précisons que pour obtenir

1. Par souci de simplicité dans les notations, $\tilde{\mathbf{L}}^{(i=0)} := \mathbf{L}_{\text{ch}}$.

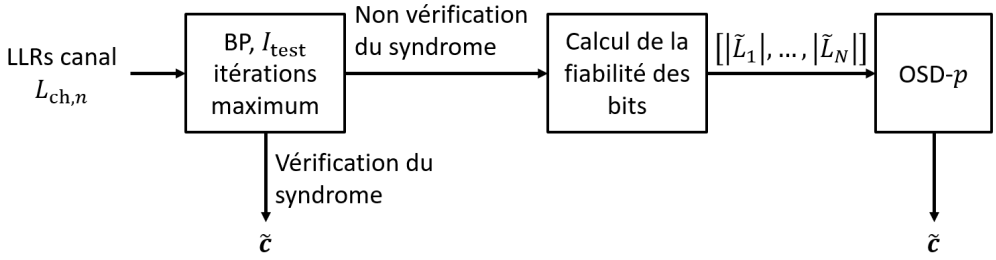


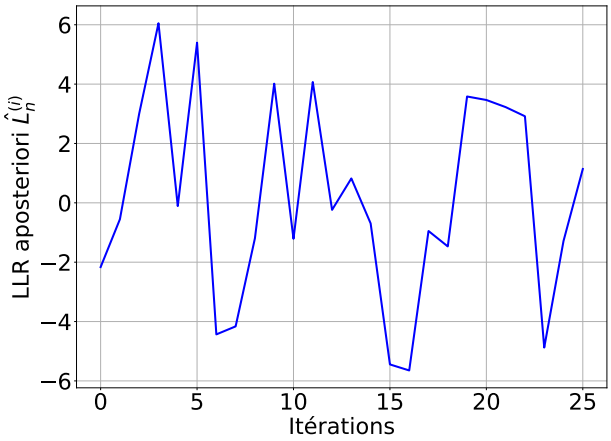
Figure 4.3 – Post-traitement OSD du BP, avec la fiabilité d'un bit donnée par la valeur absolue de son LLR a posteriori à la dernière itération de décodage du BP.

ces courbes, le BP a été évalué sur un mot bruité à 3.5 dB pour $I_{\mathrm{test}} = 25$ et que dans cet exemple le BP n'a pas convergé vers un mot de code. $\tilde{\mathbf{L}}^{(I_{\mathrm{test}})}$ n'induit donc pas forcément la mesure de fiabilité la plus appropriée pour le post-traitement OSD à cause du phénomène d'oscillation.

Pour améliorer la performance du post-traitement OSD, plusieurs études proposent de déterminer une fiabilité plus adaptée pour le post-traitement OSD que $\tilde{\mathbf{L}}^{(I_{\mathrm{test}})}$. Dans [30], il est préconisé d'employer directement les LLRs canaux $L_{\mathrm{ch},n}$ en tant que mesure de la fiabilité des bits lorsque le BP ne converge pas vers un mot de code. L'entrée de l'OSD est alors indépendante des LLRs a posteriori déterminés au cours des itérations du BP. Les auteurs de [31] suggèrent quant à eux une mesure de fiabilité pour des codes LDPC raccourcis, en considérant que certains bits d'information ont une valeur fixée et connue du récepteur. Les bits fixés sont notamment choisis afin d'exclure un ensemble de mots de code possédant un poids de Hamming faible. Une nouvelle mesure de fiabilité des noeuds de donnée pour le post-traitement OSD est également introduite dans [32]. Elle se base sur l'accumulation des LLRs a posteriori au cours des itérations du BP. En effet, si le BP n'obtient pas un mot de code après un nombre d'itérations maximal I_{test}, la fiabilité d'un noeud de donnée n est calculée à partir du LLR accumulé suivant :

$$\tilde{L}_n^{(\mathrm{S})} = \sum_{i=0}^{I_{\mathrm{test}}} \tilde{L}_n^{(i)}, n \in [1, N] \tag{4.2}$$

Intuitivement, la mesure de fiabilité résultant de l'équation (4.2) permet d'obtenir une fiabilité plus forte sur les noeuds de donnée dont le signe des $\tilde{L}_n^{(i)}$ restent le même au cours des itérations du BP, que sur ceux dont le signe des $\tilde{L}_n^{(i)}$ changent. Le vecteur $\tilde{\mathbf{L}}^{(\mathrm{S})} := \left[\tilde{L}_1^{(\mathrm{S})}, \ldots, \tilde{L}_N^{(\mathrm{S})}\right]$ assure par conséquent que les noeuds de donnée les plus fiables et fixés par l'OSD correspondent à des bits dont le BP est confiant (sans phénomène d'oscillation). La fiabilité calculée $\tilde{\mathbf{L}}^{(\mathrm{S})}$ diminue donc la probabilité de sélectionner des bits impliqués dans des cycles ou des motifs d'erreurs avec un support d'ensemble absorbant parmi les K bits les plus fiables. Autrement dit, $\tilde{\mathbf{L}}^{(\mathrm{S})}$ donne une mesure de fiabilité pour le post-traitement OSD qui potentiellement prend en compte l'impact des cycles et des ensembles absorbants lors du décodage BP.



(a) Bit 77.

(b) Bit 28.

Figure 4.4 – Exemples typiques d'un phénomène d'oscillation (Code-2, 25 itérations du BP à 3.5 dB, non convergence vers un mot de code).

4.1.4 Réduction de complexité

L'une des principales limitations pratiques de l'OSD est sa complexité [73], puisqu'une liste de $\sum_{q=0}^{p} \binom{K}{q}$ mots de code candidats doit être calculée à chaque utilisation d'un OSD-p. La complexité calculatoire du post-traitement OSD tend donc à être particulièrement coûteuse pour les codes LDPC moyens ou longs avec $K > 100$ et pour un OSD-p avec $p \geq 2$. Pour limiter cette complexité, le nombre de mots de code candidats est généralement limité, en considérant uniquement certaines inversions parmi les noeuds de donnée les plus fiables. À ce titre, une procédure pour calculer une liste de positions à inverser pendant

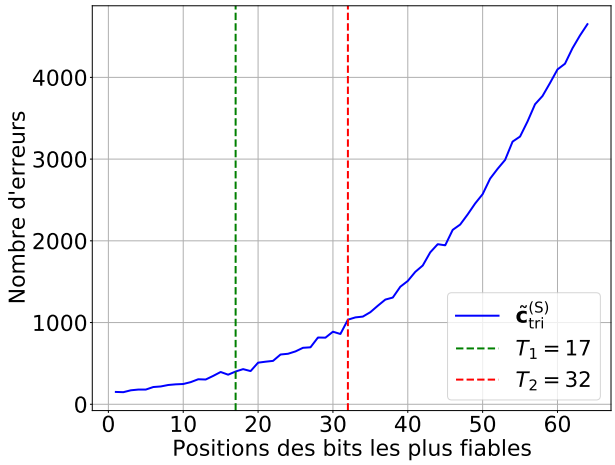


Figure 4.5 – Nombre d'erreurs vs positions dans les K premiers éléments de $\tilde{\mathbf{c}}_{\text{tri}}^{(S)}$ (Code-2, BP($I_{\text{test}} = 25$), 3.5 dB, 10000 échecs de décodage).

l'OSD-p a été introduite dans [73]. Cette liste est déterminée grâce au calcul des probabilités d'erreurs conjointes des bits les plus fiables et permet de traiter uniquement les mots de code candidats les plus probables lors de l'OSD. Dans [74] et [75], les noeuds de donnée les plus fiables sont partitionnés en segments selon des seuils de fiabilité dépendant du mot de code bruité reçu. Seuls les bits de certains segments sont ensuite sélectionnés pour l'inversion. Il convient de mentionner que les segments sélectionnés contiennent généralement les noeuds de donnée avec les fiabilités les plus faibles parmi les K plus fiables. Les deux méthodes décrites précédemment peuvent exiger un coût de calcul élevé, en raison des calculs des probabilités et des seuils. Les auteurs de [76] suggèrent pour chaque mot de code bruité d'inverser seulement les bits les plus fiables pour lesquels les fiabilités correspondantes ne respectent pas un seuil dépendant du niveau de bruit du canal. Si les K bits les plus fiables d'un mot bruité respectent tous ce seuil, aucune inversion n'est effectuée. Les éventuelles erreurs présentent dans ces bits ne seront donc pas traitées. En revanche, si pour un mot bruité $V \leq K$ bits parmi les plus fiables ne respectent pas ce seuil, $\sum_{q=0}^{p} \binom{V}{q}$ mots de code candidats sont calculés. La complexité calculatoire induite par l'OSD diffère donc entre chaque mot bruité.

Nous proposons dans cette sous-section une stratégie pour réduire la complexité calculatoire, indépendante du mot de code bruité reçu. Plus précisément, nous limitons tout d'abord la complexité de décodage en restreignant notre étude à l'OSD-2. Nous réduisons ensuite la complexité de l'OSD-2 en traitant uniquement les motifs d'au plus 2 erreurs avec les plus fortes probabilités d'occurrence. À cette fin, nous introduisons deux seuils de position, T_1 et T_2, avec $T_1 < T_2$. Pour un code linéaire, nous ne considérons alors dans $\mathbf{c}_{\text{tri}}$ que jusqu'à deux inversions dans le segment $[T_1, K]$, avec au moins une inversion dans le segment $[T_2, K]$ dans le cas de deux inversions exactement. Le schéma 4.6 illustre ces inversions autorisées

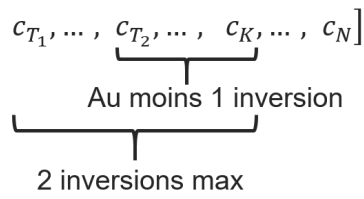


Figure 4.6 – Inversions autorisées dans $\mathbf{c}_{\text{tri}}$ pour l'OSD-2.

dans $\mathbf{c}_{\text{tri}}$. Notons que les bits ayant une position dans le segment $[1, T_1 - 1]$ correspondent aux bits avec les fiabilités les plus hautes et donc avec les plus faibles probabilités d'erreurs. De plus, un couple d'erreurs dans $[T_1, T_2 - 1]^2$ est moins probable qu'un couple d'erreurs dans $[T_2, K]^2$ ou dans $[T_2, K] \times [T_1, K]$. Pour illustrer ceci, nous avons relevé pour le Code-2 10000 mots bruités pour lesquels le BP($I_{\text{test}} = 25$) ne converge pas vers un mot de code à 3.5 dB. La figure 4.5 montre le nombre d'erreurs obtenues en fonction de la position dans les K premiers éléments de $\tilde{\mathbf{c}}_{\text{tri}}^{(S)}$, le vecteur de décisions dures associé à $\tilde{\mathbf{L}}_{\text{tri}}^{(S)}$. Nous constatons que le nombre d'erreurs croît lorsque la position dans les K premiers éléments de $\tilde{\mathbf{c}}_{\text{tri}}^{(S)}$ est élevée. Par exemple, 85% des erreurs sont contenues dans $[T_2 = 32, K = 64]$, tandis que 95% des erreurs sont comprises dans $[T_1 = 17, K = 64]$. Par conséquent, la méthode proposée limite bien le nombre de mots de code candidats pour l'OSD-2 en ne traitant que les motifs d'erreurs ayant la plus grande probabilité d'occurrence. Le nombre de mots de code candidats N_C est obtenu avec la formule suivante :

$$N_C = 1 + \binom{K - T_1}{1} + \binom{K - T_2}{2} + (T_2 - T_1)(K - T_2) \tag{4.3}$$

Le mot de code déterminé sans inversion correspond au 1 dans le calcul de N_C. Le terme $\binom{K-T_1}{1}$ provient quant à lui au nombre de mots de code obtenus en inversant uniquement un bit dans $[T_1, K]$. Enfin, le terme $\binom{K-T_2}{2}$ est le nombre de mots de code candidats lorsque 2 inversions dans $[T_2, K]^2$ sont considérées, tandis que $(T_2 - T_1)(K - T_2)$ correspond au nombre de mots de code candidats calculés avec 1 inversion dans $[T_1, T_2 - 1]$ et 1 inversion dans $[T_2, K]$. Pour $(T_1, T_2) = (17, 32)$, $N_C = 1024$, ce qui correspond à 49.2% de la complexité de l'OSD-2 pour le Code-2 $\left(\sum_{q=0}^{2} \binom{64}{q} = 2081\right)$. La méthode proposée avec $(T_1, T_2) = (17, 32)$ divise donc par deux le nombre de mots de candidats par rapport à l'OSD-2, tout en traitant les motifs d'au plus deux erreurs avec les plus grandes probabilités d'occurrence. Nous évaluerons notamment l'impact de cette méthode de réduction de la complexité sur le taux d'erreur paquet du post-traitement OSD-2 du BP standard dans la section 4.4.

Enfin, il est important de remarquer que la complexité d'un OSD d'ordre faible ($p \leq 2$) est également dominée par l'étape d'élimination gaussienne, nécessaire pour calculer H_{syst}. Cependant, pour les codes LDPC, la faible densité de la matrice de parité peut être avantageusement exploitée pour réduire considérablement la complexité de cette étape. Par exemple, la procédure proposée de [77] pour résoudre des systèmes linéaires clairsemés a été

adaptée dans [78] pour obtenir une technique d'encodage efficace pour les codes LDPC, ainsi que dans [79] pour développer un algorithme de décodage ML efficace pour les codes LDPC sur les canaux à effacement. Enfin, le cas du canal à effacement se rapproche particulièrement de l'OSD, où les bits les moins fiables peuvent être vus comme étant effacés.

4.2 Post-traitement OSD d'une diversité de BP-RNNs spécialisés

Dans cette section, nous voulons combiner notre approche de diversité présentée dans le chapitre 3 avec une étape de post-traitement OSD. Plus explicitement, un post-traitement OSD est appliqué sur la sortie souple de chaque décodeur BP-RNN spécialisé d'une diversité $\mathcal{D}_Z$, uniquement si aucun des Z BP-RNNs sélectionnés ne fournit de mot de code. Nous espérons ainsi améliorer la capacité de correction d'erreurs en exploitant la diversité de décodage apportée par l'utilisation de plusieurs décodeurs BP-RNNs spécialisés.

4.2.1 Motivations et proposition d'un post-traitement OSD des BP-RNNs

Nous considérons une diversité de BP-RNNs $\mathcal{D}_Z$ composée de Z BP-RNNs spécialisés et sélectionnés par complémentarité avec les règles définies par (3.2) et (3.3). La diversité $\mathcal{D}_Z$ peut être organisée soit en une architecture série, soit en une architecture parallèle. Deux constats expliquent l'utilisation de l'étape de post-traitement OSD après une diversité de BP-RNNs. Nous nous attendons tout d'abord à ce que le post-traitement OSD puisse bénéficier de la diversité apportée par l'utilisation de plusieurs décodeurs BP-RNNs. En effet, une diversité de Z de BP-RNNs spécialisés et complémentaires augmente la probabilité qu'au moins un de ces décodeurs ait au plus p erreurs parmi les noeuds de donnée les plus fiables. De plus, les matrices de parité des codes LDPC courts sont généralement construites de manière à obtenir la plus faible densité possible. C'est le cas par exemple lors d'une construction avec l'algorithme PEG. L'étape d'élimination gaussienne peut donc également profiter d'une réduction de complexité, même en taille courte.

Nous proposons ainsi d'appliquer un post-traitement OSD après chaque BP-RNN spécialisé de l'architecture de décodage d'une diversité $\mathcal{D}_Z$ uniquement lorsqu'aucun des BP-RNNs ne converge vers un mot de code. La fiabilité du post-traitement OSD d'un BP-RNN D_z ($z \in [1, Z]$) est calculée dans un premier temps à partir de $\tilde{\mathbf{L}}_z^{(I_{\text{test}})}$, le vecteur des LLRs a posteriori à la dernière itération de décodage de D_z. Le post-traitement OSD d'un seul BP-RNN est donc dans ce cas similaire à celui du BP proposé dans [29]. Afin de prendre en compte le phénomène d'oscillation, un vecteur de LLRs accumulés au cours des itérations de D_z et optimisés pour le post-traitement OSD sera également considéré comme mesure de fiabilité dans la section 4.4. En revanche, nous n'évaluerons pas la fiabilité donnée par les LLRs provenant du canal tel que proposé dans [30] car nous souhaitons ici exploiter la diversité liée aux différentes sorties souples de l'architecture de $\mathcal{D}_Z$. Enfin, comme Z post-traitements OSD sont effectués, nous utilisons une règle ML telle que définie dans (2.5) pour choisir le mot de code final $\tilde{\mathbf{c}}$ parmi les Z mots de code candidats. La figure 4.7 résume

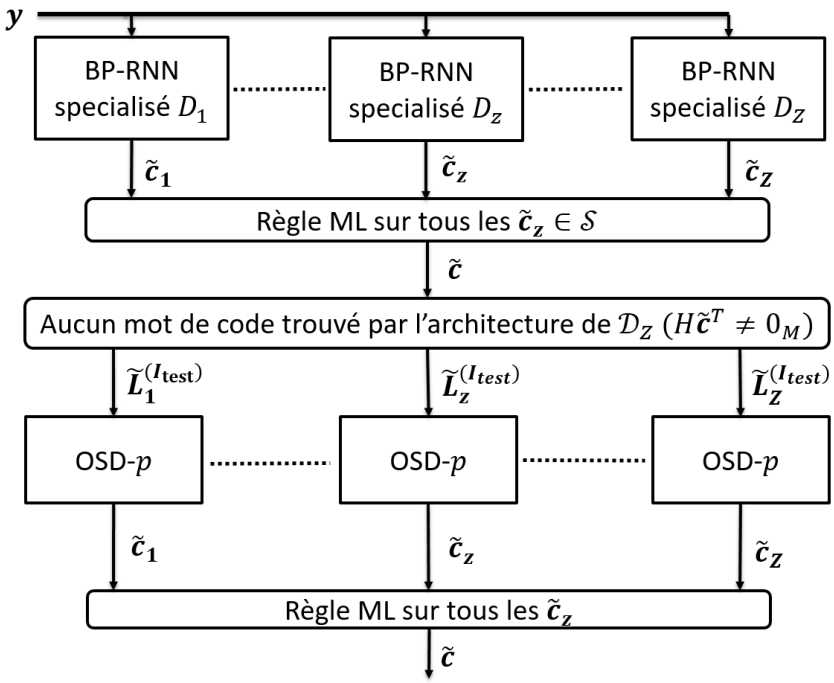


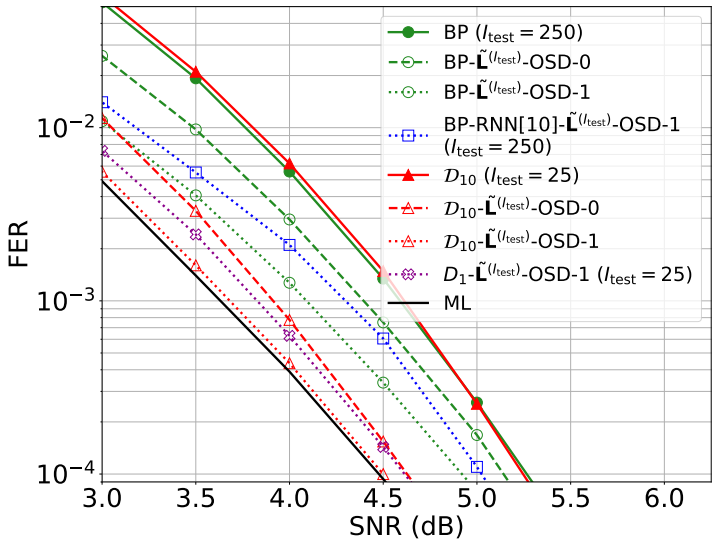
Figure 4.7 – Post-traitement OSD-p d'une diversité de BP-RNNs $\mathcal{D}_Z$ organisée en une archietcture de décodage parallèle.

la procédure de post-traitement OSD-p d'une diversité $\mathcal{D}_Z$ sous la forme d'un schéma bloc, dans le cas où la diversité $\mathcal{D}_Z$ est organisée en une architecture de décodage parallèle. L'entrée du post-traitement OSD-p d'un décodeur BP-RNN D_z est ici $\tilde{\mathrm{L}}_z^{(I_{\text{test}})}$.

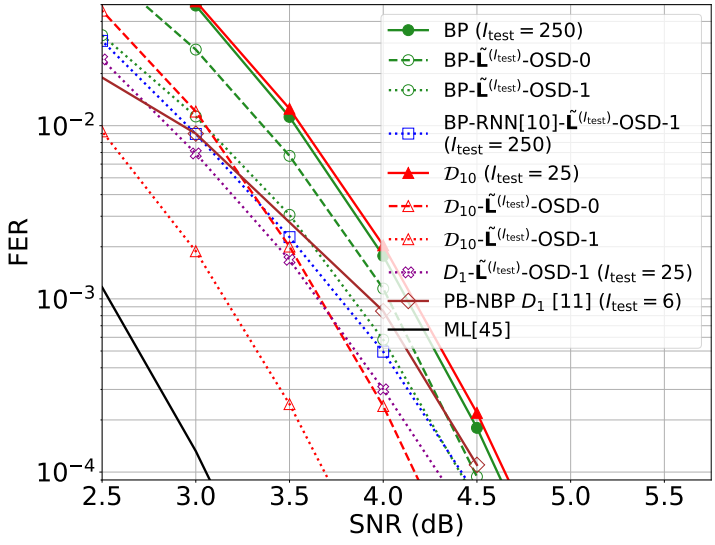
Il convient de rappeler que les résultats en termes de taux d'erreur paquet du chapitre précédent nous ont montré que l'architecture série et parallèle de $\mathcal{D}_Z$ obtiennent des performances similaires. Il s'ensuit que l'étape de post-traitement OSD offre des performances similaires lorsqu'elle est appliquée à la suite de l'architecture soit parallèle soit série. Nous notons donc simplement la stratégie de décodage proposée ici par $\mathcal{D}_Z$-$\tilde{\mathrm{L}}^{(I_{\text{test}})}$-OSD, sans indiquer l'architecture de décodage de la diversité.

4.2.2 *Résultats en termes de FER et de complexité*

Nous présentons dans cette sous-section les résultats en termes de taux d'erreur paquet (FER) obtenus avec la stratégie de post-traitement OSD d'une diversité de BP-RNNs $\mathcal{D}_Z$. Cette procédure de post-traitement est également évaluée avec une diversité de Min-Sum modélisés par des RNNs (MS-RNNs). Les paramètres utilisés pour les simulations sont les mêmes que ceux décrits en section 3.4. Les gains sont à nouveau relevés pour un taux d'erreur paquet de 10^{-4}.



(a) Code-1.



(b) Code-2.

Figure 4.8 – Résultats FER (BP-RNNs) avec l'étape de post-traitement OSD ($p = 0, 1$).

4.2.2.1 BP-RNNs et post-traitement OSD

Nous évaluons dans un premier temps le post-traitement OSD d'ordre faible ($p = 0, 1$) avec la diversité de BP-RNNs $\mathcal{D}_{10}(I_{\text{test}} = 25)$ du chapitre 3. Un post-traitement OSD avec comme mesure de fiabilité $\tilde{\mathbf{L}}^{(I_{\text{test}})}$ est également appliqué au BP($I_{\text{test}} = 250$), ainsi qu'au décodeur BP-RNN ($I_{\text{test}} = 250$) [10]. Les résultats de simulation sont présentés sur les

figures 4.8(a) et 4.8(b) pour le Code-1 ($N = 64, K = 32$) et le Code-2 ($N = 128, K = 64$) respectivement.

Pour le Code-1, nous constatons tout d'abord que le décodeur BP-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-1 fournit une meilleur performance que le décodeur non spécialisé BP-RNN-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-1. Il convient de rappeler que sur la figure 3.7(a), le BP-RNN [10] (sans OSD) possédait déjà une performance moins bonne que le BP pour $I_{\text{test}} = 250$. L'OSD-1 n'a donc pas permis de combler l'écart de performance entre le BP et le BP-RNN non spécialisé pour le Code-1. En revanche, l'utilisation du premier décodeur de notre diversité BP-RNN (D_1-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-1) présente un gain de 0.31 dB par rapport au décodeur BP-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-1 pour un FER = 10^{-4}. Nous en déduisons que les LLRs a posteriori de D_1 ($I_{\text{test}} = 25$) sont de meilleur qualité que ceux du BP($I_{\text{test}} = 250$) vis-à-vis de l'OSD. Si tous les BP-RNNs spécialisés sont pris en compte ($\mathcal{D}_{10}$-OSD-1), le gain s'élève à 0.43 dB par rapport au décodeur BP-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-1, ce qui revient à un gain supplémentaire de 0.12 dB par rapport au D_1-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-1. De plus, nous observons que $\mathcal{D}_{10}$-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-1 atteint pratiquement la performance du décodage ML. Enfin, un gain de 0.52 dB est relevé pour $\mathcal{D}_{10}$-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-0 par rapport au décodeur BP-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-0.

Pour le Code-2, nous remarquons que le décodeur non spécialisé BP-RNN-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-1 obtient une performance proche du décodeur BP-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-1. Rappelons que ces deux décodeurs possédaient déjà une performance proche sans OSD (cf figure 3.7(b)). Le vecteur de LLRs aposteriori $\tilde{\mathbf{L}}^{(I_{\text{test}})}$ du BP ou du BP-RNN [10] induit donc une mesure de fiabilité équivalente en termes de performance avec le post-traitement OSD-1 pour ce code. $\mathcal{D}_{10}$-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-$p$ apporte un gain de 0.32 dB (resp. 0.72 dB) pour $p = 0$ (resp. $p = 1$) par rapport au décodeur BP-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-$p$. En utilisant seulement le premier décodeur de notre diversité de BP-RNNs, nous observons que D_1-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-1 obtient un gain d'environ 0.11 dB par rapport au BP-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-1. Ceci confirme également que pour ce code, les LLRs a posteriori de D_1 à la dernière itération de décodage sont plus adaptés pour le post-traitement OSD-1 que ceux du BP ou du BP-RNN entraîné comme dans [10]. À des fins de comparaison, nous avons également inclus dans la figure 4.8(b) la performance du décodeur neuronal basé sur de l'élagage PB-NBP D_1 de [11]. Nous remarquons que $\mathcal{D}_{10}$-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-1 présente un gain de 0.84 dB par rapport à PB-NBP D_1. Enfin, $\mathcal{D}_{10}$-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-1 se situe à 0.63 dB de la performance du décodage ML [45].

Afin de réduire davantage l'écart avec la performance du décodage ML pour le Code-2, nous considérons également un OSD d'ordre $p = 2$ pour le post-traitement. Les résultats correspondants sont montrés sur la figure 4.9. Les performances avec un post-traitement OSD-0 ne sont plus affichées par souci d'encombrement. Nous observons que l'écart à la performance du décodage ML s'est abaissé à 0.2 dB lorsqu'un post-traitement d'ordre 2 est appliqué avec $\mathcal{D}_{10}$. De plus $\mathcal{D}_{10}$-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-2 possède un gain d'environ 0.97 dB par rapport au BP-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-2 ou au BP-RNN [10]-OSD-2. Ce gain se réduit à 0.2 dB si uniquement le premier décodeur de la diversité est concaténé avec un post-traitement OSD-2.

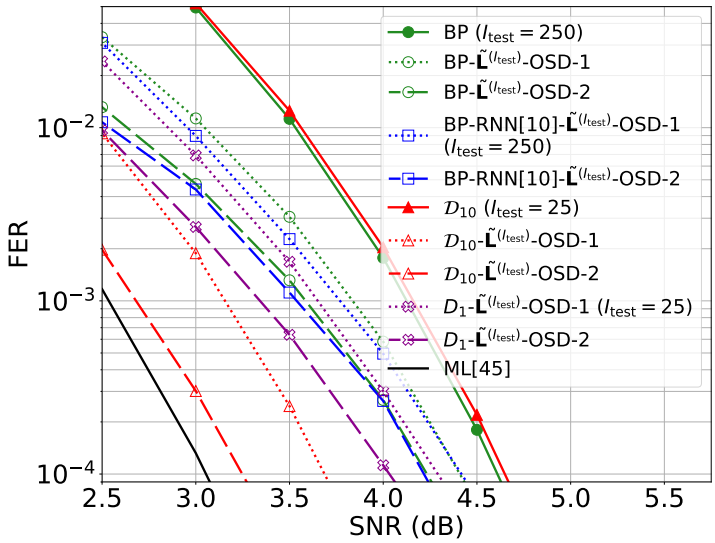


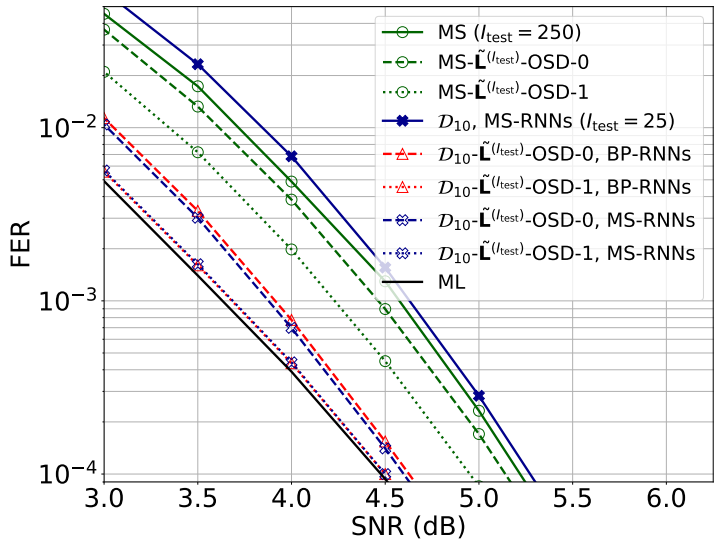
Figure 4.9 – Résultats FER (BP-RNNs) avec l'étape de post-traitement OSD pour le Code-2 ($p = 1, 2$).

4.2.2.2 MS-RNNs et post-traitement OSD

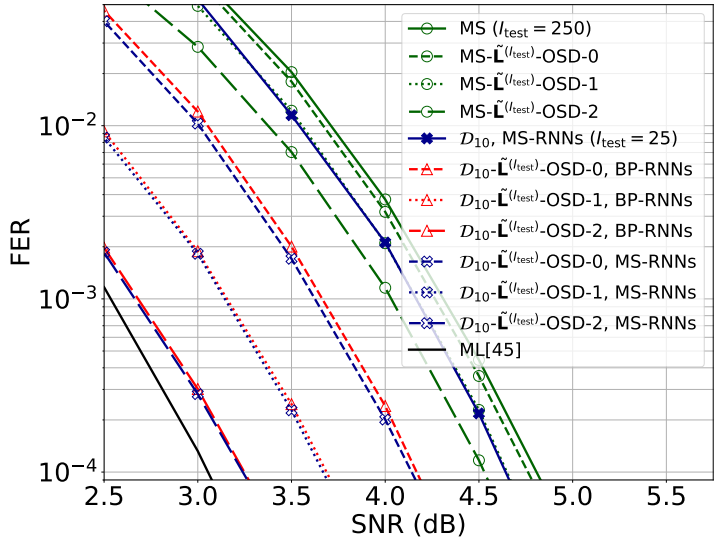
Nous comparons ici la performance de la stratégie de post-traitement OSD pour une diversité $\mathcal{D}_{10}$ composée soit de décodeurs BP-RNNs, soit de décodeurs MS-RNNs. Les BP-RNNs et les MS-RNNs sont spécialisés sur les mêmes classes d'ensembles absorbants données dans le tableau 3.3. De plus, un post-traitement OSD sur $\tilde{\mathbf{L}}^{(I_{\text{test}})}$ est appliqué après le décodeur MS($I_{\text{test}} = 250$) à titre de comparaison. Les résultats correspondants sont affichés dans les figures 4.10(a) pour le Code-1 et 4.10(b) le Code-2. Il convient de rappeler que nous avons montré dans le paragraphe 3.4.5.2, que les diversités de BP-RNNs et de MS-RNNs induisent des performances similaires pour les deux codes. Par conséquent, uniquement les performances de $\mathcal{D}_{10}$(MS-RNNs) sont affichées dans la figure 4.10. Enfin, pour le Code-1, le post-traitement OSD est d'ordre 0 ou 1 étant donné que $\mathcal{D}_{10}$(BP-RNNs)-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-1 atteint quasiment la performance du décodage ML. Pour le Code-2, nous considérons en plus un post-traitement OSD d'ordre $p = 2$ afin de se rapprocher de la performance du décodage ML.

Pour le Code-1, nous observons tout d'abord que la diversité de MS-RNNs spécialisés combinée avec l'OSD obtient des performances proches de son homologue composée de BP-RNNs spécialisés pour $p = 0, 1$. L'utilisation de MS-RNNs spécialisés au lieu de BP-RNNs spécialisés reste donc avantageux en termes de complexité lorsqu'une étape de post-traitement OSD est réalisée. Ensuite, comparé à MS-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-$p$, les gains apportés par $\mathcal{D}_{10}$(MS-RNNs)-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-$p$ sont de 0.54 dB pour $p = 0$ et de 0.45 dB pour $p = 1$.

Pour le Code-2, l'utilisation de MS-RNNs spécialisés au lieu de BP-RNNs spécialisés est aussi avantageuse en termes de complexité. En effet, $\mathcal{D}_{10}$(MS-RNNs)-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-$p$



(a) Code-1



(b) Code-2

Figure 4.10 – Résultats FER avec l'étape de post-traitement OSD, MS-RNNs spécialisés vs BP-RNNs spécialisés

possède une performance équivalente à $\mathcal{D}_{10}$(BP-RNNs)-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-$p$ pour $p = \{0, 1, 2\}$. Nous constatons également que le décodeur MS-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-1 atteint la même performance que $\mathcal{D}_{10}$(MS-RNNs). De plus, en comparant la performance de MS-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-$p$ avec le BP-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-$p$ de la figure 4.10(b), nous nous apercevons que le post-traitement OSD du

MS induit moins de gains qu'avec le BP pour les trois ordres d'OSD évalués. Enfin, $\mathcal{D}_{10}$(MS-RNNs)-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-$p$ obtient un gain croissant avec p par rapport à MS-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-$p$: 0.65 dB pour $p = 0$, 0.97 dB pour $p = 1$ et 1.28 dB pour $p = 2$. Augmenter l'ordre de l'OSD permet donc de mieux exploiter la diversité des MS-RNNs (et même des BP-RNNs) spécialisés pour le Code-2 (CCSDS).

4.2.2.3 Complexité calculatoire, matérielle et latence de décodage maximale

Nous analysons dans ce paragraphe la complexité calculatoire, matérielle et la latence de décodage maximale des différentes solutions proposées utilisant une étape de post-traitement OSD. Premièrement, nous remarquons que la latence de décodage maximale de l'architecture parallèle de $\mathcal{D}_{10}$ (BP-RNNs ou MS-RNNs) est de 25 itérations, tandis que celle du BP($I_{\text{test}} = 250$) (ou MS($I_{\text{test}} = 250$)) est égale à 250 itérations. Toutefois, à cause des multiplications liées aux poids des décodeurs neuronaux spécialisés, la complexité matérielle de $\mathcal{D}_{10}$-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD reste plus importante que celle du décodeur BP($I_{\text{test}} = 250$) avec un post-traitement OSD.

De plus, $\mathcal{D}_{10}$-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD requiert l'application de 10 OSD tandis que BP($I_{\text{test}} = 250$)-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD n'en nécessite qu'un seul. La complexité calculatoire de l'étape de post-traitement OSD de $\mathcal{D}_{10}$ (nombre de mots de code candidats calculés et d'inversions de matrices) est donc 10 fois supérieure à celle du décodeur BP($I_{\text{test}} = 250$). Au niveau de la complexité matérielle, il convient de mentionner que dans le cas de $\mathcal{D}_{10}$-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD, les 10 post-traitements OSDs peuvent être exécutés l'un après l'autre en utilisant une unique instanciation matérielle de l'algorithme OSD [2]. Ceci permet ainsi de limiter la complexité matérielle liée à l'utilisation de plusieurs OSDs lorsque la contrainte de latence de décodage maximale l'autorise.

Enfin, pour le Code-1, le décodeur D_1(BP-RNN)-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-1 permet de se rapprocher convenablement de la performance du décodage ML tout en présentant une complexité calculatoire limitée par rapport à $\mathcal{D}_{10}$-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD. En effet, un unique BP-RNN spécialisé est utilisé et un seul post-traitment OSD d'ordre 1 est appliqué, ce qui implique qu'une seule résolution de système est effectuée au lieu de 10. De plus, 33 mots de codes candidats sont testés lorsque D_1 ne converge pas vers un mot de code au lieu de 330 pour l'architecture de $\mathcal{D}_{10}$.

4.3 BP-RNNs entraînés avec une fonction de coût focale

Les résultats de la section précédente nous ont montré que combiner une étape de post-traitement OSD avec la diversité apportée par l'utilisation de plusieurs décodeurs BP-RNNs (ou MS-RNNs) spécialisés fournit un moyen efficace de combler l'écart avec le décodeur ML. Toutefois, il convient de préciser que les décodeurs BP-RNNs n'ont pas été entraînés spécifiquement pour fournir une entrée adaptée au post-traitement OSD. En effet, l'optimisation des poids du BP-RNN avec l'entropie croisée binaire (BCE) minimise son taux

2. Nous utilisons ici le même principe que pour l'implémentation matérielle de l'architecture série d'une diversité $\mathcal{D}_Z$.

d'erreur binaire (BER) seul, sans considérer sa concaténation avec un post-traitement OSD. Nous nous posons ainsi la question de modifier la fonction de coût afin d'adapter la sortie du BP-RNN pour le post-traitement OSD.

4.3.1 Motivations et principe

Considérons un BP-RNN combiné avec un post-traitement OSD-p. Si le BP-RNN ne converge pas vers un mot de code, l'OSD-p échoue notamment si plus de p bits erronés sont sélectionnés parmi les K bits les plus fiables [3]. Les K noeuds de donnée les plus fiables sont ceux détenant les décisions souples les plus grandes en valeur absolue. Les bits erronés parmi les K bits les plus fiables correspondent donc à des LLRs a posteriori du BP-RNN de signe incorrect et de large amplitude. Pour augmenter la capacité de correction du BP-RNN-OSD-p, il convient ainsi d'effectuer une optimisation des poids avec une fonction de coût amenant à réduire le nombre de LLRs de signe incorrect et de large amplitude en sortie du BP-RNN.

Nous proposons d'employer la fonction de coût focale [52] (Focal Loss ou FL en anglais) pour entraîner le BP-RNN. Son expression générale est donnée par l'équation (1.18) et s'adapte dans le cas de notre travail comme suit :

$$f_{\mathrm{FL}}(\mathbf{c}, \tilde{\mathbf{L}}) = -\frac{1}{N}\sum_{n=1}^{N} c_n \sigma\left(\tilde{L}_n\right)^{\gamma} \log\left(1-\sigma\left(\tilde{L}_n\right)\right) + \left(1-c_n\right)\left(1-\sigma\left(\tilde{L}_n\right)\right)^{\gamma} \log\left(\sigma\left(\tilde{L}_n\right)\right) \quad (4.4)$$

où $\tilde{\mathbf{L}}$ est le vecteur de LLRs a posteriori à la dernière itération de décodage du BP-RNN et γ est l'exposant focal (hyper-paramètre ajustable). Pour $\gamma > 0$, la fonction de coût focale permet de concentrer l'entraînement sur les éléments les plus à durs classifier dans le jeu d'entraînement, en leur affectant des pénalités plus importantes. Pour expliquer cela, nous allons tout d'abord nous intéresser au cas du mot de code tout zéro. En supposant que $c_n = 0, n \in [1, N]$, (4.4) se simplifie en :

$$f_{\mathrm{FL}}(\tilde{\mathbf{L}}) = -\frac{1}{N}\sum_{n=1}^{N} \left(1-\sigma\left(\tilde{L}_n\right)\right)^{\gamma} \log\left(\sigma\left(\tilde{L}_n\right)\right) \quad (4.5)$$

Nous montrons dans la figure 4.11 l'allure de la fonction de coût focale (4.5) selon la valeur des LLRs a posteriori et pour plusieurs valeurs de γ. Précisons que sous l'hypothèse du mot de code tout zéro, les LLRs a posteriori de signe incorrect sont les LLRs négatifs. Tout d'abord, nous remarquons que la fonction BCE ($\gamma = 0$) pénalise bien des LLRs négatifs mais aussi des LLRs possédant une amplitude entre 0 et 3. Dans le cas d'un post-traitement OSD, les LLRs négatifs de faible amplitude ne sont pas forcément problématiques, puisqu'ils ont peu de chances d'être sélectionnés parmi les K LLRs les plus fiables. Les LLRs problématiques, pouvant causer un échec de l'OSD et donc à pénaliser, sont par conséquent les LLR négatifs à forte amplitude. Ceci montre bien que l'entropie croisée binaire n'est pas la fonction de

3. Si la régle ML ne détermine pas le mot de code attendu, l'OSD-p échoue aussi.

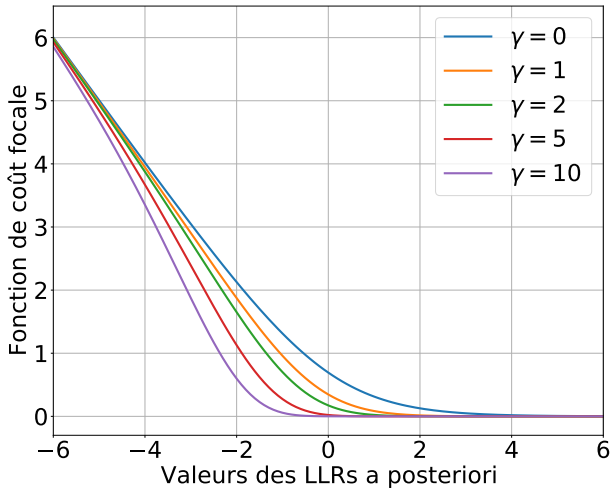
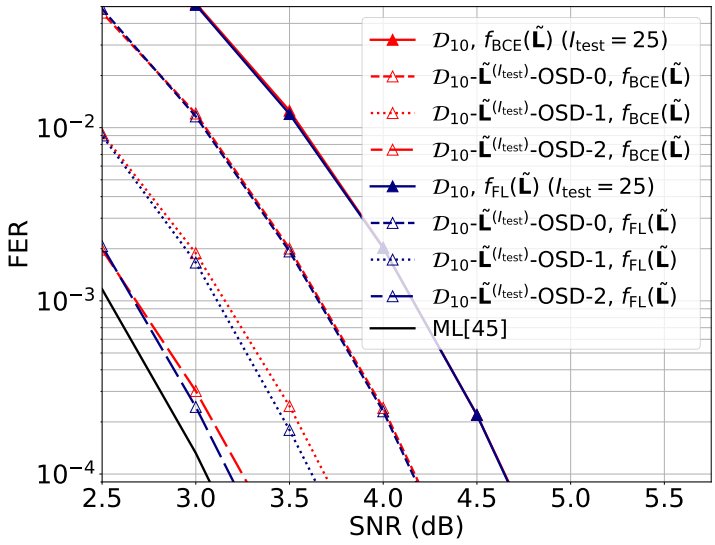


Figure 4.11 – Fonction de coût focale (4.5) en fonction de la valeur des LLRs a posteriori.

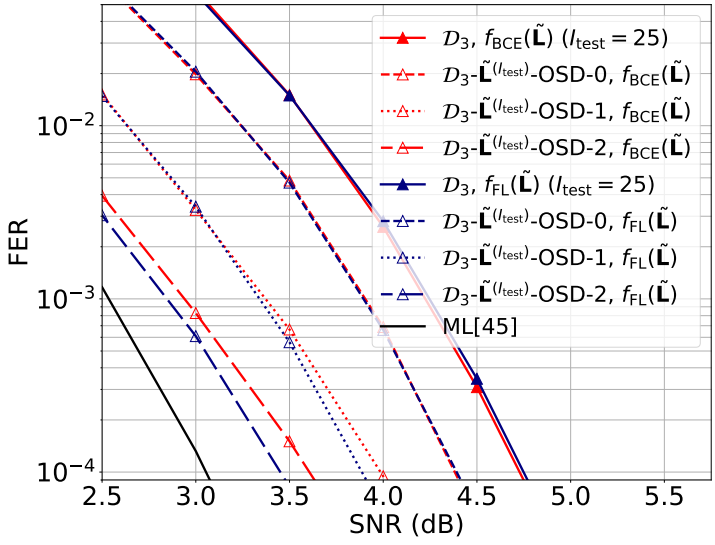
coût la plus appropriée pour le problème d'optimisation du BP-RNN combiné à l'OSD. Nous constatons que plus γ augmente, plus les LLRs négatifs pénalisés ont de large amplitude. Les LLRs de faible amplitude (négatifs ou positifs) sont quant à eux très peu pénalisés. γ détermine donc à partir de quelle amplitude les LLRs de signes négatifs sont pénalisés lors de l'entraînement du BP-RNN. Cette conclusion reste vraie pour le cas général. Il suffit en effet de considérer le symétrique de la courbe par rapport à 0 pour établir ceci avec les LLRs incorrects de signe positif. Nous proposons donc d'utiliser la fonction de coût focale pour entraîner le BP-RNN en vue du post-traitement OSD.

4.3.2 *Résultats en termes FER*

Nous comparons ici l'impact des fonctions de coût focale et BCE sur la performance en termes de FER d'une diversité de BP-RNNs spécialisés associée à un post-traitement OSD. Nous concentrons notre attention uniquement sur le Code-2, étant donné que pour le Code-1, le décodeur D_1-$\tilde{\mathrm{L}}^{(I_{\text{test}})}$-OSD-1 fournit une performance se situant à seulement 0.12 dB de la performance du décodage ML avec une complexité maîtrisée. Nous considérons ainsi la diversité $\mathcal{D}_{10}$ de BP-RNNs($I_{\text{test}} = 25$) entraînés avec la fonction de coût BCE du Code-2. La procédure de spécialisation des BP-RNNs sur les $J = 120$ classes d'ensembles absorbants est de nouveau appliquée en utilisant la fonction de coût focale (4.5) pour l'entraînement (avec $I_{\text{ent}} = 10$), puis une deuxième diversité $\mathcal{D}_{10}$ est obtenue avec la sélection sous-optimale par complémentarité. En mesurant le taux d'erreur paquet du post-traitement OSD-p avec les BP-RNNs spécialisés selon l'hyper-paramètre γ, nous avons déterminé empiriquement $\gamma = 10$ comme étant un bon choix pour les deux entraînements. Deux diversités $\mathcal{D}_{10}$ de BP-RNNs($I_{\text{test}} = 25$) spécialisés sont ainsi construites, chacune correspondant soit à la fonction



(a) $\mathcal{D}_{10}$.



(b) $\mathcal{D}_3$

Figure 4.12 – Résultats FER avec l'étape de post-traitement OSD et un entraînement des décodeurs BP-RNNs($I_{\text{test}} = 25$) avec la fonction de coût focale ($\gamma = 10$).

BCE soit à la fonction de coût focale. Les deux diversités $\mathcal{D}_3$ [4] sont également évaluées avec le post-traitement OSD afin d'étudier l'influence du nombre de décodeurs dans la diversité.

Les résultats en termes de FER avec le post-traitement OSD sont affichés pour $p \leq 2$

4. Il s'agit des trois premiers décodeurs BP-RNNs de $\mathcal{D}_{10}$.

dans la figure 4.12, avec $\mathcal{D}_{10}$ (a) et $\mathcal{D}_3$ (b). Nous remarquons tout d'abord que les diversités de BP-RNNs entraînés avec soit la fonction de coût BCE soit avec la fonction de coût focale présentent des performances similaires sans post-traitement OSD. Ceci reste vrai pour un post-traitement OSD d'ordre $p = 0$. Néanmoins, pour $p = \{1, 2\}$, l'entraînement des BP-RNNs spécialisés de $\mathcal{D}_{10}$ avec $f_{\text{FL}}(\tilde{\mathbf{L}})$ induit de légères améliorations par rapport à la fonction BCE. Les gains sont plus prononcés lorsque le nombre de décodeurs BP-RNNs est réduit. En effet, les BP-RNNs de $\mathcal{D}_3$ entraînés avec $f_{\text{FL}}(\tilde{\mathbf{L}})$ engendre un gain de 0.1 dB par rapport à ceux entraînés avec $f_{\text{BCE}}(\tilde{\mathbf{L}})$. Ce gain est accentué à 0.15 dB pour un ordre $p = 2$. Enfin, l'écart à la performance du décodage ML est réduit à 0.4 dB pour $\mathcal{D}_3$-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-$p$ si la fonction de coût utilisée est $f_{\text{FL}}(\tilde{\mathbf{L}})$.

Au vu de ces résultats, nous concluons que la fonction de coût focale appliquée sur $\tilde{\mathbf{L}}$ permet de mieux entraîner les BP-RNNs spécialisés pour post-traitement OSD que la fonction BCE. Elle fournit ainsi un vecteur $\tilde{\mathbf{L}}$ possédant en moyenne moins d'erreurs parmi les K bits les plus fiables que celui obtenu avec la fonction BCE. Enfin, précisons que la probabilité d'obtenir un vecteur de LLRs a posteriori contenant un nombre d'erreurs inférieur ou égal à l'ordre de l'OSD est d'autant plus grande que la diversité de BP-RNNs est grande et ceci quelque soit la fonction de coût considérée. L'exploitation de la diversité $\mathcal{D}_{10}$ par le post-traitement OSD recouvre donc une partie des gains engendrés par l'utilisation de la fonction de coût focale pour l'entraînement. $\mathcal{D}_3$ ne bénéficie pas autant de cet avantage, ce qui explique des gains plus visibles avec la fonction de coût focale.

4.4 Amélioration du post-traitement OSD du BP

Dans la sous-section 4.2, il a été montré que les décodeurs BP-RNNs spécialisés sont capables d'obtenir de meilleurs performances que le BP avec le post-traitement OSD. Néanmoins, utiliser le BP présente un avantage en termes de complexité matérielle, puisque les poids associés au BP-RNN induisent un coût supplémentaire en termes d'espace mémoire et complexité calculatoire. Par conséquent, améliorer la performance du post-traitement OSD du BP permettrait de fournir une solution alternative au post-traitement OSD d'une diversité de BP-RNNs spécialisés, possédant une complexité matérielle plus faible.

Nous investiguons ainsi dans cette section la combinaison du BP standard avec le post-traitement OSD. Pour ce faire, deux problématiques sont considérées. La première, similaire à celle de la section précédente, est de fournir l'entrée la plus adaptée possible au post-traitement OSD, afin d'obtenir une performance améliorée. La deuxième est de déterminer sur quelles informations souples du BP un OSD doit être appliqué lorsque l'utilisation de plusieurs OSD pour l'étape de post-traitement est considérée.

4.4.1 Motivations et solutions proposées

Notre approche pour créer une entrée adaptée au post-traitement OSD se base sur le vecteur $\tilde{\mathbf{L}}^{(\text{S})} := \sum_{i=0}^{I_{\text{test}}} \tilde{\mathbf{L}}^{(i)}$, introduite par [32]. Pour justifier ce choix, nous avons dans

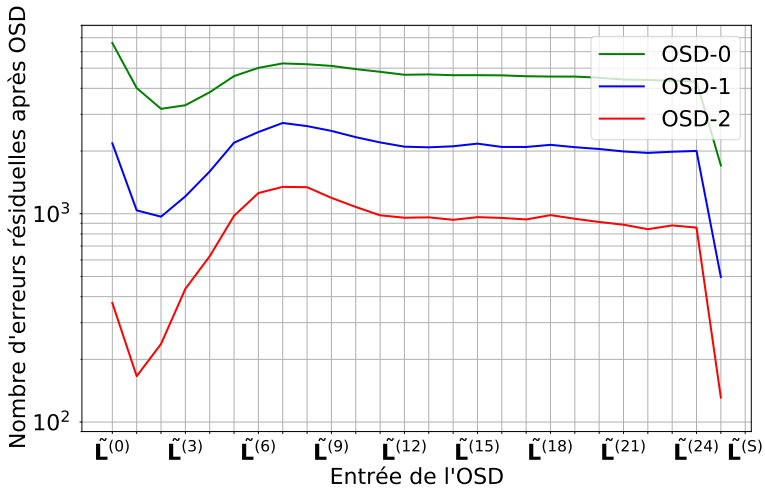


Figure 4.13 – Nombre d'erreurs résiduelles sur 10000 non convergences du BP vers un mot de code après post-traitement OSD (Code-2, $I_{\text{test}} = 25$, SNR = 3.5 dB).

un premier temps évalué la performance du post-traitement OSD-p du BP sur chaque $\tilde{\mathbf{L}}^{(i)} := \left[\tilde{L}_1^{(i)}, \ldots, \tilde{L}_N^{(i)}\right]$, $i \in [0, I_{\text{test}}]$ et sur $\tilde{\mathbf{L}}^{(\text{S})}$, pour $p \leq 2$. Le code considéré est le Code-2. Le nombre d'itérations maximal est fixé à $I_{\text{test}} = 25$, tandis que le SNR de test vaut 3.5 dB.

Pour 10000 non convergences du BP vers un mot de code, nous affichons sur la figure 4.13 le nombre d'erreurs résiduelles après le post-traitement OSD, en fonction des $\tilde{\mathbf{L}}^{(\text{i})}$ et de $\tilde{\mathbf{L}}^{(\text{S})}$. Il apparaît clairement que le post-traitement OSD-p avec la mesure de fiabilité $\tilde{\mathbf{L}}^{(\text{S})}$ obtient la meilleure performance et ceci quelque soit $p \in \{0, 1, 2\}$. Notons en particulier qu'utiliser $\tilde{\mathbf{L}}^{(0)}$ tel que suggéré dans [30] produit une moins bonne performance que $\tilde{\mathbf{L}}^{(\text{S})}$. Ces observations ont été entre autres vérifiées pour d'autres SNRs, ainsi que pour le Code-1.

La figure 4.13 montre également l'impact du phénomène d'oscillation sur la performance du post-traitement OSD. En effet, les LLRs a posteriori du BP durant les premières itérations ($i < 5$) induisent de meilleures performances avec l'OSD que les LLRs a posteriori calculés aux itérations plus tardives, pour lesquelles le décodeur BP est davantage désorienté.

En conclusion, $\tilde{\mathbf{L}}^{(\text{S})}$ constitue bien une mesure de fiabilité de meilleure qualité que les $\tilde{\mathbf{L}}^{(\text{i})}$ ($i \in [0, I_{\text{test}}]$), atténuant l'impact du phénomène d'oscillation. De plus, la mesure définie par $\tilde{\mathbf{L}}^{(\text{S})}$ ne dépend pas du type de code LDPC considéré, ce qui la rend plus flexible que la solution proposée dans [31].

Nous proposons ainsi d'utiliser une somme pondérée des LLRs a posteriori calculés aux différentes itérations de décodage (approche similaire à celle utilisée en [32]). Afin d'optimiser les poids de pondération, nous modéliserons la somme pondérée des LLRs à l'aide d'un seul neurone et utiliserons la fonction de coût focale, mieux adaptée au post-traitement OSD.

Concernant la seconde problématique, liée à l'utilisation de plusieurs décodeurs OSD, nous considérerons un décodeur OSD qui prend en entrée la somme pondérée des LLRs

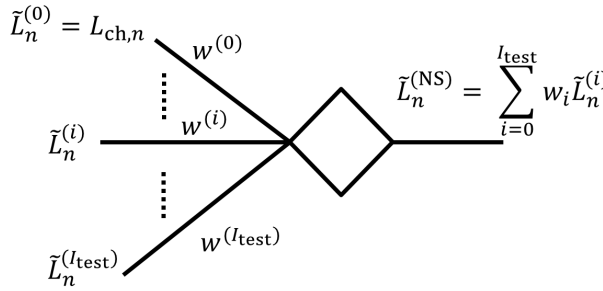


Figure 4.14 – Neurone calculant $\hat{L}_n^{(\mathrm{NS})}$

optimisée précédemment, complété par un nombre de décodeurs OSD appliqués sur des itérations du BP soigneusement choisies (autrement dit, un nombre de décodeurs OSD, prenant chacun en entrée les LLRs a postériori calculés à une certaine itération du décodage BP). La sélection de ces itérations se fera par une étude de complémentarité des décodeurs OSD appliqués à chaque itération du décodage BP.

4.4.2 LLR accumulé et optimisé pour le post-traitement OSD

4.4.2.1 Modélisation neuronale et jeu d'entraînement

Pour adapter davantage $\tilde{\mathbf{L}}^{(\mathrm{S})}$ à l'OSD, nous introduisons des poids $w^{(i)} \geq 0, i \in [0, I_{\mathrm{test}}]$, dans l'équation (4.2). Nous proposons ainsi de calculer la fiabilité d'un noeud de donnée n par le LLR accumulé des I_{test} itérations suivant :

$$\tilde{L}_n^{(\mathrm{NS})} = \sum_{i=0}^{I_{\mathrm{test}}} w^{(i)} \tilde{L}_n^{(i)}, n \in [1, N] \tag{4.6}$$

à la place de $\tilde{\mathbf{L}}_n^{(\mathrm{S})}$ de l'équation (4.2). Cette équation est ensuite modélisée par un simple neurone, comme illustrée sur la figure 4.14. Mentionnons qu'une spécialisation des poids en fonction de n dans l'équation (4.6) a été testée, sans que nous ayons remarqué de meilleures performances. Notons que dans ce cas, une couche de N neurones est nécessaire pour obtenir un vecteur de fiabilités de taille N et que le nombre de poids est multiplié par N.

Pour créer le jeu d'entraînement du neurone, nous générons dans un premier temps un jeu $\mathcal{T}_{\mathrm{BP}}$ de mots de code bruités, en considérant un canal AWGN à entrée binaire, avec un alphabet d'entrées BPSK (± 1) et un bruit de variance σ^2 fixée. $\mathcal{T}_{\mathrm{BP}}$ est ensuite décodé par le BP. Comme le canal AWGN et le BP sont symétriques, uniquement le mot de code tout zéro est considéré. Les mots bruités de $\mathcal{T}_{\mathrm{BP}}$ pour lesquels le BP n'arrive pas à converger vers un mot de code au bout de I_{test} itérations forment un sous-ensemble de $\mathcal{T}_{\mathrm{BP}}$, noté $\mathcal{T}_{\text{BP-OSD}}$. Les ensembles $\tilde{\mathbf{L}}_n := \left\{\tilde{L}_n^{(0)}, \ldots, \tilde{L}_n^{(I_{\mathrm{test}})}\right\}$, avec $n \in [1, N]$ et pour chaque mot bruité appartenant à $\mathcal{T}_{\text{BP-OSD}}$ constituent alors le jeu d'entraînement du neurone. Ce jeu d'entraînement permet

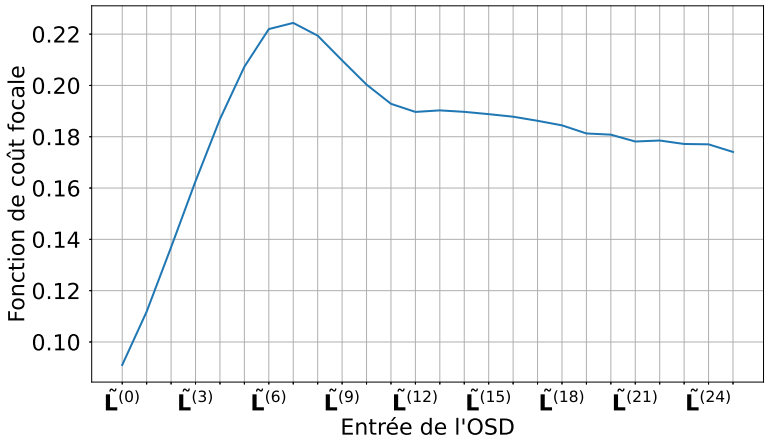


Figure 4.15 – Fonction de coût focale avec $\gamma = 10$ selon les $\tilde{\mathbf{L}}^{(i)}$, $i \in [0, I_{\text{test}}]$ (Code-2, $I_{\text{test}} = 25$, SNR = 3.5 dB).

ainsi au neurone d'être entraîné uniquement dans les cas où le BP ne converge pas vers un mot de code. Le jeu de validation est construit selon la même procédure.

4.4.2.2 Fonction de coût focale

Pour optimiser les poids du neurone, nous proposons d'utiliser la fonction de coût focale, que nous avons introduit dans la section 4.3. Dans la section 4.3.1, nous avons déjà expliqué l'intérêt de la fonction de coût locale par rapport au post-traitement OSD. Le but ici est d'étudier davantage son comportement, lorsque des décodeurs OSDs sont appliqués sur les LLRs a posteriori des itérations du BP. Pour ce faire, dans la figure 4.15, nous avons tracé la valeur moyenne de $f_{\text{FL}}\left(\tilde{L}_n^{(i)}\right) = -\left(1-\sigma\left(\tilde{L}_n^{(i)}\right)\right)^{\gamma} \log\left(\sigma\left(\tilde{L}_n^{(i)}\right)\right)$ pour $i \in [0, I_{\text{test}}]$. Notons qu'on considère ici le mot de code tout zéro et les paramètres de simulations sont les mêmes que ceux de la figure 4.13.

Nous constatons que l'allure des courbes des deux figures est relativement proche. D'une part, les $\tilde{\mathbf{L}}^{(i)}$ les plus pénalisés (itérations 5 à 9) par la fonction de coût focale correspondent aux $\tilde{\mathbf{L}}^{(i)}$ entraînant les plus mauvaises performance avec l'OSD-p ($p \leq 2$) sur la figure 4.13. D'autre part, les $\tilde{\mathbf{L}}^{(i)}$ des premières itérations ($0 < i \leq 3$) sont les moins pénalisés par la fonction de coût focale et obtiennent les meilleures performances avec l'OSD. La fonction de coût focale capture ainsi le comportement global de l'OSD selon les $\tilde{\mathbf{L}}^{(i)}$. Notons toutefois qu'en $\tilde{\mathbf{L}}^{(0)}$, les allures des courbes sur deux les figures ne concordent pas.

Pour expliquer ceci, nous avons calculé le nombre et la moyenne des LLRs négatifs parmi les K LLRs les plus fiables, pour les vecteurs $\tilde{\mathbf{L}}^{(0)}$ et $\tilde{\mathbf{L}}^{(1)}$ dans le jeu de test. Rappelons que nous considérons ici le mot de code tout zéro, donc un LLR négatif correspond à un bit mal décodé. Les vecteurs $\tilde{\mathbf{L}}^{(0)}$ possèdent 9194 LLRs négatifs, de valeur moyenne -5.19, tandis que les vecteurs. $\tilde{\mathbf{L}}^{(1)}$ ont 5241 LLR négatifs, de valeur moyenne -6.42. $\tilde{\mathbf{L}}^{(0)}$ contient donc

plus d'erreurs en moyenne que $\tilde{\mathbf{L}}^{(1)}$ parmi les K noeuds de données les plus fiables. D'où leurs écarts de performance avec l'OSD sur la figure 4.13. L'amplitude moyenne est néanmoins plus grande pour $\tilde{\mathbf{L}}^{(1)}$. Ceci implique donc des pénalités plus fortes en moyenne pour $\tilde{\mathbf{L}}^{(1)}$ avec la fonction de coût focale, ce qui coïncide avec son allure sur la figure 4.15. Une fonction de coût plus fine que la fonction de coût focale présentée ici serait ainsi à prévoir pour de futurs travaux.

Pour finir, notre objectif est d'optimiser les poids définissant un LLR accumulé $\tilde{L}_n^{(\mathrm{NS})}$, selon l'équation (4.6). En considérant le mot de code tout zéro pour construire le jeu d'entraînement, la fonction de coût focale s'écrit :

$$f_{\mathrm{FL}}\left(\tilde{L}_n^{(\mathrm{NS})}\right) = -\left(1-\sigma\left(\tilde{L}_n^{(\mathrm{NS})}\right)\right)^{\gamma} \log\left(\sigma\left(\tilde{L}_n^{(\mathrm{NS})}\right)\right) \tag{4.7}$$

4.4.3 Post-traitement OSD multiples

Afin de se rapprocher de la performance du décodage ML, nous considérons ici un budget de Z OSDs maximum pour le post-traitement du BP. Pour sélectionner sur quelles itérations du BP un OSD doit être appliqué, nous proposons une méthode de sélection par complémentarité des $i \in [0, I_{\mathrm{test}}]$ itérations et de la somme neuronale NS donnée par l'équation (4.6). Il convient de mentionner que cette procédure de sélection emploie une technique similaire à celle utilisée pour construire la diversité de BP-RNNs spécialisés $\mathcal{D}_Z$, compte tenu que les BP-RNNs sont également sélectionnés par complémentarité dans la sous-section 3.3.1.

Pour réaliser ceci, nous suivons les étapes ci-dessous :

(1) Un jeu de test $\mathcal{T}_{\text{BP-OSD}}$ est dans premier temps généré en suivant la méthode décrite dans le paragraphe 4.4.2.1.

(2) La performance de décodage du post-traitement OSD-p est ensuite évaluée sur $\mathcal{T}_{\text{BP-OSD}}$ après chaque itération i du BP ainsi qu'avec la somme neuronale NS. Nous notons par $\mathcal{F}^{(i)} \subset \mathcal{T}_{\text{BP-OSD}}$ (resp. $\mathcal{F}^{(\mathrm{NS})} \subset \mathcal{T}_{\text{BP-OSD}}$) le sous-ensemble des mots bruités sur lequel l'OSD-p appliqué après l'itération i (resp. la somme neuronale NS) a engendré un échec.

(3) Nous construisons récursivement une liste, dénotée $\mathcal{L}$, dans laquelle les itérations et NS sont triées selon leur complémentarité avec l'OSD-p. Cette liste est initialisée avec NS, soit $\mathcal{L} = \{\mathrm{NS}\}$, étant donné que la somme neuronale est optimisée pour le post-traitement OSD.

(4) Pour ajouter un nouvel élément i_{new} à $\mathcal{L}$, nous proposons d'appliquer la règle suivante :

$$i^{\mathrm{new}} = \underset{i \in \{0,\ldots,I_{\mathrm{test}}\} \setminus \mathcal{L}}{\operatorname{argmin}} \left| \mathcal{F}_{\mathcal{L}} \cap \mathcal{F}^{(i)} \right|, \tag{4.8}$$

où $\mathcal{F}_{\mathcal{L}} := \mathcal{F}^{(\mathrm{NS})}$ si $\mathcal{L} = \{\mathrm{NS}\}$, $\mathcal{F}_{\mathcal{L}} := \mathcal{F}^{(\mathrm{NS})} \cap \left(\cap_{i \in \mathcal{L}} \mathcal{F}^{(i)}\right)$ sinon. La règle ci-dessus est appliquée récursivement $I_{\mathrm{test}} + 1$ fois, jusqu'à ce que $\mathcal{L}$ contienne toutes les itérations

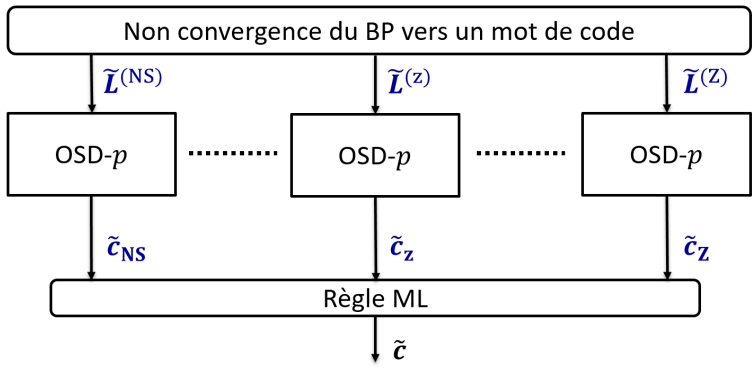


Figure 4.16 – BP-$\mathcal{L}_Z$-OSD-p

et la somme neuronale. Si l'argument minimum de (4.8) n'est pas unique, un choix arbitraire est fait parmi ces valeurs.

(5) Pour $Z \leq I_{\text{test}} + 2$, $\mathcal{L}_Z$ est définie comme étant la sous liste des Z premiers élements de $\mathcal{L}$. $\mathcal{L}_Z$ est ainsi une liste ordonnée de Z éléments, indiquant chacun l'entrée d'un décodeur OSD et représentant Z niveaux de fiabilités complémentaires vis-à-vis du post-traitement OSD-p.

Nous proposons ensuite une stratégie de post-traitements OSD-p multiples basée sur la liste $\mathcal{L}_Z$. Plus précisément, un post-traitement OSD-p est appliqué après chaque élément de $\mathcal{L}_Z$ dans le cas où le BP ne converge pas vers un mot de code. Une règle ML telle que définie dans (2.5) est utilisée pour choisir le mot de code final parmi les Z mots de code candidats. Nous notons cette méthode de décodage par BP-$\mathcal{L}_Z$-OSD-p et la figure 4.16 montre le schéma bloc correspondant.

Enfin, nous soulignons que ni la modélisation par neurone de la somme pondérée, ni la construction de $\mathcal{L}_Z$, ne dépendent des dimensions N et K du code LDPC considéré. Par conséquent, la méthodologie présentée dans cette section est reproductible pour tout code LDPC.

4.4.4 Résultats en termes de FER

Cette sous-section est dédiée à l'évaluation des différentes stratégies discutées, à travers l'évaluation du taux d'erreur paquet. Les gains sont évalués pour un FER de 10^{-4}.

4.4.4.1 Paramètres de simulations

Trois codes LDPC ont été considérés pour nos simulations. Leurs paramètres sont donnés dans le tableau 4.1. Le nombre d'itérations I_{test} a été fixé à 25 pour le Code-2 et le Code-5 et à

Tableau. 4.1 – Paramètres des codes LDPC

	N	K	R_c	d_n	d_m
Code **CCSDS** [44] (Code-2)	128	64	0.5	3-5	8
Code de **Tanner** [57] (Code-5)	155	64	0.41	3	5
Code de **MacKay** [80] (Code-6)	1008	504	0.5	3	6

Tableau. 4.2 – Paramètres Keras

Paramètres	**Valeurs des paramètres**
Méthode de descente de gradient	RMSprop [53] (initialisé avec un pas d'adaptation de 10^{-3})
Nombre d'époques	50
Taille des paquets d'entraînement	128
Taille des paquets de test	512

10 pour le Code-6. Pour limiter le nombre de mots de code candidats lors du post-traitement OSD-p, nous évaluons ce dernier pour $p = \{0, 1, 2\}$.

Concernant le neurone présenté en sous-section 4.4.2, un premier jeu $\mathcal{T}_{\text{BP-OSD}}$ comprenant 10000 mots bruités a été généré pour construire le jeu d'entraînement. Les poids, initialisés à 1, ont ensuite été optimisés sur 50 époques, afin que la fonction de coût focale converge vers une limite (et ainsi éviter du sous-apprentissage). De plus, en mesurant le FER de l'OSD-p avec $\tilde{\mathrm{L}}^{(\mathrm{NS})}$ selon l'exposant focal $\gamma \geq 0$, nous avons déterminé empiriquement $\gamma = 10$ comme étant un bon choix [5]. Les hyper-paramètres Keras sont explicités plus en détail dans le tableau 4.2. Enfin, le neurone est entraîné pour chaque valeur de SNR allant de 2.5 dB à 4.5 dB (resp. 1.5 dB à 3.5 dB), avec un pas de 0.5 dB pour le Code-2 (resp. le Code-5/Code-6).

Nous considérons un budget maximum de 3 post-traitements OSD-p afin de se rapprocher de la performance ML tout en limitant la complexité induite par l'OSD. Une liste $\mathcal{L}$ est construite selon la méthode décrite en sous-section 4.4.3 pour chaque valeur de SNR et une liste $\mathcal{L}_3$ est ensuite déterminée à partir de $\mathcal{L}$ comme décrit en sous-section 4.4.3.

À des fins de comparaisons, la stratégie de post-traitement OSD d'une diversité $\mathcal{D}_3$ de trois BP-RNNs spécialisés telle que présentée en section 4.2 est également évaluée pour le Code-2. Plus précisément, les 3 BP-RNNs sont entraînés avec la fonction de coût focale sur les 3 premières classes d'ensembles absorbants du tableau 3.3. Nous considérons ensuite trois stratégies de décodage, chacune correspondant à une mesure de fiabilité différente. La première et la seconde sont respectivement $\mathcal{D}_3$-$\tilde{\mathrm{L}}^{(I_{\text{test}})}$-OSD-$p$ et $\mathcal{D}_3$-$\tilde{\mathrm{L}}^{(\mathrm{S})}$-OSD-$p$. La troisième consiste à optimiser pour chaque BP-RNN un neurone seul avec les paramètres décrits ci-dessus. Trois fiabilités $\tilde{\mathrm{L}}^{(\mathrm{NS})}$ sont ainsi calculées puis évaluées avec un post-traitement

5. En particulier, une optimisation avec une entropie croisée binaire ($\gamma = 0$) nous a donné une moins bonne performance.

Tableau. 4.3 – Récapitulatif des différents décodeurs comparés par figure

Figures	Codes	Décodeurs comparés
4.17	Code-2	BP-$\tilde{\mathbf{L}}^{(\mathrm{S})}$-OSD-$p$ BP-$\tilde{\mathbf{L}}^{(\mathrm{NS})}$-OSD-$p$ BP-$\mathcal{L}_3$-OSD-p
4.18	Code-2	$\mathcal{D}_3$-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-$p$ $\mathcal{D}_3$-$\tilde{\mathbf{L}}^{(\mathrm{S})}$-OSD-$p$ $\mathcal{D}_3$-$\tilde{\mathbf{L}}^{(\mathrm{NS})}$-OSD-$p$
4.19	Code-2	D_1-$\tilde{\mathbf{L}}^{(\mathrm{NS})}$-OSD-$p$ $\mathcal{D}_3$-$\tilde{\mathbf{L}}^{(\mathrm{NS})}$-OSD-$p$ BP-$\mathcal{L}_3$-OSD-p
4.20	Code-5	BP-$\tilde{\mathbf{L}}^{(\mathrm{NS})}$-OSD-$p$ BP-$\mathcal{L}_3$-OSD-p BP-$\mathcal{L}_3$-OSD-2 (Complexité réduite avec la méthode de la sous-section 4.1.4)
4.21	Code-6	BP-$\tilde{\mathbf{L}}^{(\mathrm{NS})}$-OSD-$p$ BP-$\mathcal{L}_3$-OSD-p BP-$\mathcal{L}_3$-OSD-2 (Complexité réduite avec la méthode de la sous-section 4.1.4)

OSD-p. Le décodeur correspondant est alors noté par $\mathcal{D}_3$-$\tilde{\mathbf{L}}^{(\mathrm{NS})}$-OSD-$p$.

Enfin, le tableau 4.3 présente par figure les comparaisons effectuées entre les différentes méthodes de décodage proposées. De plus, l'annexe C fournit une analyse des poids du neurone simple optimisés avec la fonction de coût focale pour le post-traitement OSD du BP.

4.4.4.2 Code CCSDS : Post-traitement OSD du BP

Nous comparons sur la figure 4.17 les performances des décodeurs BP-$\tilde{\mathbf{L}}^{(\mathrm{S})}$-OSD-$p$, BP-$\tilde{\mathbf{L}}^{(\mathrm{NS})}$-OSD-$p$ et BP-$\mathcal{L}_3$-OSD-p pour le Code-2. Nous observons dans un premier temps que BP-$\tilde{\mathbf{L}}^{(\mathrm{NS})}$-OSD-$p$ présente une performance légèrement meilleure que BP-$\tilde{\mathbf{L}}^{(\mathrm{S})}$-OSD-$p$ pour $p = 1$ (similaire pour $p = 0$). Pour un ordre p fixé à 2, le post-traitement OSD avec $\tilde{\mathbf{L}}^{(\mathrm{NS})}$ permet d'obtenir un gain de 0.16 dB par rapport à celui utilisant $\tilde{\mathbf{L}}^{(\mathrm{S})}$. Donc, plus l'ordre de l'OSD augmente, mieux $\tilde{\mathbf{L}}^{(\mathrm{NS})}$ est adapté au post-traitement OSD. Les poids introduits dans l'équation (4.2) permettent donc d'atténuer les $\tilde{L}_n^{(i)}$ problématiques lors du calcul des fiabilités $\tilde{L}_n^{(\mathrm{NS})}$ et de diminuer en conséquence la probabilité d'avoir $p \geq 1$ erreurs dans les K noeuds de donnée les plus fiables. Nous constatons ensuite que l'application d'un OSD-p après chaque vecteur de LLRs de $\mathcal{L}_3$ apporte un gain croissant avec p par rapport à $\tilde{\mathbf{L}}^{(\mathrm{NS})}$: le gain est respectivement de 0.12 dB, 0.22 dB et 0.27 dB pour $p = \{0, 1, 2\}$. De plus, nous remarquons que BP-$\mathcal{L}_3$-OSD-2 obtient une performance située à seulement 0.17 dB de la performance du décodage ML [45].

Au niveau de la complexité calculatoire de BP-$\mathcal{L}_3$-OSD-p, le nombre de mots de code

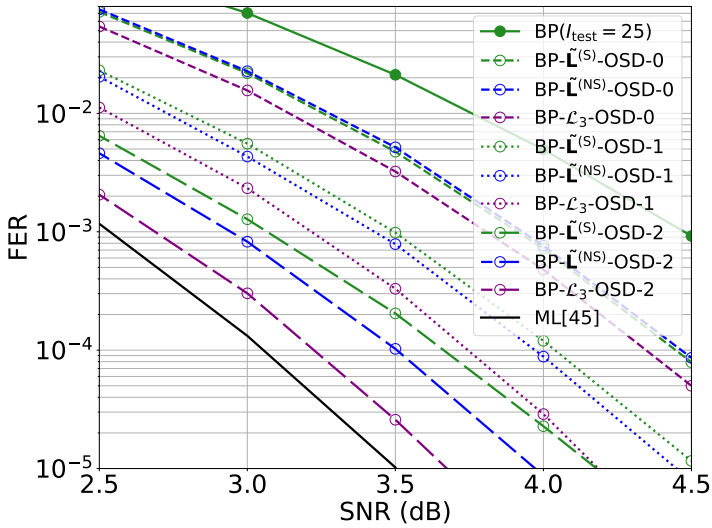


Figure 4.17 – Résultats FER pour le Code-2, $\tilde{\mathbf{L}}^{(S)}$ vs $\tilde{\mathbf{L}}^{(NS)}$.

testés lorsque le BP échoue est égale à $3 \times \sum_{i=0}^{p} \binom{K}{i}$ étant donné que $Z = 3$ post-traitement OSD-p sont effectués. Ainsi, la valeur de Z impacte linéairement la complexité calculatoire liée au nombre de mots de code testés et au nombre de résolutions de système. BP-$\mathcal{L}_3$-OSD-p possède donc un coût calculatoire lié à l'OSD trois fois plus grand que BP-$\tilde{\mathbf{L}}^{(NS)}$-OSD-$p$ ou que BP-$\tilde{\mathbf{L}}^{(S)}$-OSD-p. Notons aussi que le calcul du vecteur $\tilde{\mathbf{L}}^{(NS)}$ requiert également 26 multiplications supplémentaires par rapport à $\tilde{\mathbf{L}}^{(S)}$ à cause de la pondération.

4.4.4.3 Code CCSDS : Post-traitement OSD de BP-RNNs spécialisés

La figure 4.18 affiche les performances des décodeurs $\mathcal{D}_3$-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-$p$, $\mathcal{D}_3$-$\tilde{\mathbf{L}}^{(S)}$-OSD-p et $\mathcal{D}_3$-$\tilde{\mathbf{L}}^{(NS)}$-OSD-$p$. Nous remarquons dans un premier temps que pour $p = 0$, ces trois stratégies obtiennent des performances similaires. En revanche, le décodeur $\mathcal{D}_3$-$\tilde{\mathbf{L}}^{(NS)}$-OSD-$p$ présente un gain de 0.06 dB (resp. 0.14 dB) pour $p = 1$ (resp. $p = 2$) par rapport à $\mathcal{D}_3$-$\tilde{\mathbf{L}}^{(S)}$-OSD-p. Ces gains s'accentuent à 0.13 dB et 0.26 dB par rapport à $\mathcal{D}_3$-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-OSD-$p$. $\mathcal{D}_3$-$\tilde{\mathbf{L}}^{(NS)}$-OSD-$p$ est donc une meilleure solution que les deux autres en termes de performance pour $p = \{1, 2\}$. Elle ne requiert de plus qu'une faible complexité matérielle supplémentaire liée au calcul des trois sorties neuronales $\tilde{\mathbf{L}}^{(NS)}$. En effet, pour chaque $\tilde{\mathbf{L}}^{(NS)}$, 26 multiplications sont induites par les 26 poids, ce qui revient à 78 multiplications supplémentaires pour $\mathcal{D}_3$-$\tilde{\mathbf{L}}^{(NS)}$-OSD-$p$. Notons que stocker ces poids en mémoire requiert aussi un léger coût de stockage mémoire supplémentaire.

Le décodeur $\mathcal{D}_3$-$\tilde{\mathbf{L}}^{(NS)}$-OSD-$p$ est ensuite comparé avec le décodeur BP-$\mathcal{L}_3$-OSD-p sur la figure 4.19. Nous observons que BP-$\mathcal{L}_3$-OSD-0 apporte une légère amélioration par rapport à $\mathcal{D}_3$-$\tilde{\mathbf{L}}^{(NS)}$-OSD-0. Le gain s'accentue à 0.1 dB pour $p = 1$ puis à 0.12 dB pour $p = 2$. Par conséquent, utiliser le décodeur BP seul et construire une liste de vecteurs

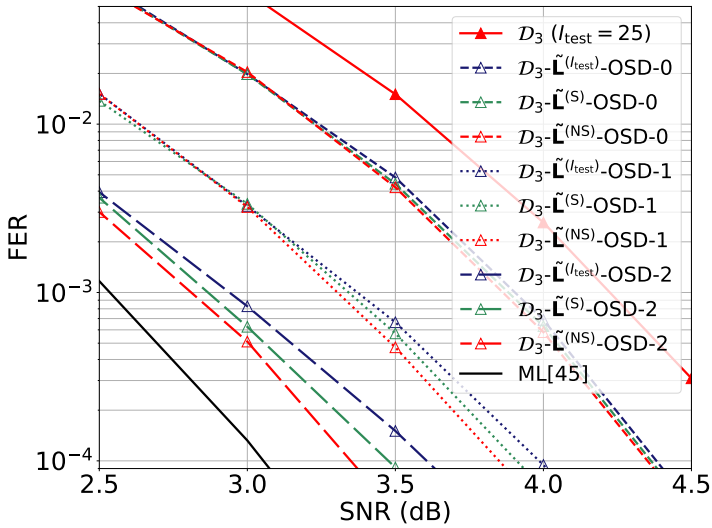


Figure 4.18 – Résultats FER pour le Code-2, $\mathcal{D}_3$.

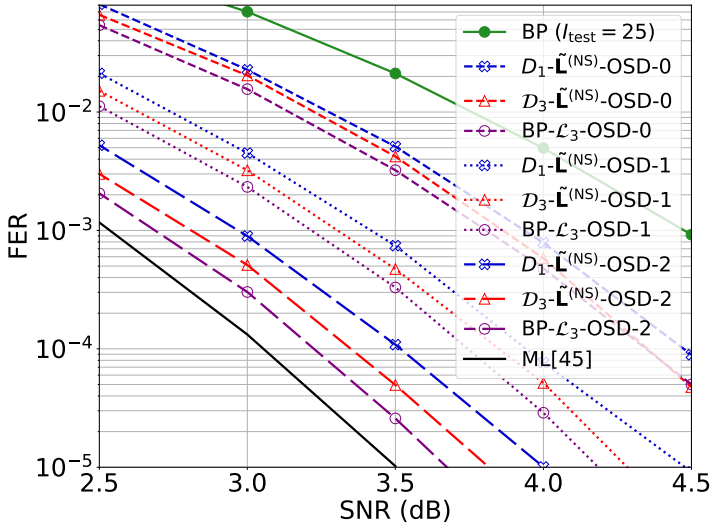


Figure 4.19 – Résultats FER pour le Code-2, $\mathcal{D}_3$ vs $\mathcal{L}_3$.

de LLRs complémentaires est une meilleure stratégie en termes de performance avec le post-traitement OSD-p. De plus, les poids des BP-RNNs spécialisés de $\mathcal{D}_3$ induisent des multiplications supplémentaires lors des étapes data-pass et a posteriori ainsi qu'un coût de stockage mémoire supplémentaire par rapport au BP. BP-$\mathcal{L}_3$-OSD-p présente ainsi un meilleur compromis performance/complexité que $\mathcal{D}_3$-$\tilde{\mathbf{L}}^{(NS)}$-OSD-$p$. Enfin, en comparant les figures 4.17 et 4.19, nous remarquons que D_1-$\tilde{\mathbf{L}}^{(NS)}$-OSD-$p$ et BP-$\tilde{\mathbf{L}}^{(NS)}$-OSD-$p$ obtiennent les

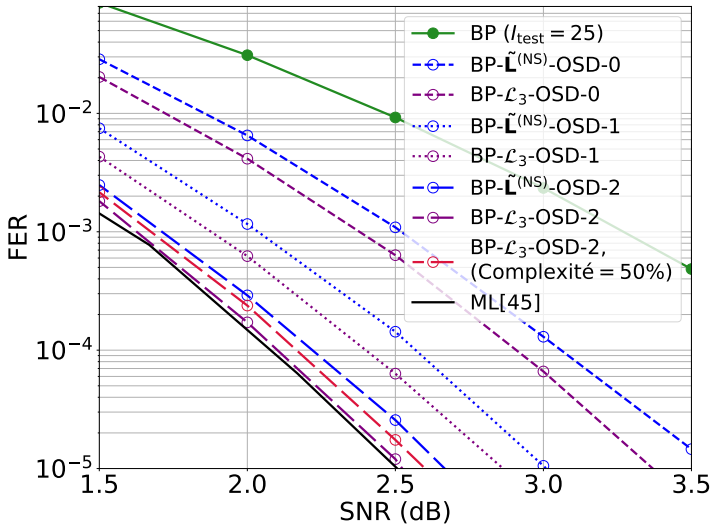


Figure 4.20 – Résultats FER pour le Code-5

mêmes performances pour $p = \{0, 1, 2\}$. BP-$\tilde{\mathbf{L}}^{(NS)}$-OSD-$p$ offre donc un meilleur compromis performance/complexité que D_1-$\tilde{\mathbf{L}}^{(NS)}$-OSD-$p$.

4.4.4.4 Code de Tanner : Post-traitement OSD du BP

Les résultats de simulations obtenus avec le Code-5 (Tanner) sont présentés sur la figure 4.20. Notons que créer une diversité de BP-RNNs spécialisés sur les ensembles absorbants responsables des dégradations du BP dans la région de *waterfall* est en pratique trop complexe pour ce code. En effet, le calcul des ensembles absorbants d'une taille équivalente à la capacité de correction du code de Tanner avec l'algorithme AS-DFS 2 requiert un coût calculatoire trop important du fait de la taille et du rendement du code. La performance du post-traitement OSD d'une diversité de BP-RNNs spécialisés $\mathcal{D}_3$ n'est donc pas évaluée pour ce code. Le décodage par BP-$\mathcal{L}_3$-OSD-0 présente un gain de 0.16 dB par rapport à BP-$\tilde{\mathbf{L}}^{(NS)}$-OSD-0. Ce gain reste similaire lorsque l'ordre de l'OSD vaut 1 ou 2. De plus, la stratégie proposée BP-$\mathcal{L}_3$-OSD-2 atteint quasiment la performance du décodage ML.

Pour ce code, nous considérons également un décodeur BP-$\mathcal{L}_3$-OSD-2 calculant 50% moins de mots de candidats lors du post-traitement OSD. Cette réduction de la complexité est réalisée grâce à la méthode introduite en sous-section 4.1.4. Le couple optimal (T_1, T_2) est calculé pour chaque valeur de SNR test. Plus précisément, tous les couples (T_1,T_2) satisfaisant (4.3) sont déterminés pour un nombre de mots de code candidats N_c maximal souhaitée. Une réduction de complexité de 50% équivaut ici à $N_C = 3121$ puisque $3 \times \sum_{i=0}^{2} \binom{64}{i} = 6243$ mots de code sont testés à chaque échec de décodage BP avec le décodeur BP-$\mathcal{L}_3$-OSD-2. Nous évaluons ensuite pour chaque valeur de SNR la performance de BP-$\mathcal{L}_3$-OSD-2 avec tous les couples trouvés, puis nous sélectionnons celui donnant le meilleur taux d'erreur

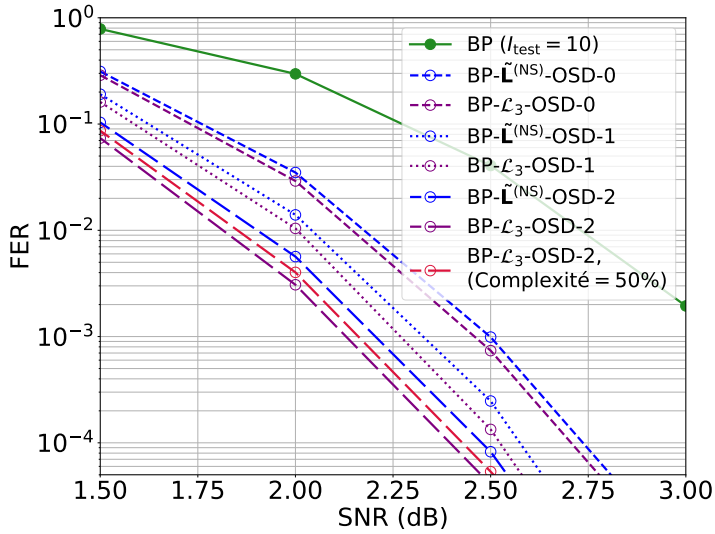


Figure 4.21 – Résultats FER pour le Code-6

paquet.

La performance correspondante au décodeur BP-$\mathcal{L}_3$-OSD-2 de complexité réduite est illustrée sur la figure 4.20. Nous remarquons que réduire le nombre de mots de code candidats de 50% lors du post-traitement OSD-2 induit une dégradation de seulement 0.08 dB. Le décodeur BP-$\mathcal{L}_3$-OSD-2 est donc capable d'approcher la performance du décodage ML, même lorsqu'une réduction de la complexité du post-traitement OSD-2 est imposée.

4.4.4.5 Code de MacKay : Évaluation sur des codes plus longs

La figure. 4.21 montre les résultats de simulations pour le Code-6 (MacKay). Ce code possède des dimensions N et K plus grandes que celles du Code-2 et le Code-5. Toutefois, la méthodologie présentée dans cette section reste reproductible pour ce code plus long, étant donné qu'elle ne dépend pas des dimensions du code considéré. Nous notons que BP-$\mathcal{L}_3$-OSD-p fournit de petites améliorations (0.05 dB) par rapport à BP-$\tilde{\mathbf{L}}^{(NS)}$-OSD-$p$ pour $p = 1$ et des performances similaires pour $p = 0$. Le gain est accentué à 0.1 dB pour un ordre $p = 2$.

Toutefois, comme $K = 504$, $N_c = 381783$ mots de code candidats sont calculés par BP-$\mathcal{L}_3$-OSD-2 à chaque échec de décodage BP. Par conséquent, réduire le nombre de mots code calculés lors du post-traitement OSD-2 est nécessaire pour le code MacKay. Le décodeur BP-$\mathcal{L}_3$-OSD-2 est ainsi évalué avec une réduction du nombres de mots de code calculés de 50%. Un couple de seuils optimaux (T_1, T_2) est à nouveau déterminé pour chaque valeur de SNR testé. Nous observons que le décodeur BP-$\mathcal{L}_3$-OSD-2 avec une réduction de la complexité de 50% tend à être quasiment aussi efficace que le décodeur standard BP-$\mathcal{L}_3$-OSD-2, avec une dégradation de 0.03 dB uniquement.

4.5 Conclusion

Ce chapitre a abordé la combinaison du post-traitement OSD et du décodage itératif BP/BP-RNN. Dans un premier temps, nous avons proposé la stratégie de décodage $\boldsymbol{\mathcal{D}_Z}$-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-**OSD-p**, dans laquelle une diversité $\mathcal{D}_Z$ de BP-RNNs spécialisés sur les motifs d'erreurs avec un support d'ensemble absorbant est concaténée avec un post-traitement OSD d'ordre p. Il a été mis en évidence que l'étape de post-traitement exploite avantageusement la diversité de décodage pour combler l'écart avec la performance du décodage ML. De plus, nous avons montré que la fonction de coût focale est une meilleure alternative que la fonction BCE pour l'optimisation des poids des BP-RNNs, induisant des performances encore meilleures pour $\boldsymbol{\mathcal{D}_Z}$-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-**OSD-p**.

Dans un second temps, nous nous sommes intéressés à améliorer la performance du post-traitement OSD du BP pour des codes LDPC courts. Le but était ici de fournir une solution alternative à $\boldsymbol{\mathcal{D}_Z}$-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-**OSD-p**, possédant une complexité matérielle plus faible. À cette fin, nous avons proposé le vecteur de LLR accumulé pour le post-traitement OSD, $\tilde{\mathbf{L}}^{(\mathbf{NS})}$. L'optimisation a notamment été réalisée grâce à une méthode d'entraînement utilisant la fonction de coût focale. Nous avons ensuite construit une liste ordonnée $\mathcal{L}_Z$ d'itérations du BP complémentaires vis-à-vis de l'OSD et comprenant également la somme pondérée. Cette liste nous a ainsi permis de proposer un nouvelle stratégie de décodage, **BP-$\boldsymbol{\mathcal{L}_Z}$-OSD-p**, où un post-traitement OSD est appliqué après chaque élément de $\mathcal{L}_Z$. Nos résultats montrent que cette nouvelle méthode de décodage est capable d'atteindre la performance du décodage ML pour des codes LDPC courts. De plus, **BP-$\boldsymbol{\mathcal{L}_Z}$-OSD-p** présente de meilleures performances que $\boldsymbol{\mathcal{D}_Z}$-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-**OSD-p** (ou même $\boldsymbol{\mathcal{D}_Z}$-$\tilde{\mathbf{L}}^{(\mathbf{NS})}$-**OSD-p**) pour $p = \{0, 1, 2\}$, tout en gardant une complexité plus faible du fait de l'utilisation du BP standard au lieu de BP-RNNs spécialisés. Enfin, cette seconde méthodologie peut être reproduite pour n'importe quelle longueur de code, à condition que la complexité induite par le post-traitement OSD soit limitée.

Ces contributions ont donné lieu aux publications dans le journal international Transactions on Communications en 2022 [J1] ($\boldsymbol{\mathcal{D}_Z}$-$\tilde{\mathbf{L}}^{(I_{\text{test}})}$-**OSD-p**) et dans la conférence International Symposium on Topics in Coding de 2023 [IC2] (**BP-$\tilde{\mathbf{L}}^{(\mathbf{NS})}$-OSD-p** et **BP-$\boldsymbol{\mathcal{L}_Z}$-OSD-p**).

Chapitre 5

Conclusions et perspectives

Durant ces dernières années, la conception de codes correcteurs d'erreurs courts efficaces et d'algorithmes de décodage associés a soulevé de nombreux défis dans le contexte des transmissions de paquets courts pour l'Internet des Objets. En particulier, la modélisation du décodeur BP par réseaux de neurones a suscité un intérêt significatif afin d'améliorer la capacité de correction fortement dégradée des codes LDPC courts. Le travail proposé dans cette thèse s'inscrit dans cette thématique, en y développant deux axes de recherche principaux. D'une part, la création d'une diversité de décodeurs spécialisés a été explorée grâce aux développements de nouvelles méthodes d'entraînement pour le BP modélisé par un réseau de neurones récurrent (BP-RNN). D'autre part, la combinaison d'un décodeur BP ou d'une diversité de BP-RNNs avec un post-traitement OSD a été traitée, afin de fournir des stratégies de décodages efficaces pour atteindre la performance de décodage par maximum de vraisemblance.

Contributions principales

Dans le cadre de ces deux axes de recherche, nous avons proposé dans ce manuscrit les contributions principales suivantes :

- ***Diversité de décodeurs BP-RNNs spécialisés [IC1, J1] :*** La création d'une diversité de décodeurs BP-RNNs $\mathcal{D}$, motivée par [16, 17, 55], permet de compter sur plusieurs décodeurs neuronaux pour améliorer la performance de décodage. Pour réaliser ceci, nous avons développé de nouvelles méthodes d'entraînement, spécialisant les décodeurs BP-RNNs dans le décodage de motifs d'erreurs partageant les mêmes propriétés structurelles. Une première méthode d'entraînement a consisté à classifier les supports de petits motifs d'erreurs pour des codes LDPC courts avec une faible capacité de correction (rendement élevé), puis à entraîner un BP-RNN spécifiquement par classe de supports. Cette méthode d'entraînement a ensuite été adaptée pour des codes LDPC courts avec une meilleure capacité de correction (rendement plus faible et/ou possédant une taille de mot de code plus grande), en concentrant notre attention sur des motifs d'erreurs possédant un support d'ensemble absorbant. À cette fin, nous avons développé un algorithme d'énumération des ensembles absorbants, AS-DFS.
- ***Procédure de sélection et architectures de décodage d'une diversité de BP-RNNs [IC1, J1] :*** Notre seconde contribution a abordé la question de l'organisation d'une diversité

de BP-RNNs spécialisés en une architecture de décodage. Pour limiter la complexité calculatoire et matérielle, nous avons tout d'abord réduit le nombre de décodeurs BP-RNNs en sélectionnant uniquement les plus complémentaires en termes de probabilités d'erreurs conjointes. Nous avons ensuite proposé deux architectures de décodage, dans lesquelles les décodeurs BP-RNNs sélectionnés sont exécutés soit en parallèle soit en série. Les résultats de simulations ont montré que l'architecture parallèle d'une diversité $\mathcal{D}_Z$ de Z BP-RNNs spécialisés sur les ensembles absorbants permet d'améliorer la capacité de correction de codes LDPC courts, sans augmenter la latence de décodage maximale par rapport au BP. Nous avons également montré que l'architecture série d'une diversité de BP-RNNs spécialisés est équivalente au décodeur BP avec un même nombre maximal d'itérations de décodage.

- ***Post-traitement OSD d'une diversité de BP-RNNs spécialisés [J1] :*** Afin d'améliorer davantage la capacité de correction des codes LDPC courts, nous avons élaboré la stratégie de décodage $\mathcal{D}_Z$-$\tilde{\mathrm{L}}^{(I_{\text{test}})}$-**OSD-p**, combinant une diversité de BP-RNNs spécialisés avec une étape de post-traitement OSD-p. Il est alors montré que cette stratégie tire efficacement partie de la diversité apportée par l'utilisation de plusieurs BP-RNNs spécialisés, fournissant ainsi un moyen efficace de combler l'écart avec le décodage par maximum de vraisemblance.
- ***Post-traitement OSD du BP [IC2] :*** La combinaison du BP standard avec le post-traitement OSD a été également investiguée dans le but d'en améliorer les performances. Pour réaliser ceci, nous avons tout d'abord optimisé un neurone simple avec la fonction de coût focale afin de calculer pour chaque mot bruité destiné au post-traitement le vecteur de LLRs accumulés adapté à l'OSD, $\tilde{\mathbf{L}}^{(\mathbf{NS})}$. Notons que c'est à notre connaissance la première utilisation de la fonction de coût focale en décodage. Nous avons ensuite construit une liste de Z vecteurs LLRs a posteriori complémentaires vis-à-vis de l'OSD, $\boldsymbol{\mathcal{L}}_Z$, comprenant notamment $\tilde{\mathbf{L}}^{(\mathbf{NS})}$. Enfin, nous avons proposé la stratégie de décodage **BP-**$\boldsymbol{\mathcal{L}}_Z$**-OSD-p**, où un post-traitement OSD-p est appliqué après chaque vecteur de $\boldsymbol{\mathcal{L}}_Z$. Les résultats numériques nous ont montré que cette stratégie est capable elle aussi de se rapprocher de la performance ML, fournissant ainsi un compromis performance/complexité intéressant par rapport à $\mathcal{D}_Z$-$\tilde{\mathrm{L}}^{(I_{\text{test}})}$-**OSD-p**.

Choix de la stratégie de décodage

Cette thèse a porté sur des études algorithmiques pour le décodage de codes LDPC courts. Nous proposons toutefois de revisiter les algorithmes proposés afin d'orienter sur un choix de la stratégie de décodage selon si une minimisation de la latence de décodage maximale et/ou de la complexité matérielle est souhaitée.

- ***Minimisation de la latence de décodage maximale :*** Afin de minimiser la latence de décodage maximale, nous excluons la possibilité d'appliquer un post-traitement OSD, malgré son apport notable en termes de performance. Nous considérons ensuite une minimisation ou non de la complexité matérielle :

- ***Minimisation de la complexité matérielle :*** Si l'utilisateur doit minimiser la complexité matérielle, nous proposons d'utiliser soit le décodeur **BP-RNN non spécialisé** de [10], soit le **meilleur BP-RNN spécialisé sur les ensembles absorbants** $\mathbf{D}_1$. Ces deux décodeurs possèdent en effet les mêmes performances et améliorent la capacité de correction des codes LDPC courts par rapport au BP. La configuration de poids du BP-RNN lui permet également de réutiliser les implémentations matérielles conventionnelles du BP. Le coût de stockage en mémoire supplémentaire provient donc uniquement du stockage des poids. Notons toutefois que les multiplications liées aux poids du BP-RNN ajoute une complexité matérielle par rapport au BP.
- ***Minimisation non requise de la complexité matérielle :*** S'il n'est pas nécessaire pour l'utilisateur de minimiser la complexité matérielle, **l'architecture parallèle d'une diversité de MS-RNNs spécialisés sur les ensembles absorbants** est recommandée. En effet, elle obtient une meilleure performance que le BP ou le MS pour une même latence de décodage maximale. De plus, nous avons montré que l'architecture parallèle de MS-RNNs spécialisés présente des performances similaires face à son homologue composée de BP-RNNs.

- ***Minimisation non requise de la latence de décodage maximale :*** Si l'application considérée ne nécessite pas une minimisation de la latence de décodage maximale, nous proposons d'utiliser un post-traitement OSD quand la complexité matérielle liée à l'OSD le permet.

  - ***Minimisation de la complexité matérielle :*** Dans le cas d'une minimisation de la complexité matérielle, nous conseillons d'utiliser le **décodeur BP standard seul** sans post-traitement OSD. Le BP atteint effectivement les mêmes performances que l'architecture série ou parallèle d'une diversité de BP-RNNs spécialisés pour un même nombre maximal d'itérations. De plus, il requiert une complexité matérielle plus faible, du fait de d'absence de pondération lors des étapes data-pass et a posteriori.
  - ***Minimisation non requise de la complexité matérielle :*** Pour ce dernier cas d'application, l'utilisateur cherche à atteindre la meilleure performance possible. La stratégie de post-traitement OSD multiples du BP standard **BP-$\mathcal{L}_Z$-OSD-p** est alors préconisée. Nous conseillons de fixer l'ordre de l'OSD à 2 afin d'obtenir des performances proches du décodage ML. Plusieurs raisons justifient le choix de cette stratégie. Tout d'abord, elle obtient de meilleures performances que le post-traitement OSD multiples d'une diversité de BP-RNNs spécialisés sur les ensembles absorbants, même lorsque qu'un neurone simple est optimisé pour chaque BP-RNN ($\mathcal{D}_Z$-$\tilde{\mathrm{L}}^{(\mathbf{NS})}$-**OSD-p**). Ensuite, l'utilisation du BP au lieu d'une diversité BP-RNNs assure une plus faible complexité matérielle. Enfin, cette stratégie est facilement reproductible quelque soit le code LDPC (court) considéré.

Perspectives

Une perspective immédiate à ce travail est de reproduire nos contributions sur le post-traitement OSD du BP avec le décodeur MS, moins complexe, puis de comparer les performances obtenues avec **BP-$\check{\mathbf{L}}^{(\mathbf{NS})}$-OSD-$p$** puis **BP-$\mathcal{L}_Z$-OSD-p**.

Une seconde perspective serait d'étudier d'autres diversités de vecteurs de LLRs pour des post-traitements OSD multiples du BP ou du MS. En particulier, utiliser une diversité de vecteurs $\check{\mathbf{L}}^{(\mathrm{NS})}$ dans laquelle chacun est spécialisé sur un type différent de motifs d'erreurs nous parait être une piste intéressante. Le constat derrière cette approche est le même que pour la création du diversité de BP-RNNs spécialisés, à savoir qu'un seul réseau de neurones ne peut pas apprendre à traiter de manière optimale tous les types d'erreurs. Pour réaliser ceci, nous envisageons tout d'abord d'établir un lien précis entre "les niveaux de difficulté des motifs" pour le décodage BP et la performance du post-traitement OSD. Une fiabilité selon chaque niveau de difficulté des motifs pourrait ensuite être calculée en cas d'échec du BP. Ceci reviendrait donc à entraîner un $\check{\mathbf{L}}^{(\mathrm{NS})}$ par classe de difficulté et porte ainsi des ressemblances avec la spécialisation des BP-RNNs sur les petits motifs d'erreurs ou les ensembles absorbants.

Un décodeur non exploré lors de cette thèse est le décodage par inversion de bit (Bit Flipping ou BF en anglais) [34]. Son attractivité réside dans sa faible complexité calculatoire, comparée à celle du BP. Toutefois, l'algorithme BF ne se base pas sur la propriété d'échange extrinsèque de l'information, ce qui implique des performances nettement moins satisfaisantes que celles du BP. Afin de remédier à cela, le décodage à inversion de bits et effet inertiel (Gradient Descent Bit Flipping with Momentum ou GDBF/m en anglais), une version légèrement plus complexe du BF, a été étudié dans [81]. Il y est notamment montré qu'avec des paramètres choisis judicieusement, le GBDF/m est capable d'atteindre la performance du BP. Optimiser les paramètres du GBDF/m avec un réseau de neurones pourrait donc se révéler intéressant pour fournir une décodeur aussi (voire plus) performant que le BP et moins complexe. Une méthode d'apprentissage par renforcement, telle que montrée dans [82] avec le BF, serait notamment à mettre en place pour prendre en compte le caractère discret du GBDF/m.

Une autre possibilité pour améliorer les performances des codes LDPC courts serait de directement optimiser la construction du graphe de Tanner avec des réseaux de neurones. De plus, le décodage BP se basant sur des graphes bipartis, il pourrait être intéressant de le modéliser par un réseaux de neurones en graphes (Graph Neural Network ou GNN en anglais) [83], puis d'optimiser le graphe biparti afin d'améliorer les performances du BP. Le principe de cette méthode porterait des similarités avec le décodeur BP neuronal basé sur de l'élagage des noeuds de parité [59]. La construction de nouvelles familles de codes courts et de leurs algorithme de décodages entièrement par réseaux de neurones, tel que dans [84] et [85], pourrait aussi être analysée plus en profondeur.

Enfin, établir un bilan énergétique des solutions proposées dans ce manuscrit nous paraît être une perspective essentielle au vu des enjeux écologiques actuels. En effet, l'empreinte carbone sans cesse croissante des nouvelles technologies remet en question les gains énergé-

tiques supposés bénéfiques liés au déploiement de l'IoT et de la 5G (ou 6G maintenant) [86]. Une étude énergétique pourrait donc montrer si les solutions proposées ici sont compatibles avec la notion de sobriété énergétique. Pour réaliser ceci, nous proposons entre autres de suivre une méthode équivalente à celle décrite par [87]. Plus précisément, l'auteur interroge l'intérêt énergétique du MIMO massif selon l'évolution du trafic envisagée (frugalité ou expansion), et selon la prise en compte ou non du coût de fabrication dans le bilan de puissance total. Dans notre cas d'application, le bilan énergétique du décodage aidé par réseaux de neurones devrait être estimé tant en termes de puissance requise pour l'entraînement et le fonctionnement dans un appareil numérique, que de puissance exigée pour la mise en place matérielle de ces appareils. Cette étude étayerait aussi les discussions de complexité proposées tout au long de ce manuscrit.

Annexes

Annexe A

Étude des hyper-paramètres d'entraînement du BP-RNN

Cette annexe fournit une étude supplémentaire évaluant l'impact de la méthode de descente de gradient, du pas d'apprentissage initial et de l'initialisation des poids du décodeur BP-RNN sur sa performance.

A.1 Méthode de descente de gradient

Nous revenons tout d'abord sur la discussion du paragraphe 1.3.2.3, discutant du choix de la méthode de descente de gradient, en comparant la performance du BP-RNN de [10] entraîné avec soit RMSprop [53], soit Adam [54]. Les paramètres de simulations sont les suivants. Le code considéré est le Code-3, possédant $N = 64$ bits dont $K = 46$ d'informations. Les nombres d'itérations de test et d'entraînement sont fixés à 5. Le BP-RNN [10] est entraîné pour chaque valeur de SNR comprise entre 1 dB et 9 dB, avec un pas de 1 dB. Chaque ensemble de poids obtenu a ensuite été utilisé pour la valeur de SNR correspondante lors du test. Enfin, le tableau A.1 résume les paramètres d'entraînement autres que la méthode de descente de gradient.

La figure A.1 illustre les résultats obtenus en termes de FER. Nous constatons qu'Adam et RMSprop présentent les mêmes performances en termes de FER. Les deux méthodes de descente de gradient amènent donc à une optimisation similaire des poids du BP-RNN.

A.2 Pas d'apprentissage initial

Le pas d'apprentissage initial de la méthode de descente de gradient est un hyper-paramètre important pour l'optimisation d'un réseau de neurones. En effet, un pas trop

Tableau. A.1 – Paramètres Keras

Paramètres	Valeurs des paramètres
Pas d'apprentissage initial	10^{-3}
Nombre d'époques	15
Taille des paquets d'entraînement	8192
Nombre de paquets d'entraînement	37 à 122 (dépend du SNR)
Taille des paquets de test	16384
Nombre de paquets de test	19 à 61 (dépend du SNR)

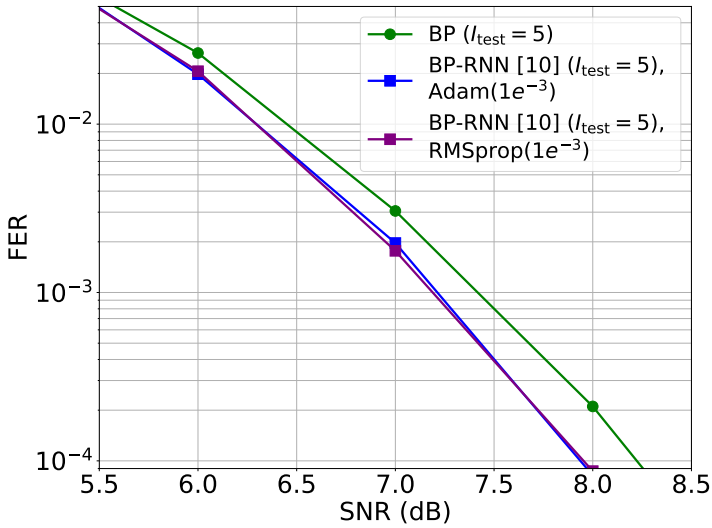


Figure A.1 – Résultats FER du Code-3, Adam vs RMSprop.

petit induit généralement un entraînement long, restant potentiellement bloqué, tandis qu'un pas trop grand conduit à un entraînement instable, ne convergeant pas vers une solution optimale.

Nous évaluons ici l'impact du pas d'apprentissage initial sur l'optimisation des poids du BP-RNN. Pour réaliser ceci, nous avons calculé le FER du BP-RNN, selon les pas d'apprentissage initiaux suivant : 10^{-2}, 10^{-3} et 10^{-4}. Les autres paramètres de simulations sont les mêmes que ceux de la sous-section précédente, à l'exception du nombre d'époques. La méthode de descente de gradient RMSprop est utilisée.

Nous observons sur la figure A.2 que pour 15 époques, le BP-RNN obtient les mêmes performances pour les pas d'apprentissage initiaux 10^{-2} et 10^{-3}. Pour un pas de 10^{-4}, 15 époques ne sont pas suffisantes pour obtenir la convergence du coût d'entraînement vers une limite finie. Pour atteindre cette limite, nous avons dû augmenter le nombre d'époques à 45. L'optimisation des poids résultante converge alors vers une solution proche des autres pas d'apprentissage initiaux. Il est donc recommandé de fixer le pas d'apprentissage avec un ordre de grandeur de 10^{-2} ou 10^{-3}, afin de réduire le temps d'entraînement. Enfin, précisons qu'un pas initial égal à 10^{-1} a aussi été testé pour SNR = 8 dB. Nous avons remarqué que le coût d'entraînement oscille fortement, ce qui indique un pas d'apprentissage initial trop grand.

A.3 Initialisation des poids

Un autre facteur impactant l'optimisation des poids d'un NN est l'initialisation de ses poids. Historiquement, l'initialisation des poids d'un réseau de neurones consistait à affecter à

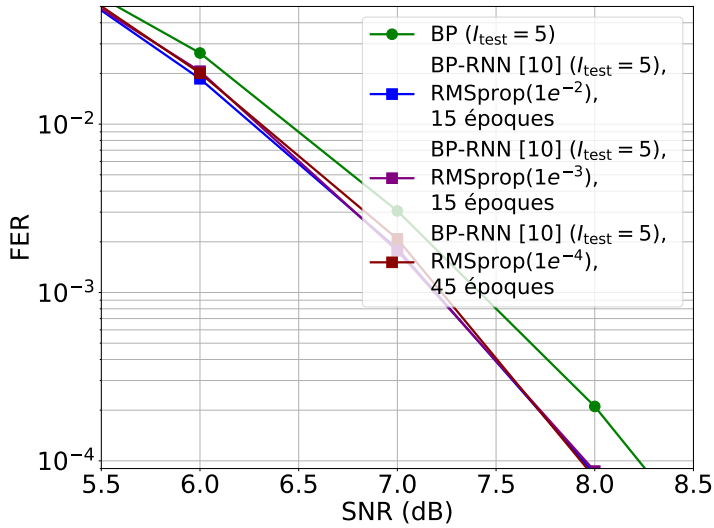


Figure A.2 – Résultats FER du Code-3, selon le pas d'apprentissage initial.

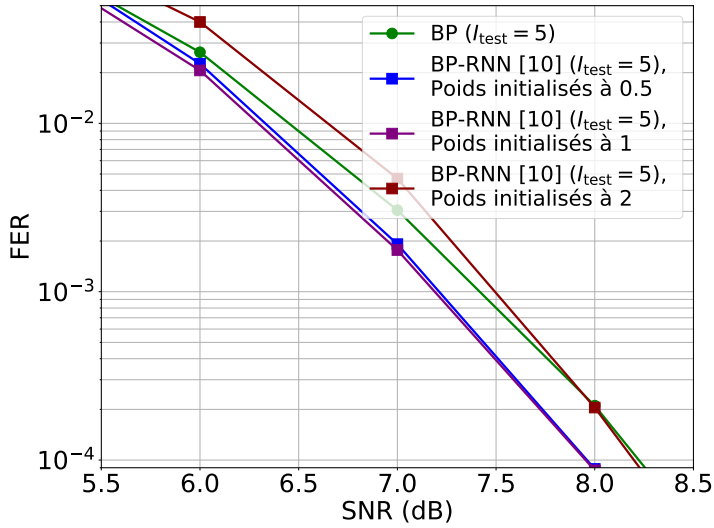


Figure A.3 – Résultats FER du Code-3, selon l'initialisation des poids du BP-RNN.

chaque poids un petit nombre aléatoire. Plus récemment, des heuristiques ont été développées afin d'initialiser spécifiquement les poids en fonction du problème traité. Par exemple, la librairie python Keras contient de nombreuses méthodes d'initialisation déjà implémentées. Notons également que l'utilisateur peut créer sa propre méthode d'initialisation personnalisée grâce à de la programmation orientée objet (comme pour les étages neuronaux d'ailleurs).

Pour le BP-RNN, l'initialisation intuitive est d'affecter à chaque poids la valeur 1. De cette

manière, le BP-RNN non entraîné correspond au BP. En d'autres termes, le BP est le point de départ de l'optimisation. Pour confirmer ce choix d'initialisation, nous avons évalué l'impact respectif d'une initialisation de tous les poids à 1, 2, et 0.5 sur la performance du BP-RNN en termes de FER. Les paramètres de simulations sont les mêmes que ceux de la section A.1 et la méthode de descente de gradient choisie est RMSprop. Les résultats correspondants sont affichés sur la figure A.3. Nous observons que l'initialisation des poids à 0.5 conduit à des performances similaires que l'initialisation à 1. En revanche, une initialisation des poids à 2 n'améliore pas (voire dégrade pour les hauts SNRs) les performances par rapport au BP. Les valeurs initiales des poids sont donc dans ce cas trop grandes et conduisent à un phénomène d'explosion des gradients. En conclusion, partir du BP pour l'optimisation des poids du BP-RNNs est est un choix d'initialisation que nous recommandons, permettant d'éviter des problèmes liés à la descente de gradient.

Annexe B

Autres approches de spécialisation du BP-RNN

En parallèle des approches de spécialisation présentées dans les chapitres 2 et 3, nous avons également développé deux autres méthodes de construction du jeu d'entraînement du BP-RNN. Ces méthodes ont pour but de créer des jeux d'entraînement ne contenant pas de motifs d'erreurs facilement décodés par le BP étant donné qu'ils sont peu utiles pour la phase d'apprentissage. Plus précisément, la première méthode consiste à générer un jeu d'entraînement à partir des mots bruités provoquant un échec lors du décodage BP, tandis que la seconde consiste à entraîner le BP-RNN sur les motifs d'erreurs possédant les types absorbants les plus présents dans les échecs du BP.

B.1 Entraînement sur les échecs du BP

Dans cette section, nous proposons d'entraîner le BP-RNN sur les mots bruités pour lesquels le BP ne converge pas vers un mot de code. L'objectif est ainsi de forcer le BP-RNN à apprendre le décodage de motifs d'erreurs compliqués pour le BP. Ceci contraste par conséquent avec l'entraînement standard, dont le jeu d'entraînement contient essentiellement des mots bruités facilement décodés par le BP.

Pour construire le jeu d'entraînement du BP-RNN, nous considérons la transmission du mot de code tout zéro mappé sur une BPSK à travers un canal AWGN. Un mot bruité du jeu d'entraînement est obtenu en suivant les étapes suivantes :

(1) Génération d'un échantillon de bruit $\mathbf{z} := [z_1, \ldots, z_N]$, avec $z_n \sim \mathcal{N}(0, \sigma^2 = 1)$, $n = 1, \ldots, N$.

(2) Choisir un SNR assez grand (ou de manière équivalente un σ_0 assez petit) tel que le mot bruité $\mathbf{y} = [y_n = 1 + \sigma_0 z_n]_{n=1}^N$ soit décodé par le BP. En pratique, si nous supposons qu'au moins l'un des z_n est négatif, nous choisissons $\sigma_0 = -\frac{1}{\min_{n\in\{1,\ldots,N\}} z_n}$. Sinon, nous recommençons à l'étape (1).

(3) Décrementer graduellement la valeur du jusqu'à obtenir un échec du BP :

- $\mathrm{SNR}_i = \mathrm{SNR}_{i-1} - \delta$, avec δ un pas décremental du SNR fixé.
- $\sigma_i = \sqrt{10^{-\frac{\mathrm{SNR}_i}{10}}}$.
- $\mathbf{y} = [y_n = 1 + \sigma_i z_n]_{n=1}^N$.
- Décodage de $\mathbf{y}$ avec le BP.
- Si le BP échoue, le SNR d'échec est $\mathrm{SNR}_F = \mathrm{SNR}_i$ et cette boucle s'arrête.

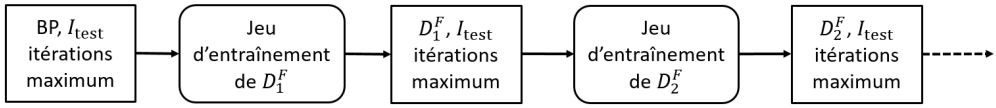


Figure B.1 – Chaîne d'entraînement en cascade.

(4) Ajouter le mot bruité $\mathbf{y} = [y_n = 1 + \sigma_F z_n]_{n=1}^N$ au jeu d'entraînement.

Ce processus est répété autant de fois que nécessaire jusqu'à obtenir la taille du jeu d'entraînement désirée. Le décodeur BP-RNN entraîné sur ce jeu d'entraînement est noté D_1^F.

Notons que ce processus peut être réitéré avec un second décodeur BP-RNN D_2^F, qui sera entraîné sur un jeu d'entraînement contenant les échecs du BP-RNN D_1^F. Un entraînement en cascade à la manière de [17] est ainsi envisageable. Le schéma bloc de la figure B.1 illustre cette approche d'entraînement. Les décodeurs peuvent ensuite être organisés soit en série (l'ordre étant donné par la place dans la chaîne d'entraînement en cascade), soit en parallèle.

Nous évaluons ensuite le taux d'erreur paquet de D_1^F pour le Code-1, de taille $N = 64$ et de rendement de codage $R_c = 0.5$. Les échecs du BP pour générer son jeu d'entraînement sont calculés pour 50 itérations de décodage maximum. Cette valeur nous assure que le jeu d'entraînement de D_1^F contient majoritairement des mots bruités pour lesquels effectuer des itérations supplémentaires de BP ne permettrait pas de converger vers un mot de code. D_1^F est entraîné puis testé avec ce même nombre d'itérations maximum. La taille du jeu d'entraînement est fixée à 100000 pour obtenir un nombre suffisant de mots bruités. Les paramètres d'entraînement Keras sont les mêmes que ceux du tableau 2.4.

Les résultats correspondants sont montrés dans la figure B.2. Nous affichons également à titre de comparaison la performance du BP et du BP-RNN [10], pour 10 itérations de décodage. Nous remarquons que le jeu d'entraînement sur les échecs du BP n'induit quasiment aucun gain par rapport au BP. La même observation a été faite avec un entraînement en cascade de 3 BP-RNNs. Enfin, un décodeur $D_1^F(I_{\text{test}} = 10)$ entraîné sur les échecs du BP pour 10 itérations de décodage maximum fournit une performances proche de celle de BP($I_{\text{test}} = 10$).

La raison de l'échec de cette approche réside dans la trop grande diversité des motifs d'erreurs dans le jeu d'entraînement. En effet, 1544 types absorbants sont présents dans les motifs du jeu d'entraînement. De nombreuses structures de graphes sont par conséquent représentées dans le jeu d'entraînement de D_1^F. De plus, ce sont des motifs d'erreurs difficiles pour le BP. La tâche d'apprentissage est ainsi trop compliquée pour le décodeur BP-RNN D_1^F. Ces résultats fournissent donc une justification supplémentaire pour classifier les motifs selon leur sous-graphe induit.

B.2 Entraînement sur les classes les plus présentes dans les échecs du BP

Afin de prendre en compte la diversité des motifs d'erreurs mettant en difficulté le BP, nous proposons un entraînement sur les motifs possédant les types absorbants les plus

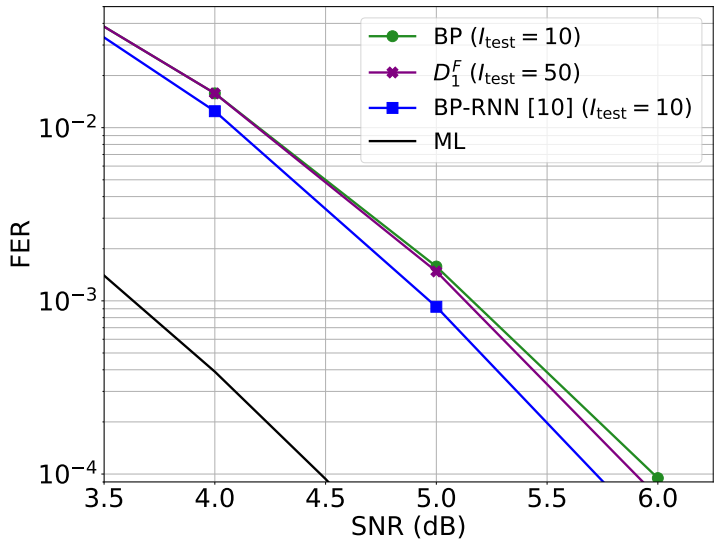


Figure B.2 – Résultats FER du Code-1, D_1^F.

présents dans les échecs du BP.

Pour réaliser ceci, nous générons tout d'abord un jeu de test contenant des mots bruités pour lesquels le BP ne converge pas vers un mot de code (même procédure que pour la génération de $\mathcal{T}_{\text{BP-OSD}}$ du paragraphe 4.4.2.1). Pour chaque mot bruité de ce jeu, le motif d'erreurs et son type absorbant sont déterminés. Le nombre d'occurrences de chaque type absorbant relevé est ensuite calculé. Nous considérons ensuite uniquement les Z types absorbants les plus récurrents, afin de réduire le nombre de BP-RNN à entraîner. Pour construire le jeu d'entraînement correspondant à chaque type absorbant ν-(ω, ε) trouvé, nous générons aléatoirement un certain nombre d'erreurs de taille ν au SNR d'entraînement souhaité, puis nous calculons son type absorbant. Si le type absorbant est égal à ν-(ω, ε), alors le mot bruité est ajouté au jeu d'entraînement. Sinon, un nouveau motif de ν erreurs est généré. Ce processus est répété jusqu'à obtenir la taille du jeu d'entraînement désirée. Il est important de préciser que la construction de ce jeu d'entraînement ne requiert pas le calcul a priori de tous les motifs ayant un type absorbant ν-(ω, ε). Toutefois, la construction est plutôt fastidieuse et longue par définition. Enfin, les Z BP-RNNs entraînés sont organisés en une architecture parallèle. Le choix de Z revient donc à un compromis performance/complexité.

Les résultats numériques de cette approche de spécialisation sont discutés ci-dessous pour le Code-2 ($N = 128, K = 64$). Pour chaque valeur de SNR comprise entre 1 dB et 6 dB, avec un pas de 1 dB, nous calculons 1000 mots bruités donnant un échec au décodage de BP($I_{\text{test}} = 10$). Nous avons choisi $Z = 30$ et nous notons par $\mathcal{D}_{30}^{\text{Pres}}$ la diversité de 30 BP-RNNs spécialisés sur les 30 types absorbants les plus récurrents . Les autres paramètres d'entraînement sont ceux donnés par le tableau 3.2. De plus, nous avons fixé $I_{\text{train}} = 10$ et $I_{\text{test}} = 10$ pour chaque BP-RNN de $\mathcal{D}_{30}^{\text{Pres}}$.

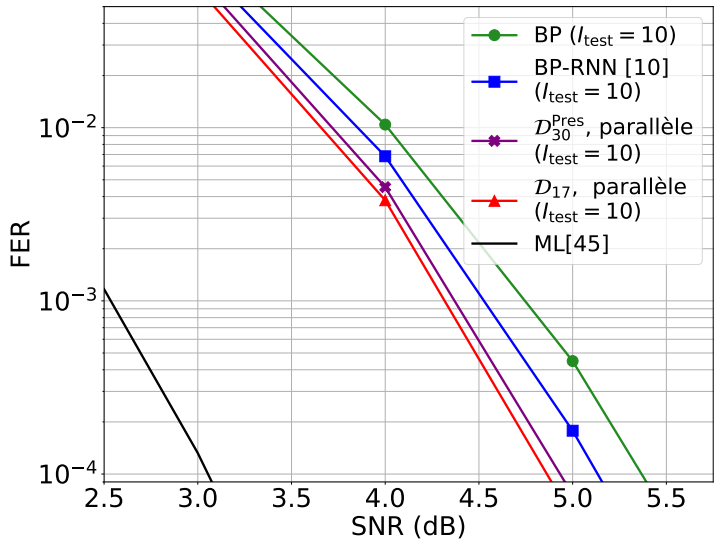


Figure B.3 – Résultats FER du Code-2, $\mathcal{D}_{30}^{\text{Pres}}$.

La figure B.3 illustre la performance FER de l'architecture parallèle de $\mathcal{D}_{30}^{\text{Pres}}$. En guise de références, nous affichons les performances du BP($I_{\text{test}} = 10$) et du BP-RNN [10]($I_{\text{test}} = 10$). Nous avons également calculé la sélection sous-optimale de 17 BP-RNNs entraînés sur les ensembles absorbants possédant une taille $\nu \leq \nu_{\max} = 7$. La diversité $\mathcal{D}_{17}$ est organisée en une architecture parallèle. Nous constatons que l'architecture parallèle de $\mathcal{D}_{30}^{\text{Pres}}$ obtient un gain de 0.2 dB par rapport au BP-RNN [10]. Toutefois, malgré un nombre plus important de décodeurs, $\mathcal{D}_{30}^{\text{Pres}}$ n'induit pas de meilleures performances que la diversité $\mathcal{D}_{17}$. Il est donc préférable de calculer les ensembles absorbants puis de construire la diversité correspondante de BP-RNNs afin d'obtenir un meilleur compromis performance/complexité.

Annexe C

Analyse des poids optimisés pour le post-traitement OSD du BP

Nous complétons dans cette annexe les résultats numériques obtenus dans la section 4.4, en étudiant les poids optimisés du neurone simple entraîné avec la fonction de coût focale.

C.1 Poids optimisés

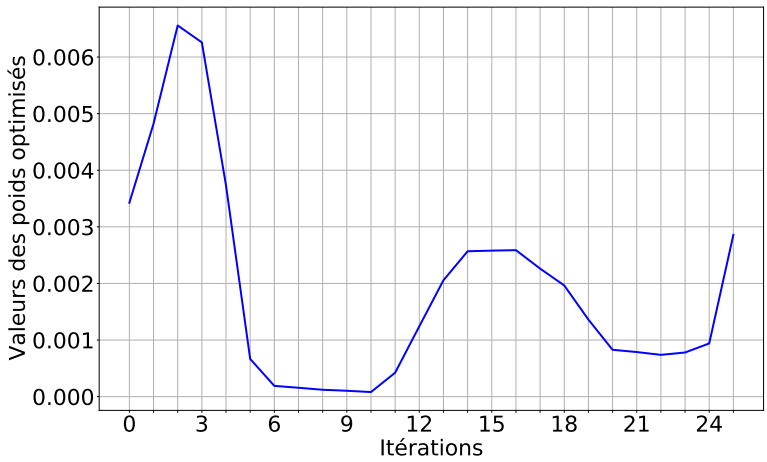
Nous affichons dans la figure C.1 les valeurs des poids optimisés $w^{(i)}$ ($i \in [0, I_{\text{test}}]$) de l'équation (4.6), pour le Code-2 (a) et le Code-5 (b). Nous montrons également dans figure C.2 le nombre d'erreurs résiduelles après un post-traitement OSD-p ($p \leq 2$) selon les vecteurs de LLRs a posteriori $\tilde{\mathbf{L}}^{(i)}$ pour les deux codes à des fins d'analyses. Le nombre maximal d'itérations est fixé à $I_{\text{test}} = 25$, tandis que le SNR de test vaut 3.5 dB pour le Code-2 et 3.5 dB pour le Code-5. Les post-traitements OSDs sont évalués sur 10000 non convergences du BP vers un mot de code.

Pour le Code-2, nous constatons en comparant les deux figures que les poids les plus faibles sont bien affectés aux vecteurs $\tilde{\mathbf{L}}^{(i)}$ les plus mauvais avec l'OSD ($i \in [5, 9]$), tandis que les poids les plus grands correspondent bien aux premières itérations ($0 < i \leq 3$). Nous remarquons toutefois que le poids affecté à $\tilde{\mathbf{L}}^{(0)}$ est le deuxième plus grand, malgré des performances moins bonnes de l'OSD-p avec $\tilde{\mathbf{L}}^{(0)}$. Ces observations coïncident avec celles établies dans le paragraphe 4.4.2.2. De plus, l'allure de la courbe des poids selon i correspond bien à celle de la fonction de coût focale illustrée dans la figure 4.15. Une pondération plus cohérente avec la performance de l'OSD avec $\tilde{\mathbf{L}}^{(0)}$ serait donc à considérer pour le calcul d'une fiabilité donnée par un neurone.

Les conclusions sont similaires pour le Code-5. Les meilleurs $\tilde{\mathbf{L}}^{(i)}$ (itérations 1 à 4 inclus) avec l'OSD sont bien favorisés par la pondération par rapport aux autres $\tilde{\mathbf{L}}^{(i)}$. Le problème de pondération de $\tilde{\mathbf{L}}^{(0)}$ examiné pour le Code-2 est aussi présent.

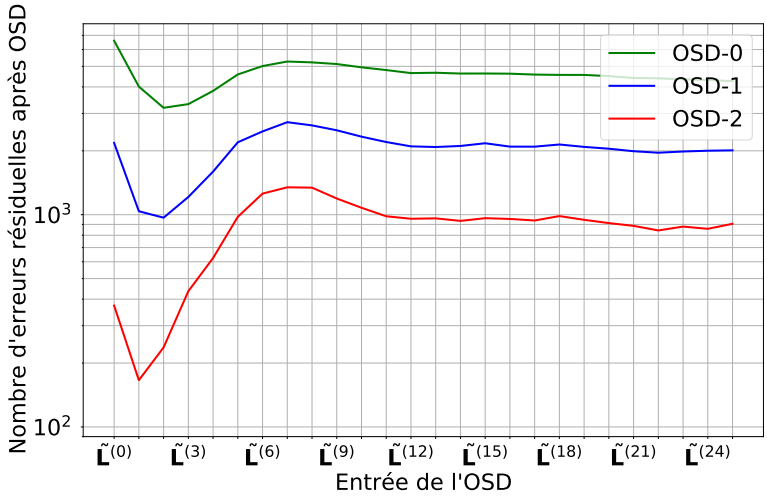
Ces résultats nous permettent ainsi de bien vérifier que l'optimisation des poids par la fonction de coût focale a bien capturé le comportement global de l'OSD selon les $\tilde{\mathbf{L}}^{(i)}$ (avec l'exception de $\tilde{\mathbf{L}}^{(0)}$).

(a) Code-2 (SNR = 3.5 dB)

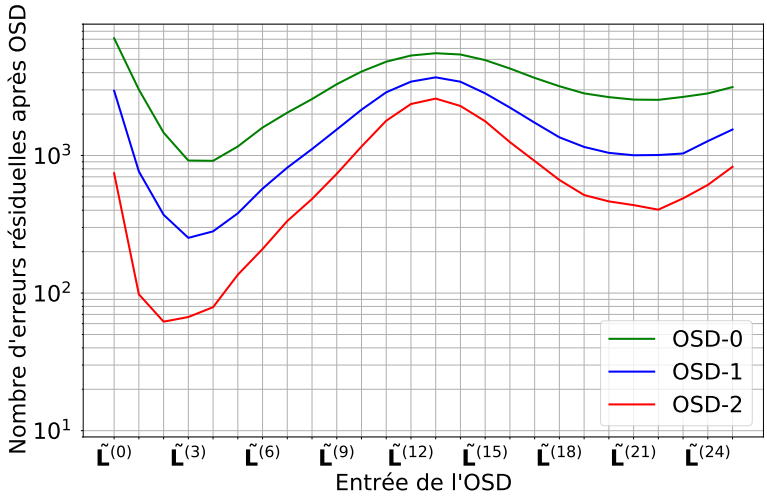


(b) Code-5 (SNR = 2.5 dB)

Figure C.1 – Poids optimisés $w^{(i)}$ vs l'itération i ($i \in [0, I_{\text{test}}]$).



(a) Code-2 (SNR = 3.5 dB)



(b) Code-5 (SNR = 2.5 dB)

Figure C.2 – Nombre d'erreurs résiduelles sur 10000 échecs du BP($I_{\text{test}} = 25$) après un post-traitement OSD.

Bibliographie

[1] Mojtaba Vaezi, Amin Azari, Saeed R Khosravirad, Mahyar Shirvanimoghaddam, M Mahdi Azari, Danai Chasaki, and Petar Popovski. Cellular, wide-area, and non-terrestrial iot : A survey on 5G advances and the road toward 6G. *IEEE Communications Surveys and Tutorials* **24** (2), 1117–1174 (2022). Cited on page/s xxi.

[2] Ali Zaidi, Fredrik Athley, Jonas Medbo, Ulf Gustavsson, Giuseppe Durisi, and Xiaoming Chen. 5G physical layer : principles, models and technology components. Academic Press (2018). Cited on page/s xxi.

[3] Yury Polyanskiy, H Vincent Poor, and Sergio Verdú. Channel coding rate in the finite blocklength regime. *IEEE Transactions on Information Theory* **56** (5), 2307–2359 (2010). Cited on page/s xxi.

[4] Mustafa Cemil Coşkun, Giuseppe Durisi, Thomas Jerkovits, Gianluigi Liva, William Ryan, Brian Stein, and Fabian Steiner. Efficient error-correcting codes in the short blocklength regime. *Physical Communication* **34**, 66–79 (2019). Cited on page/s xxi.

[5] R. G. Gallager. Low density parity check codes. MIT Press, Cambridge (1963). Research Monograph series. Cited on page/s xxi, 2.

[6] R. Tanner. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory* **27** (5), 533–547 (1981). Cited on page/s xxi, 2.

[7] T.J. Richardson, M.A. Shokrollahi, and R.L. Urbanke. Design of capacity-approaching irregular low-density parity-check codes. *IEEE Transactions on Information Theory* **47** (2), 619–637 (2001). Cited on page/s xxi, 3, 7.

[8] N. Wiberg. Codes and decoding on general graphs. *PhD thesis*. Department of Electrical Engineering, Linköping University, Sweden (1996). Cited on page/s xxi, 8.

[9] Eliya Nachmani, Yair Be'ery, and David Burshtein. Learning to decode linear codes using deep learning. In *54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)* pages 341–346 (2016). doi: 10.1109/ALLERTON.2016.7852251. Cited on page/s xxi, 31.

[10] Eliya Nachmani, Elad Marciano, Loren Lugosch, Warren J Gross, David Burshtein, and Yair Be'ery. Deep learning methods for improved decoding of linear codes. *IEEE Journal of Selected Topics in Signal Processing* **12** (1), 119–131 (2018). Cited on page/s xxii, 2, 15, 27, 28, 29, 30, 31, 33, 34, 38, 46, 48, 49, 50, 51, 52, 53, 58, 59, 65, 66, 72, 73, 74, 77, 79, 98, 99, 121, A-1, A-6, A-8.

[11] Andreas Buchberger, Christian Häger, Henry D Pfister, Laurent Schmalen, and Alexandre Graell i Amat. Pruning and quantizing neural belief propagation decoders. *IEEE Journal on Selected Areas in Communications* **39** (7), 1957–1966 (2020). Cited on page/s xxii, 2, 27, 33, 34, 35, 50, 70, 77, 81, 83, 85, 99.

[12] Nghia Doan, Seyyed Ali Hashemi, Elie Ngomseu Mambou, Thibaud Tonnellier, and Warren J Gross. Neural belief propagation decoding of CRC-polar concatenated codes. In *IEEE International Conference on Communications (ICC)* pages 1–6 (2019). Cited on page/s xxii.

[13] Mengke Lian, Fabrizio Carpi, Christian Häger, and Henry D Pfister. Learned belief-propagation decoding with simple scaling and SNR adaptation. In *IEEE International Symposium on Information Theory (ISIT)* pages 161–165 (2019). Cited on page/s xxii, 29, 30, 31, 51, 52.

[14] Xiangyu Chen and Min Ye. Cyclically equivariant neural decoders for cyclic codes. *arXiv preprint arXiv :2105.05540* (2021). Cited on page/s xxii.

[15] Maximilian Stark, Gerhard Bauch, Linfang Wang, and Richard D Wesel. Information bottleneck decoding of rate-compatible 5G-LDPC codes. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)* pages 1–6. IEEE (2020). Cited on page/s xxii.

[16] Xin Xiao, Bane Vasić, Ravi Tandon, and Shu Lin. Designing finite alphabet iterative decoders of LDPC codes via recurrent quantized neural networks. *IEEE Transactions on Communications* **68** (7), 3963–3974

(2020). Cited on page/s xxii, xxiii, 33, 38, 42, 119.

[17] Xin Xiao, Nithin Raveendran, Bane Vasić, Shu Lin, and Ravi Tandon. FAID diversity via neural networks. In *11th International Symposium on Topics in Coding (ISTC)* pages 1–5 (2021). Cited on page/s xxii, xxiii, 2, 32, 33, 38, 42, 119, A-6.

[18] Eliya Nachmani and Lior Wolf. Hyper-graph-network decoders for block codes. *Advances in Neural Information Processing Systems* **32** (2019). Cited on page/s xxii.

[19] Sebastian Cammerer, Jakob Hoydis, Fayçal Aït Aoudia, and Alexander Keller. Graph neural networks for channel decoding. In *2022 IEEE Globecom Workshops (GC Wkshps)* pages 486–491. IEEE (2022). Cited on page/s xxii.

[20] Kou Tian, Chentao Yue, Changyang She, Yonghui Li, and Branka Vucetic. A scalable graph neural network decoder for short block codes. *arXiv preprint arXiv :2211.06962* (2022). Cited on page/s xxii.

[21] Loren Lugosch and Warren J Gross. Neural offset min-sum decoding. In *2017 IEEE International Symposium on Information Theory (ISIT)* pages 1361–1365. IEEE (2017). Cited on page/s xxii.

[22] Linfang Wang, Sean Chen, Jonathan Nguyen, Divsalar Dariush, and Richard Wesel. Neural-network-optimized degree-specific weights for LDPC MinSum decoding. *arXiv preprint arXiv :2107.04221* (2021). Cited on page/s xxii.

[23] Jincheng Dai, Kailin Tan, Zhongwei Si, Kai Niu, Mingzhe Chen, H Vincent Poor, and Shuguang Cui. Learning to decode protograph LDPC codes. *IEEE Journal on Selected Areas in Communications* **39** (7), 1983–1999 (2021). Cited on page/s xxii.

[24] Nemin Shah and Yash Vasavada. Neural layered decoding of 5G LDPC codes. *IEEE Communications Letters* **25** (11), 3590–3593 (2021). Cited on page/s xxii.

[25] Beeshanga Abewardana Jayawickrama and Ying He. Improved layered normalized min-sum algorithm for 5G NR LDPC. *IEEE Wireless Communications Letters* **11** (9), 2015–2018 (2022). Cited on page/s xxii.

[26] Salman Habib, Allison Beemer, and Jörg Kliewer. Belief propagation decoding of short graph-based channel codes via reinforcement learning. *IEEE Journal on Selected Areas in Information Theory* **2** (2), 627–640 (2021). Cited on page/s xxii.

[27] Salman Habib, Allison Beemer, and Joerg Kliewer. Reldec : Reinforcement learning-based decoding of moderate length LDPC codes. *arXiv preprint arXiv :2112.13934* (2021). Cited on page/s xxii.

[28] M.P.C. Fossorier and Shu Lin. Soft-decision decoding of linear block codes based on ordered statistics. *IEEE Transactions on Information Theory* **41** (5), 1379–1396 (1995). doi: 10.1109/18.412683. Cited on page/s xxii, 14, 83, 88, 90.

[29] M.P.C. Fossorier. Iterative reliability-based decoding of low-density parity check codes. *IEEE Journal on Selected Areas in Com.* **19** (5), 908–917 (2001). Cited on page/s xxii, 88, 90, 91, 96.

[30] Marco Baldi, Nicola Maturo, Enrico Paolini, and Franco Chiaraluce. On the applicability of the most reliable basis algorithm for LDPC decoding in telecommand links. In *Int. Conference on Inf. and Com. Systems (ICICS)* pages 1–6. IEEE (2015). Cited on page/s xxii, 88, 92, 96, 107.

[31] K. Watanabe, R. Kaguchi, and T. Shinoda. Shortened LDPC codes accelerate OSD decoding performance. *EURASIP Journal on Wireless Communications and Networking* **2021** (1), 1–18 (2021). Cited on page/s xxii, 88, 92, 107.

[32] M. Jiang, C. Zhao, E. Xu, and L. Zhang. Reliability-based iterative decoding of LDPC codes using likelihood accumulation. *IEEE communications letters* **11** (8), 677–679 (2007). Cited on page/s xxii, 88, 92, 106, 107.

[33] Lara Dolecek, Pamela Lee, Zhengya Zhang, Venkat Anantharam, Borivoje Nikolic, and Martin Wainwright. Predicting error floors of structured LDPC codes : Deterministic bounds and estimates. *IEEE Journal on Selected Areas in Communications* **27** (6), 908–917 (2009). Cited on page/s xxiii, 11, 12, 39.

[34] RG Gallager and Low-Density Parity-Check Codes. Research monograph series (1963). Cited on page/s 4, 122.

[35] David Declercq, Marc Fossorier, and Ezio Biglieri. Channel coding : Theory, algorithms, and applications : Academic press library in mobile and wireless communications. Academic Press (2014). Cited on page/s 6, 8.

[36] Benjamin Smith, Masoud Ardakani, Wei Yu, and Frank R Kschischang. Design of irregular LDPC codes

with optimized performance-complexity tradeoff. *IEEE Transactions on Communications* **58** (2), 489–499 (2010). Cited on page/s 7, 69, 83.

[37] Xiao-Yu Hu, Evangelos Eleftheriou, and Dieter-Michael Arnold. Regular and irregular progressive edge-growth tanner graphs. *IEEE transactions on information theory* **51** (1), 386–398 (2005). Cited on page/s 8, 9.

[38] David JC MacKay. Good error-correcting codes based on very sparse matrices. *IEEE transactions on Information Theory* **45** (2), 399–431 (1999). Cited on page/s 10.

[39] Tom Richardson, Amin Shokrollahi, and Rüdiger Urbanke. Design of provably good low-density parity check codes. In *2000 IEEE International Symposium on Information Theory (Cat. No. 00CH37060)* page 199. IEEE (2000). Cited on page/s 10.

[40] Tom Richardson. Error floors of LDPC codes. In *Proceedings of the Annual Allerton Conference on Communication Control and Computing* volume 41 pages 1426–1435 (2003). Cited on page/s 10.

[41] Brendan J Frey, R Kotter, and Alex Vardy. Skewness and pseudocodewords in iterative decoding. In *IEEE International Symposium on Information Theory* page 148 (1998). Cited on page/s 11.

[42] David JC MacKay and Michael S Postol. Weaknesses of Margulis and Ramanujan-Margulis low-density parity-check codes. *Electronic Notes in Theoretical Computer Science* **74**, 97–104 (2003). Cited on page/s 11.

[43] Lara Dolecek, Zhengya Zhang, Venkat Anantharam, Martin J Wainwright, and Borivoje Nikolic. Analysis of absorbing sets and fully absorbing sets of array-based LDPC codes. *IEEE Transactions on Information Theory* **56** (1), 181–201 (2010). Cited on page/s 11, 58.

[44] Short block length LDPC codes for TC synchronization and channel codding (CCSDS 231.1-O-1). Consultative Committee for Space Data Systems (CCSDS), Technical Report (April 2015). Cited on page/s 12, 13, 112.

[45] Michael Helmling, Stefan Scholl, Florian Gensheimer, Tobias Dietz, Kira Kraft, Stefan Ruzika, and Norbert Wehn. Database of channel codes and ML simulation results. www.uni-kl.de/channel-codes (2019). Cited on page/s 14, 99, 113.

[46] Xiaohu You, Chuan Zhang, Xiaosi Tan, Shi Jin, and Hequan Wu. AI for 5G : Research directions and paradigms. *Science China Information Sciences* **62**, 1–13 (2019). Cited on page/s 15.

[47] Tianqi Wang, Chao-Kai Wen, Hanqing Wang, Feifei Gao, Tao Jiang, and Shi Jin. Deep learning for wireless physical layer : Opportunities and challenges. *China Communications* **14** (11), 92–111 (2017). Cited on page/s 15.

[48] Timothy O'shea and Jakob Hoydis. An introduction to deep learning for the physical layer. *IEEE Transactions on Cognitive Communications and Networking* **3** (4), 563–575 (2017). Cited on page/s 15.

[49] Francois Chollet. Deep learning with python. Simon and Schuster (2021). Cited on page/s 15.

[50] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, *et al.* Learning internal representations by error propagation (1985). Cited on page/s 17.

[51] Kevin P Murphy. Probabilistic machine learning : an introduction. MIT press (2022). Cited on page/s 19, 22.

[52] TY. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *Proc. of the IEEE Int. Conference on Computer Vision* (2017). Cited on page/s 22, 103.

[53] Tijmen Tieleman, Geoffrey Hinton, *et al.* Lecture 6.5-rmsprop : Divide the gradient by a running average of its recent magnitude. *COURSERA : Neural networks for machine learning* **4** (2), 26–31 (2012). Cited on page/s 30, 46, 72, 112, A-1.

[54] Diederik P Kingma and Jimmy Ba. Adam : A method for stochastic optimization. *arXiv preprint arXiv :1412.6980* (2014). Cited on page/s 30, A-1.

[55] David Declercq, Bane Vasic, Shiva Kumar Planjery, and Erbao Li. Finite alphabet iterative decoders – Part II : Towards guaranteed error correction of LDPC codes via iterative decoder diversity. *IEEE Transactions on Communications* **61** (10), 4046–4057 (2013). Cited on page/s 33, 34, 38, 42, 119.

[56] Nithin Raveendran, David Declercq, and Bane Vasić. A sub-graph expansion-contraction method for error floor computation. *IEEE Transactions on Communications* **68** (7), 3984–3995 (2020). Cited on page/s 33.

[57] R. Tanner, D. Sridhara, and T. Fuja. A class of group-structured LDPC codes. In *Proc. ISTA* pages 365–370.

Citeseer (2001). Cited on page/s 33, 112.

[58] Sun-Chong Wang. Artificial neural network. In *Interdisciplinary computing in Java programming* pages 81–100. Springer (2003). Cited on page/s 41.

[59] Andreas Buchberger, Christian Häger, Henry D Pfister, Laurent Schmalen, and Alexandre Graell Amat. Pruning neural belief propagation decoders. In *IEEE International Symposium on Information Theory (ISIT)* pages 338–342. IEEE (2020). Cited on page/s 51, 81, 122.

[60] Mehdi Karimi and Amir H Banihashemi. On characterization of elementary trapping sets of variable-regular LDPC codes. *IEEE Transactions on Information Theory* **60** (9), 5188–5203 (2014). Cited on page/s 59.

[61] Yoones Hashemi and Amir H Banihashemi. On characterization and efficient exhaustive search of elementary trapping sets of variable-regular LDPC codes. *IEEE Communications Letters* **19** (3), 323–326 (2015). Cited on page/s 59.

[62] Hossein Falsafain and Sayyed Rasoul Mousavi. Exhaustive enumeration of elementary trapping sets of an arbitrary Tanner graph. *IEEE Communications Letters* **20** (9), 1713–1716 (2016). Cited on page/s 59.

[63] Shadi Abu-Surra, David Declercq, Dariush Divsalar, and William E Ryan. Trapping set enumerators for specific LDPC codes. In *IEEE Information Theory and Applications Workshop (ITA)* pages 1–5. IEEE (2010). Cited on page/s 59.

[64] Lara Dolecek, Zhengya Zhang, Venkat Anantharam, Martin J Wainwright, and Borivoje Nikolic. Analysis of absorbing sets and fully absorbing sets of array-based LDPC codes. *IEEE Transactions on Information Theory* **56** (1), 181–201 (2009). Cited on page/s 59.

[65] Mehdi Karimi and Amir H Banihashemi. Efficient algorithm for finding dominant trapping sets of LDPC codes. *IEEE Transactions on Information Theory* **58** (11), 6942–6958 (2012). Cited on page/s 59.

[66] Chih-Chun Wang, Sanjeev R Kulkarni, and H Vincent Poor. Finding all small error-prone substructures in LDPC codes. *IEEE Transactions on Information Theory* **55** (5), 1976–1999 (2009). Cited on page/s 59.

[67] Gyu Bum Kyung and Chih-Chun Wang. Exhaustive search for small fully absorbing sets and the corresponding low error-floor decoder. In *IEEE International Symposium on Information Theory (ISIT)* pages 739–743. IEEE (2010). Cited on page/s 59.

[68] Xiaojie Zhang and Paul H Siegel. Efficient algorithms to find all small error-prone substructures in LDPC codes. In *IEEE Global Telecommunications Conference (GLOBECOM)* pages 1–6. IEEE (2011). Cited on page/s 59.

[69] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing* **1** (2), 146–160 (1972). Cited on page/s 60.

[70] Marvin Geiselhart, Moustafa Ebada, Ahmed Elkelesh, Jannis Clausius, and Stephan Ten Brink. Automorphism ensemble decoding of quasi-cyclic LDPC codes by breaking graph symmetries. *IEEE Communications Letters* (2022). Cited on page/s 79.

[71] W. Zhou and M. Lentmaier. Improving short-length LDPC codes with a CRC and iterative ordered statistic decoding. In *CISS* pages 1–6 (2019). Cited on page/s 90.

[72] S. Gounai and T. Ohtsuki. Decoding algorithms based on oscillation for low-density parity check codes. *IEICE Trans. on fundamentals of electronics, Com. and computer sciences* **88** (8), 2216–2226 (2005). Cited on page/s 91.

[73] M. P. C. Fossorier and S. Lin. Computationally efficient soft-decision decoding of linear block codes based on ordered statistics. *IEEE Tr. on Information Theory* **42** (3), 738–750 (1996). Cited on page/s 93, 94.

[74] C. Yue, M. Shirvanimoghaddam, Y. Li, and B. Vucetic. Segmentation-discarding ordered-statistic decoding for linear block codes. In *2019 IEEE GLOBECOM* pages 1–6. IEEE (2019). Cited on page/s 94.

[75] Changhyeon Kim, Dongyun Kam, Seokki Kim, Giyoon Park, and Youngjoo Lee. Simplified ordered statistic decoding for short-length linear block codes. *IEEE Communications Letters* **26** (8), 1720–1724 (2022). Cited on page/s 94.

[76] D. Wu, Y. Li, X. Guo, and Y. Sun. Ordered statistic decoding for short polar codes. *IEEE Com. Letters* **20** (6) (2016). Cited on page/s 94.

[77] Brian A LaMacchia and Andrew M Odlyzko. Solving large sparse linear systems over finite fields. In *Proc. of Annual Int. Cryptology Conf. on Advances in Cryptology (CRYPT0'90)* pages 109–133 (1990). Cited on page/s

95.

[78] T.J. Richardson and R.L. Urbanke. Efficient encoding of low-density parity-check codes. *IEEE Transactions on Information Theory* **47** (2), 638–656 (2001). Cited on page/s 96.

[79] D. Burshtein and G. Miller. Efficient maximum-likelihood decoding of LDPC codes over the binary erasure channel. *IEEE Trans. on Information Theory* **50** (11), 2837–2844 (2004). Cited on page/s 96.

[80] D. J. C. MacKay. Encyclopedia of sparse graph codes. http://www.inference.org.uk/mackay/codes/data.html (2008). Cited on page/s 112.

[81] Valentin Savin. Gradient descent bit-flipping decoding with momentum. In *2021 11th International Symposium on Topics in Coding (ISTC)* pages 1–5. IEEE (2021). Cited on page/s 122.

[82] Fabrizio Carpi, Christian Häger, Marco Martalò, Riccardo Raheli, and Henry D Pfister. Reinforcement learning for channel coding : Learned bit-flipping decoding. In *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)* pages 922–929. IEEE (2019). Cited on page/s 122.

[83] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* volume 2 pages 729–734. IEEE (2005). Cited on page/s 122.

[84] Ashok V Makkuva, Xiyang Liu, Mohammad Vahid Jamali, Hessam Mahdavifar, Sewoong Oh, and Pramod Viswanath. Ko codes : inventing nonlinear encoding and decoding for reliable wireless communication via deep-learning. In *International Conference on Machine Learning* pages 7368–7378. PMLR (2021). Cited on page/s 122.

[85] Guillaume Larue, Louis-Adrien Dufrene, Quentin Lampin, Hadi Ghauch, and Ghaya Rekaya-Ben Othman. Neural belief propagation auto-encoder for linear block code design. *IEEE Transactions on Communications* **70** (11), 7250–7264 (2022). Cited on page/s 122.

[86] David Bol, Thibault Pirson, and Rémi Dekimpe. Moore's law and ict innovation in the anthropocene. In *2021 Design, Automation and Test in Europe Conference and Exhibition (DATE)* pages 19–24. IEEE (2021). Cited on page/s 123.

[87] Philippe Ciblat. À propos du mimo massif dans un contexte de sobriété numérique. In *GRETSI* (2022). Cited on page/s 123.