Titre: Title:	Codes convolutionnels doublement orthogonaux récursifs : analyse et recherche des nouveaux codes, évaluation des performances
Auteur: Author:	Laurent Rouleau
Date:	2008
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Rouleau, L. (2008). Codes convolutionnels doublement orthogonaux récursifs : analyse et recherche des nouveaux codes, évaluation des performances [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/8362/

UNIVERSITÉ DE MONTRÉAL

CODES CONVOLUTIONNELS DOUBLEMENT ORTHOGONAUX RÉCURSIFS : ANALYSE ET RECHERCHE DES NOUVEAUX CODES, ÉVALUATION DES PERFORMANCES

LAURENT ROULEAU

DÉPARTEMENT DE GÉNIE ÉLECTRIQUE

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(TÉLÉCOMMUNICATIONS)

JUILLET 2008

Your file Votre référence
ISBN: 978-0-494-46077-1
Our file Notre référence
ISBN: 978-0-494-46077-1

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

CODES CONVOLUTIONNELS DOUBLEMENT ORTHOGONAUX RÉCURSIFS : ANALYSE ET RECHERCHE DES NOUVEAUX CODES, ÉVALUATION DES PERFORMANCES

présenté par: ROULEAU Laurent

en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de:

M. FRIGON Jean-François, Ph.D., président

M. HACCOUN David, Ph.D., membre et directeur de recherche

M. CARDINAL Christian, Ph.D., membre

REMERCIEMENTS

Le travail de recherche présenté dans ce mémoire n'aurait pu être accompli sans le soutien de plusieurs personnes à qui je dédie ces quelques lignes.

Je souhaite en premier lieu remercier mon directeur de recherche, le D[r] David Haccoun, pour son aide financière accordée tout au long de mon travail de recherche.

J'exprime aussi ma gratitude à l'École Supérieure d'Électricité (Supélec) qui a permis cet échange double diplomant et en particulier à M[me] De Swarte.

Merci aussi à tous mes partenaires de laboratoire, et en particulier à Éric Roy, pour toutes les discussions et idées qu'on a pu partager. Je pense aussi à David, Julien, Damien, Bertrand, Zouheir, Samuel et Benjamin.

Mes pensées vont enfin à mes parents, pour leur soutien inconditionnel, ainsi qu'à mes amis et colocataires de Montréal grâce à qui ce séjour à l'étranger restera une expérience inoubliable.

RÉSUMÉ

L'objectif du travail de recherche présenté dans ce mémoire est d'étudier une classe particulière des codes convolutionnels doublement orhogonaux, (CSO^2C) : les codes doublement orthogonaux récursifs, ($R-CSO^2C$). En effet, ces codes permettent d'obtenir de bonnes performances de correction d'erreurs de transmission à de faibles rapports signal sur bruit, typiquement inférieurs à 2dB, pour pouvoir rivaliser avec les codes dits Turbo. Cette étude est nécessaire, car il a été démontré que les codes doublement orthogonaux non récursifs connus jusqu'à maintenant, bien que performants, le sont à de plus hauts rapports signal sur bruit.

Parmis les codes correcteurs d'erreur connus à ce jour, les codes Turbo sont ceux qui offrent les meilleures performances à de faibles rapports signal sur bruit. En effet, grâce à l'utilisation d'entrelaceurs au niveau du codeur et du décodeur, les codes Turbo maintiennent une indépendance complète des observables tout au long du processus itératif de décodage. Cependant, ces mêmes entrelaceurs sont à l'origine d'un important retard au décodage, aussi appellée latence, ainsi que d'une augmentation de la complexité.

Les codes CSO^2C, eux, garantissent une indépendance des observables sur les deux premières itérations seulement, mais la suppression des entrelaceurs au codage et au décodage permet d'obtenir une latence plus faible au décodage ainsi qu'une complexité plus faible. Les codes $R-CSO^2C$ seront dans ce mémoire définis au sens large ($R-CSO^2C-$

WS) ainsi qu'au sens strict ($R - CSO^2C - SS$). Comme pour les codes non récursifs, le sens large signifie que, contrairement au sens strict, l'indépendance des observables à la deuxième itération du décodage est incomplète.

Pour les deux cas, nous avons défini l'ensemble des conditions que ces codes doivent vérifier. Nous proposons également plusieurs méthodes de construction de ces codes. Cependant, l'objectif dans la recherche de nouveaux codes n'est pas le même. En effet, nous chercherons, au sens large, pour un nombre de connexions donné, à minimiser la longueur des codes afin de réduire au maximum la latence, puisque les performances d'erreur sont directement liées au nombre de connexions. En revanche, au sens strict, il a été observé que les performances d'erreur ne dépendent plus uniquement du nombre de connexions, mais aussi d'autres paramètres tels que la longueur des codes ou la répartition des connexions dans la partie récursive de la matrice de représentation des codes $R - CSO^2C - SS$.

Les performances d'erreur obtenues par simulations ont montré que les codes doublement orthogonaux récursifs au sens large ne sont pas adaptés pour des valeurs de E_b/N_0 inférieures à 3dB, E_b étant l'énergie avec laquelle on émet chaque bit et $N_0/2$ étant la densité spectrale bilatérale de puissance du bruit. En revanche, au sens strict, nous avons pu atteindre une probabilité d'erreur par bit de 10^{-6} avec $E_b/N_0 = 1.3dB$. De plus, les codes récursifs au sens strict les plus courts donnent, pour de plus grandes valeurs de E_b/N_0, de meilleures performances d'erreur que tous les codes doublement orthogonaux connus, avec

une latence au décodage très réduite. Les codes $R - CSO^2C - SS$ sont donc les codes doublement orthogonaux connus les plus performants, que ce soit à de faibles rapports signal sur bruit avec les codes les plus longs, ou à de hauts rapports signal sur bruit avec les codes les plus courts.

ABSTRACT

This thesis presents a particular class of Convolutionnal Self-Doubly Orthogonal Codes (CSO^2C), which are called Recursive-Convolutionnal Self-Doubly Orthogonal Codes ($R-CSO^2C$), in order to obtain good error performances at signal to noise ratios (SNR) lower than 2 dB and thus offering an alternative to Turbo Codes. Indeed, non recursive CSO^2C codes provide good error performances, but only at higher SNR values.

Among the error correcting known codes, Turbo codes are among those claim the best error performances at low SNR values. On the one hand, thanks to the use of interleavers in the coding and decoding parts, Turbo codes provide a complete independence of observables in the iterative decoding process. On the other hand, these interleavers yield a substantial decoding latency and add to the complexity of the communication system.

CSO^2C codes provide the independence of observables only for the first two iterations, but the elimination of interleavers in both the coding and decoding parts decrease the decoding latency and reduce somewhat the system complexity. $R - CSO^2C$ are defined in both the wide and strict senses and are noted $R - CSO^2C - WS$ and $R - CSO^2C - SS$ respectively. As for the non recursive CSO^2C codes, wide sense means that the independence of observables in the second decoding iteration may not be complete.

In both cases, we defined all conditions that $R-CSO^2C-WS$ and $R-CSO^2C-SS$

codes must check. Several construction methods are proposed for recursive codes. But the objectives of the search for these codes are different. On the wide sense, error performances are related to the number of connections of the codes. Hence, for a given number of connections, a search objective is to minimize the spans of the codes. However, for the strict sense codes, error performances depend on several parameters such as the spans of the codes, the total number of connections or the structure of the recursive part of the $R - CSO^2C - SS$ representing matrix .

Error performances provided by simulations show that $R - CSO^2C - WS$ codes are not competitive at SNR values lower than 3dB. However, for the strict sense codes, a bit error probability of 10^{-6} at a SNR value lower than 1.3 dB may be achieved. Moreover, at higher SNR values, shortest $R - CSO^2C - SS$ codes reach better bit error probability than any known CSO^2C codes, with a latency very reduced. Thus, $R - CSO^2C - SS$ codes are the most powerful self-doubly orthogonal codes, whether at low SNR values with the longest codes, or at high SNR values with the shortest ones.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

$BPSK$:	Binary Phase Shift Keying
R :	Taux de codage
m :	Span du code
L :	Latence
d_h :	Distance de Hamming
d_{free} :	Distance libre
$CSOC$:	Codes convolutionnels simplement orthogonaux
CSO^2C :	Codes convolutionnels doublement orthogonaux
$CSO^2C - WS$:	Codes convolutionnels doublement orthogonaux au sens large
$S - CSO^2C - WS$:	Codes convolutionnels doublement orthogonaux au sens large simplifiés
$CSO^2C - SS$:	Codes convolutionnels doublement orthogonaux au sens strict
$R - CSOC$:	Codes convolutionnels simplement orthogonaux récursifs
$R - CSO^2C$:	Codes convolutionnels doublement orthogonaux récursifs
$R - CSO^2C - WS$:	Codes convolutionnels doublement orthogonaux récursifs au sens large
$R - CSO^2C - SS$:	Codes convolutionnels doublement orthogonaux récursifs au sens strict
LRV :	Logarithme du rapport de vraisemblance
SNR :	Rapport signal sur bruit (en dB)
E_b :	Energie par bit

P_e :	Probabilité d'erreur
P_b :	Probabilité d'erreur binaire
$N_0/2$:	Variance du bruit blanc gaussien additif
$\lvert \cdot \rvert$:	Valeur absolue
$\lfloor \cdot \rfloor$:	Partie entière
$\oplus$:	Addition modulo-2
$\diamond$:	Opération add-min

LISTE DES TABLEAUX

LISTE DES ANNEXES

INTRODUCTION

Mise en contexte

Dans les années 1950, Shannon a révolutionné le monde des télécommunications en démontrant que, tant que les bits étaient transmis à travers un canal à un taux inférieur à une certaine limite appelée capacité du canal, il était théoriquement possible d'obtenir une probabilité d'erreur arbitrairement petite, à la condition d'utiliser un codage correcteur d'erreurs approprié [17]. Cependant, il n'a jamais indiqué la manière d'atteindre cette limite théorique. Une véritable course s'est alors engagée dans la recherche de codes correcteurs d'erreurs capables d'arriver de façon pratique à la fameuse capacité.

C'est dans ce contexte que la théorie du codage doublement orthogonal [6], [11] a été mise au point il y a une décennie, en 1997, par les professeurs Haccoun, Cardinal et Gagnon, suite à la découverte des codes Turbo en 1993 par messieurs Berrou et Glavieux [3]. En effet, ces codes Turbo, utilisant un décodage itératif symbole par symbole, ont révolutionné le monde du codage en s'approchant de la limite prédite par Shannon comme personne n'y était parvenu auparavant. Cependant, ces résultats ont été obtenus au prix de l'introduction d'un important retard, dû à la partie importante de ces codeurs et décodeurs : des entrelaceurs. Ces entrelaceurs sont à l'origine de l'indépendance totale des observables au fil des itérations, ce qui permet d'obtenir ces remarquables résultats.

L'idée des professeurs Haccoun, Cardinal et Gagnon a été de garder l'indépendance des observables, au moins sur deux itérations, tout en supprimant l'entrelaceur. Pour cela, ils

ont mis au point une théorie de décodage itératif à seuil où la complexité se trouve dans la construction du code. Ainsi, la principale difficulté consiste en la génération de bons codes doublement orthogonaux, mais leur mise en application, pour des résultats de plus en plus proches des codes Turbo, est d'une complexité bien inférieure. De plus, la suppression des entrelaceurs diminue fortement le retard introduit par les codes Turbo et la complexité des algorithmes de codage et de décodage.

Notre étude s'est intéressée en particulier aux codes doublement orthogonaux récursifs [4]. C'est en effet cette classe de codes doublement orthogonaux qui est la plus prometteuse en termes de performances d'erreur. Mais ces codes récursifs n'ont encore jamais fait l'objet d'une étude à part entière même si tout le processus de décodage de ces codes a déjà été déterminé dans [4]. Ainsi, le premier objectif de mon projet de recherche a été de déterminer l'ensemble des conditions que ces codes doivent respecter. Puis, une fois ces conditions obtenues, je me suis attaché à trouver diverses méthodes de construction des codes afin de réduire la latence introduite au maximum, sans dégrader les résultats. Enfin, plusieurs simulations ont été effectuées afin de vérifier les performances d'erreur des codes obtenus, et de pouvoir les comparer aux performances des codes doublement orthogonaux connus.

Organisation du mémoire

Le mémoire a été construit de la manière suivante :

Le chapitre 1 rassemble tout ce qui doit être connu sur la théorie de l'information ainsi

que sur le codage pour pouvoir comprendre la suite du mémoire. Il y est donc rappelé ce qu'est le codage de canal, la notion de bruit ou encore le décodage à seuil. On y définit aussi les codes convolutionnels, les codes simplement orthogonaux et les codes récursifs systématiques.

Le chapitre 2 introduit la notion de décodage itératif, décrit la théorie du codage doublement orthogonal et explique la différence entre les codes convolutionnels doublement orthogonaux au sens large et au sens strict. On y compare aussi les performances d'erreur de tous les codes doublement orthogonaux connus pour un taux de codage de 1/2.

Le chapitre 3 regroupe toute l'étude effectuée sur les codes convolutionnels doublement orthogonaux récursifs au sens large. On y définit les conditions que ces codes doivent respecter et les méthodes utilisées pour construire ce genre de codes. Enfin, les performances d'erreur obtenues par simulations sont présentées.

Le chapitre 4 est la suite directe du chapitre 3 puisqu'il regroupe toute l'étude faite sur les codes convolutionnels doublement orthogonaux récursifs au sens strict. Le cheminement du chapitre est donc le même que pour le précédent, à savoir : recherche de conditions, construction de codes et détermination des performances d'erreur par simulations.

Enfin, le chapitre 5 prendra la forme d'une grande conclusion qui résumera les meilleurs résultats obtenus et donnera quelques pistes qui pourront être suivies dans les travaux futurs.

Contributions apportées

Les différentes contributions apportées par mon travail de recherche sont :

- détermination des conditions complètes et spécifiques des codes convolutionnels doublement orthogonaux récursifs au sens large et au sens strict.
- construction de nouveaux codes convolutionnels doublement orthogonaux récursifs au sens large, avec minimisation du span de ces codes jusqu'à $J = 7$.
- amélioration de l'algorithme de décodage des codes convolutionnels doublement orthogonaux récursifs au sens large.
- simulation des performances d'erreur des codes récursifs au sens large jusque $J = 11$
- construction de nouveaux codes convolutionnels doublement orthogonaux récursifs au sens strict, avec la possibilité de faire varier le span et l'emplacement des connexions du codeur.
- étude des meilleures configurations pour les codes convolutionnels doublement orthogonaux récursifs au sens strict.
- détermination par simulations des performances d'erreur des codes obtenus.

CHAPITRE 1

LES BASES DU CODAGE

Tout le monde se rend compte de l'émergence fulgurante des télécommunications au cours des dernières années. Il est même difficile de se rappeler comment nous pouvions vivre il y a encore 15 ans, sans téléphone cellulaire ni Internet, tant ceci paraît aujourd'hui indispensable, autant dans notre vie professionnelle que personnelle.

Cependant, dans le public, peu de personnes savent ce qui se cache derrière les termes de codage, de bruit ou encore d'orthogonalité. Ce premier chapitre va ainsi introduire quelques bases de télécommunication qui permettront de pouvoir suivre ce mémoire dans son ensemble.

1.1 Une communication numérique

Le principe de base d'une communication numérique est d'envoyer une séquence de bits à partir d'un émetteur et de pouvoir retrouver cette séquence au récepteur. Les deux principales contraintes associées à cette communication sont la fiabilité et la rapidité. En effet, il faudrait que la séquence de bits obtenue au récepteur soit identique à celle émise, tout en assurant une transmission aussi rapide que possible. La figure 1.1 présente un schéma de base d'un système de communication point à point utilisant le codage correcteur d'erreurs.

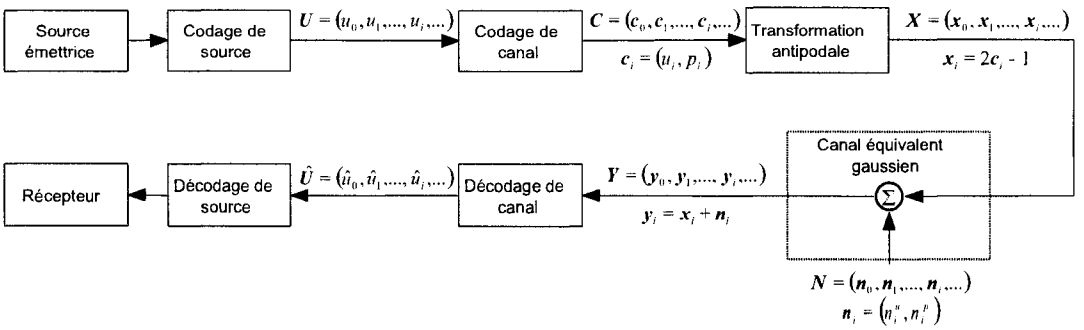

FIG. 1.1 Résumé d'une télécommunication avec un canal non quantifié

1.1.1 Différents types de codage

La première partie d'une communication numérique consiste en une opération nommée *codage de source*. Ce codage a pour but de réduire au maximum la redondance d'information introduite par la source. Il y aura alors, grâce à cette compression, moins d'information à transmettre, d'où un gain de temps. Ce type de codage ne sera cependant pas traité par la suite.

Nous avons ensuite le *codage de canal*, qui occuppe un rôle complètement différent. En effet, on va maintenant rajouter une redondance contrôlée de l'information dans le but de prévenir d'éventuelles erreurs, qui pourraient être introduites pendant la transmission. C'est ce type de codage qui va nous intéresser en particulier dans la suite de ce mémoire. Nous allons chercher des codes convolutionnels doublement orthogonaux récursifs qui vont nous permettre de minimiser la probabilité d'erreur par bit. Nous noterons $\boldsymbol{U} = (u_0, u_1, \ldots, u_i, \ldots)$ la séquence de bits à l'entrée du codeur, où $u_i \in \{0, 1\}$ est un bit d'information, émis avec l'énergie E_b, et $i = 0, 1, 2, \ldots$ est l'instant de transmission.

Tous les codes utilisés dans la suite du mémoire sont des codes dits systématiques et

de taux de codage $R = 1/2$. Ceci signifie que pour chaque bit u_i à l'entrée du codeur, nous aurons deux bits en sortie : le bit d'information lui-même (code systématique) et un deuxième symbole, p_i, dit de parité, et calculé par des additions modulo-2, représentées par le symbole $\oplus$, sur un nombre fini de bits d'information précédant le bit d'information courant u_i, où $i = 0, 1, 2, \ldots$. Ces symboles de parité serviront à calculer des symboles de syndromes, qui nous permettront à leur tour de détecter d'éventuelles erreurs de transmission. Ainsi, nous aurons à la sortie du codeur la séquence $\boldsymbol{C} = (\boldsymbol{c}_0, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_i, \ldots)$, dont chaque élément $\boldsymbol{c}_i = (u_i, p_i)$ est composé du bit d'information courant u_i et du symbole de parité associé p_i.

Un exemple élémentaire de codage de canal systématique de taux $R = 1/3$ (ajout de deux symboles de parité en plus du bit d'information), serait de coder la séquence $\boldsymbol{U}$ en répétant le bit d'information. La séquence à la sortie du codeur sera donc $\boldsymbol{C} = (\boldsymbol{c}_0, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_i, \ldots)$ avec $i = 0, 1, 2, \ldots$ et $\boldsymbol{c}_i = (u_i, p1_i, p2_i)$ tels que $p1_i = p2_i = u_i$. Ceci signifie simplement qu'au lieu d'envoyer un "0" ou un "1", on envoie "000" ou "111". Nous allons voir un peu plus loin pourquoi tripler le nombre de bits de cette façon peut permettre de corriger des erreurs de transmission.

1.1.2 Modèle du canal

La séquence $\boldsymbol{C}$ est ensuite envoyée vers le canal de transmission qui peut être décrit selon deux approches. On peut le considérer quantifié, auquel cas nous ne travaillons qu'avec des bits, ou non quantifié, ce qui nous permettra d'utiliser des valeurs réelles.

1.1.2.1 Canal quantifié

Dans le cas du canal quantifié, la séquence $\boldsymbol{C}$ est directement envoyée dans un canal binaire symétrique dont le bruit est représenté par la séquence $\boldsymbol{E} = (\boldsymbol{e}_0, \boldsymbol{e}_1, \ldots, \boldsymbol{e}_i, \ldots)$ dont chaque élément $\boldsymbol{e}_i = (e_i^u, e_i^p)$ a deux composantes e_i^u et e_i^p qui vont agir respectivement sur le bit dinformation u_i et sur le symbole de parité p_i comme le montre la figure 1.2 et l'équation (1.1).

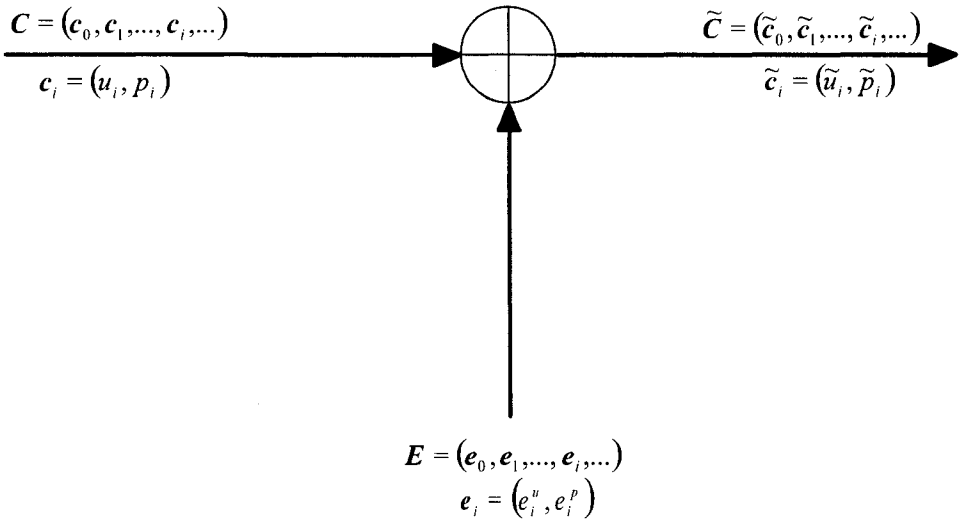

FIG. 1.2 Canal Binaire Symétrique en quantification ferme

On aura donc à la sortie du canal la séquence $\tilde{\boldsymbol{C}} = (\tilde{\boldsymbol{c}}_0, \tilde{\boldsymbol{c}}_1, \ldots, \tilde{\boldsymbol{c}}_i, \ldots)$ dont chaque élément $\tilde{\boldsymbol{c}}_i = (\tilde{u}_i, \tilde{p}_i)$ vaut :

$$\begin{aligned} \tilde{u}_i &= u_i \oplus e_i^u \\ \tilde{p}_i &= p_i \oplus e_i^p \end{aligned} \tag{1.1}$$

Il en découle que nous aurons à faire face à une erreur à chaque fois qu'une des composantes de $\boldsymbol{e}_i = (e_i^u, e_i^p)$ aura la valeur 1. Dans notre cas du canal binaire symétrique, avec une modulation BPSK utilisée pour transmettre l'information dans un canal réel dont le bruit additif est blanc, Gaussien, de moyenne nulle et de spectre de densité de puissance bilatérale égal à $N_0/2$, la probabilité qu'une des composantes de $\boldsymbol{e}_i$ soit égale à 1 vaut γ_0 dont la valeur est [18] :

$$\gamma_0 = Q(\sqrt{2RE_b/N_0}) = Q(\sqrt{E_b/N_0}) \quad \text{car } R = 1/2.$$

$$\text{avec} \quad Q(x) = \int_x^\infty \frac{1}{\sqrt{2\pi}} e^{-t^2/2} dt$$

1.1.2.2 Canal non quantifié

Avant de transmettre les symboles $\boldsymbol{c}_i$ vers le canal physique non quantifié, nous avons, pour la modulation BPSK, une transformation qui va les préparer en symboles de canal antipodaux. La séquence transformée sera notée $\boldsymbol{X} = (\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_i, \ldots)$, dont chaque élément $\boldsymbol{x}_i = (x_i^u, x_i^p) = 2 \cdot \boldsymbol{c}_i - 1$ est calculé selon :

$$\begin{aligned} x_i^u &= 2 \cdot u_i - 1 \\ x_i^p &= 2 \cdot p_i - 1 \end{aligned}$$

Les éléments x_i^u et x_i^p prennent donc des valeurs dans l'ensemble $\{-1, 1\}$.

La séquence $\boldsymbol{X}$ est ensuite envoyée vers le canal de transmission qui peut être décom-

posé comme le montre la figure 1.3.

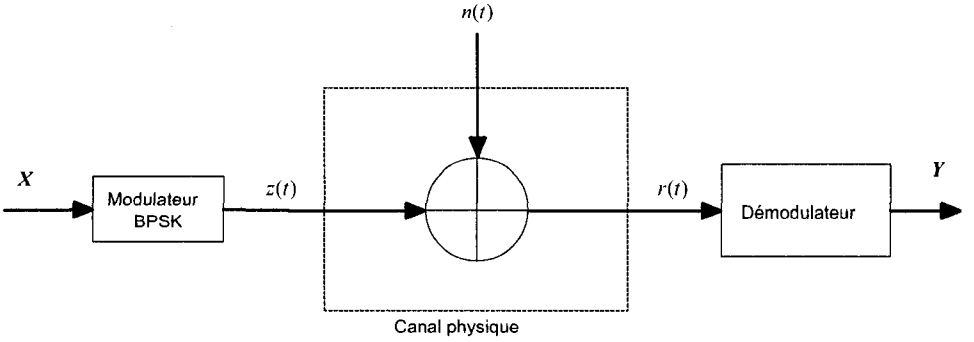

FIG. 1.3 Canal équivalent Gaussien

Nous obtenons ainsi, à la sortie du modulateur BPSK, le signal $z(t)$, qui est transmis dans le canal physique à bruit blanc, additif et Gaussien de moyenne nulle, noté $n(t)$. Ce bruit sera considéré comme une séquence $\boldsymbol{N} = (\boldsymbol{n}_0, \boldsymbol{n}_1, \ldots, \boldsymbol{n}_i, \ldots)$, avec $i = 0, 1, 2, \ldots$, où chaque symbole $\boldsymbol{n}_i$ est composé de deux valeurs, $\boldsymbol{n}_i = (n_i^u, n_i^p)$, qui agiront respectivement sur x_i^u et x_i^p comme le montre l'équation (1.2). Les éléments de bruit n_i^u et n_i^p sont des variables aléatoires, réelles, Gaussiennes, de moyenne nulle et de variance $N_0/2$. Enfin, le signal reçu en sortie du canal physique, $r(t) = z(t) + n(t)$, est envoyé dans un démodulateur, dont le filtre adapté donnera en sortie la séquence considérée comme non quantifiée $\boldsymbol{Y} = (\boldsymbol{y}_0, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_i, \ldots)$. Chaque élément de cette séquence vaut $\boldsymbol{y}_i = (y_i^u, y_i^p)$ et peut être calculé selon :

$$\begin{aligned} y_i^u &= x_i^u + n_i^u \\ y_i^p &= x_i^p + n_i^p \end{aligned} \tag{1.2}$$

1.1.3 Décodage

La dernière partie de la communication numérique consiste enfin à décoder la séquence $\boldsymbol{Y}$ qui sort du canal. Le décodeur devra ainsi essayer de retrouver la séquence initiale $\boldsymbol{U}$. Pour ceci, il effectue certains calculs qui seront développés plus loin dans ce mémoire, et donne en sortie une dernière séquence $\hat{\boldsymbol{U}} = (\hat{u}_0, \hat{u}_1, \ldots, \hat{u}_i, \ldots)$, appelée séquence décodée, qui correspond à la décision finale faite par le décodeur. Nous pourrons alors vérifier la fiabilité du codage, en évaluant la probabilité d'erreur par bit transmis, qui vaut $P_e = Prob(\hat{u}_i \neq u_i)$.

Nous pouvons maintenant revenir à notre exemple de codeur de canal. Nous avions décidé de remplacer les "0" par "000" et les "1" par "111". En reprenant les notations introduites précédemment, le récepteur va donc observer une séquence de symboles codés $\boldsymbol{y}_i$, avec $i = 0, 1, 2, \ldots$, qui auront pour valeur $\boldsymbol{y}_i = \boldsymbol{x}_i + \boldsymbol{n}_i$. Nous rappelons que $\boldsymbol{x}_i$ peut être décomposé selon $\boldsymbol{x}_i = (x_i^u, x_i^{p1}, x_i^{p2})$ et vaut donc $(-1, -1, -1)$ si $u_i = 0$ et $(1, 1, 1)$ si $u_i = 1$. Nous avons de la même manière $\boldsymbol{n}_i = (n_i^u, n_i^{p1}, n_i^{p2})$, où chaque élément de $\boldsymbol{n}_i$ est une variable aléatoire, réelle, Gaussienne, de moyenne nulle et de variance $N_0/2$.

On va alors demander au décodeur de garder la valeur majoritaire dans la séquence de symboles $\boldsymbol{y}_i$ qu'il reçoit, c'est-à-dire que si le décodeur trouve $(0, 1, 0)$ sur le groupe de trois bits qu'il étudie, il décidera que le bit d'information d'origine était 0. En revanche, si le décodeur observait $(1, 1, 0)$, il décidera la valeur 1 pour le bit d'information transmis par la source.

Nous pouvons alors calculer la probabilité d'erreur par bit dans les deux cas. Supposons

que le bit émis soit $u_i = 0$. On a donc $x_i = -1$. Sans le codage de canal, la probabilité d'erreur vaut la probabilité que la composante n_i soit supérieure à 1. Si on suppose que n_i est une variable aléatoire suivant la loi Gaussienne $N(0,1)$, on résume les interprétations possibles du décodeur dans le tableau 1.1 :

bit émis	bit décodé	probabilité
0	0 $\surd$	$1 - p_e$
0	1 $\times$	p_e

avec $p_e = Q(1) = 1.59 \cdot 10^{-1}$

TAB. 1.1 Interprétations du décodeur sans le codage de canal

La probabilité d'erreur vaut donc $P_e = p_e = 1.59 \cdot 10^{-1}$

Avec le codage de canal, en prenant la même loi pour les composantes de n_i, les décisions du décodeur changent et sont résumées dans le tableau 1.2 :

bit émis	séquence décodée	bit décidé	probabilité
0	000	0 $\surd$	$(1-p_e)^3$
0	001	0 $\surd$	$p_e(1-p_e)^2$
0	010	0 $\surd$	$p_e(1-p_e)^2$
0	011	1 $\times$	$p_e^2(1-p_e)$
0	100	0 $\surd$	$p_e(1-p_e)^2$
0	101	1 $\times$	$p_e^2(1-p_e)$
0	110	1 $\times$	$p_e^2(1-p_e)$
0	111	1 $\times$	p_e^3

avec $p_e = Q(1) = 1.59 \cdot 10^{-1}$

TAB. 1.2 Interprétations du décodeur avec le codage de canal

La probabilité d'erreur vaut donc $P_e = p_e^3 + 3 \cdot p_e^2(1-p_e) = 6.78 \cdot 10^{-2}$.

Nous pouvons ainsi voir qu'en rajoutant de la redondance de l'information, nous avons réussi à améliorer les performances du décodeur. Cependant, un tel code triple le nombre de symboles à transmettre pour une amélioration assez faible et est peu intéressant.

1.2 Codes convolutionnels simplement orthogonaux

Nous avons vu dans la partie précédente la nécessité du codage de canal pour une communication numérique. Or, nous allons être amenés dans la suite de ce mémoire à discuter de différents types de codes. Nous verrons tout d'abord dans le prochain chapitre les codes convolutionnels doublement orthogonaux au sens large ($CSO^2C - WS$, en Anglais : Convolutional Self-Doubly Orthogonal Codes in the Wide Sense) et au sens strict ($CSO^2C - SS$: Convolutional Self-Doubly Orthogonal Codes in the Strict Sense). Puis, nous étudierons ensuite de manière plus approfondie les codes convolutionnels doublement orthogonaux récursifs au sens large ($R - CSO^2C - WS$: Recursive Convolutional Self-Doubly Orthogonal Codes in the Wide Sense) et au sens strict ($R - CSO^2C - SS$: Recursive Convolutional Self-Doubly Orthogonal Codes in the Strict Sense). Cependant, pour chacun de ces types de codes, le codeur, toujours systématique et convolutionnel, diffère légèrement. Afin de se familiariser avec les notions de codes convolutionnels, d'orthogonalité et de récursivité, nous allons maintenant introduire un codeur convolutionnel utilisé pour les codes convolutionnels simplement orthogonaux ($CSOC$: Convolutional Self Orthogonal Codes), ainsi que le décodeur associé, initialement proposés par Massey dans [15].

1.2.1 Codage convolutionnel

Tous les codeurs présentés dans la suite de ce mémoire sont des codeurs convolutionnels, qui ont été introduits en 1955 pas Elias [8]. Un exemple de codeur convolutionnel

systématique de taux de codage $R = 1/2$ est donné à la figure 1.4.

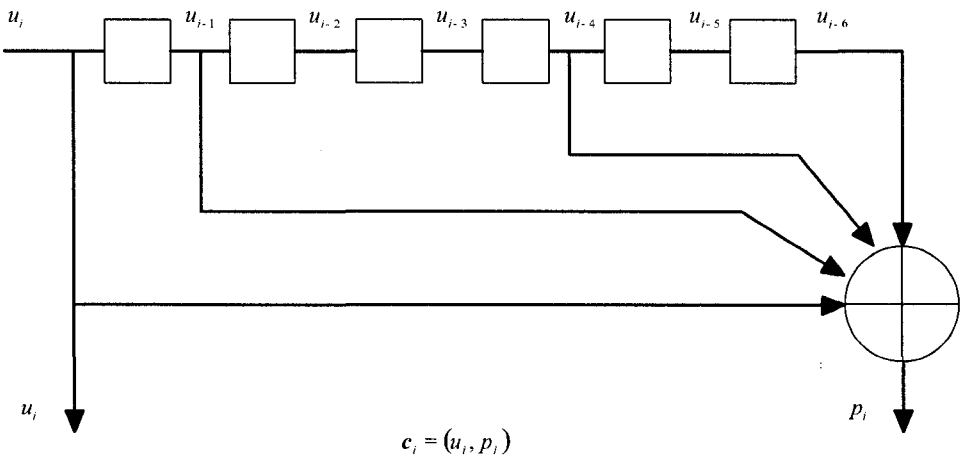

FIG. 1.4 Exemple de codeur convolutionnel avec R=1/2, J=4 et m=6

Certains termes reliés au codeur doivent être assimilés pour comprendre la suite du mémoire. Il y a tout d'abord le nombre de bits reliés à l'additionneur, noté J. Les numéros des cases du registre à décalage connectées à l'additionneur sont notés α_j avec $j \in \{1, \ldots, J\}$ et le codeur est représenté par le vecteur de connexions $\boldsymbol{\alpha} = \{\alpha_1, \alpha_2, \cdots, \alpha_J\}$. Enfin, le span du codeur, ou la mémoire du codeur, vaut $m = \alpha_J$.

Nous avons donc à la figure 1.4 un codeur convolutionnel avec $J = 4$ connexions qui sont

$$\begin{cases} \alpha_1 = 0 \\ \alpha_2 = 1 \\ \alpha_3 = 4 \\ \alpha_4 = 6 \end{cases}$$

On peut donc représenter ce codeur par le vecteur $\boldsymbol{\alpha} = \{0\ 1\ 4\ 6\}$ et son span, ou sa

mémoire, vaut $m = \alpha_J = 6$.

Enfin, le symbole de parité est calculé par :

$$p_i = \sum_{j=1}^{J} \oplus u_{i-\alpha_j} = u_i \oplus u_{i-1} \oplus u_{i-4} \oplus u_{i-6} \quad \text{avec } i = 0, 1, 2, \ldots \tag{1.3}$$

où u_i représente le bit d'information à l'instant i.

1.2.2 Représentation des codes convolutionnels

Il existe de nombreuses manières de représenter graphiquement un code convolutionnel. Nous avons les diagrammes d'état, les treillis ou encore les arbres. Seule la représentation en arbre va ête détaillée ici.

Sur chaque noeud de l'arbre est représenté un état. Il vaut, à l'instant i, le vecteur $\{u_{i-1}\ u_{i-2} \cdots u_{i-(\alpha_J)}\}$. Il existe donc 2^{α_J} états différents.

De ces noeuds partent 2 branches. Celle vers le haut indiquera que nous avons en entrée du codeur le bit $u_i = 0$ et celle vers le bas représentera le cas où $u_i = 1$.

Enfin, on indique sur chaque branche la valeur de la sortie du codeur, qui contient le bit d'information u_i et le symbole de parité p_i défini à l'équation (1.3).

L'arbre correspondant au codeur de la figure 1.4 sera donc :

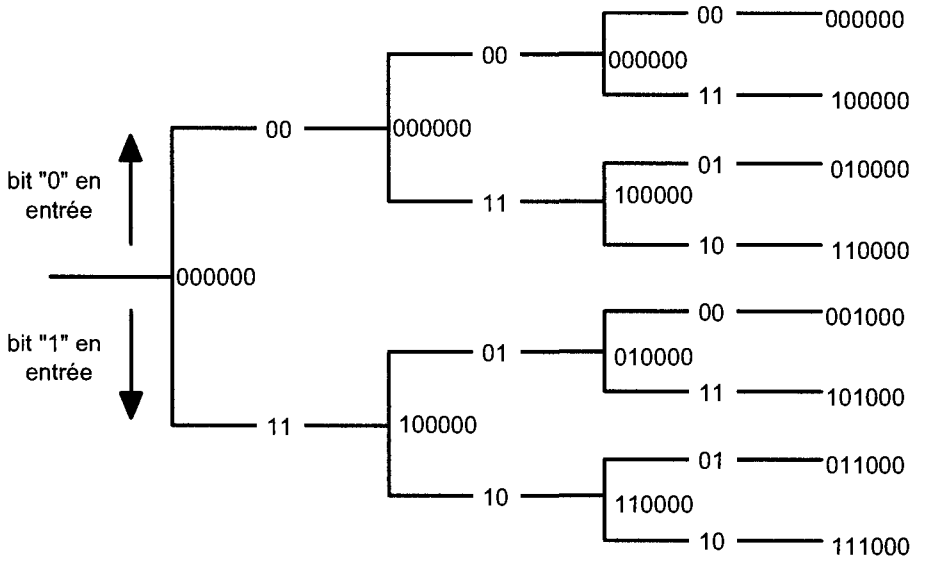

FIG. 1.5 Arbre du code convolutionnel représenté par le vecteur $\boldsymbol{\alpha} = \{0\ 1\ 4\ 6\}$

1.2.3 Condition d'orthogonalité

Nous introduisons maintenant la notion d'orthogonalité proposée par Massey [15].

Définition 1.1 (Codes convolutionnels simplement orthogonaux ($CSOC$)) *Un code convolutionnel représenté par un vecteur de J connexions $\boldsymbol{\alpha} = \{\alpha_1\ \alpha_2 \cdots \alpha_J\}$ sera simplement orthogonal si et seulement si toutes les différences positives $(\alpha_k - \alpha_j)$, avec k et j dans $(1, \ldots, J)$ et $k > j$, sont distinctes.*

Nous pouvons vérifier que le code introduit à la figure 1.4 est bien un code $CSOC$ en calculant toutes les différences positives $(\alpha_k - \alpha_j)$, tel que montré au tableau 1.3.

k	j	$\alpha_k - \alpha_j$
2	1	1
3	1	4
3	2	3
4	1	6
4	2	5
4	3	2

TAB. 1.3 Vérification de l'orthogonalité du codeur de la figure 1.4

1.2.4 Décodage

Nous avons vu dans la première partie du chapitre que nous pouvions modéliser le canal de deux manières différentes. Bien que les simulations présentées dans la suite du mémoire aient toutes été effectuées à partir d'un canal non quantifié, il est important, au point de vue théorique, de détailler le processus de décodage avec un canal quantifié, inventé par Massey [15] et basé sur un calcul de symboles de syndromes.

1.2.4.1 Décodage de type dur

La sortie du canal quantifié est la séquence $\tilde{\boldsymbol{C}} = (\tilde{\boldsymbol{c}}_0, \tilde{\boldsymbol{c}}_1, \ldots, \tilde{\boldsymbol{c}}_i, \ldots)$ dont chaque élément se décompose selon $\tilde{\boldsymbol{c}}_i = (\tilde{u}_i, \tilde{p}_i)$. Pour pouvoir effectuer une décision, le décodeur va recalculer le symbole de parité p'_i à partir des bits $\tilde{u}_i$ reçus, et le compare au symbole de parité reçu correspondant, $\tilde{p}_i$. On obtient alors le symbole de syndrome s_i :

$$s_i = \tilde{p}_i \oplus p'_i \tag{1.4}$$

Ainsi, en reprenant les équations (1.3) et (1.4), s_i devient :

$$s_i = \tilde{p}_i \oplus \sum_{j=1}^{J} \oplus \tilde{u}_{i-\alpha_j} \tag{1.5}$$

Puis, en reinjectant les équations (1.1) dans (1.5)

$$s_i = p_i \oplus e_i^p \oplus \sum_{j=1}^{J} \oplus u_{i-\alpha_j} \oplus e_{i-\alpha_j}^u \tag{1.6}$$

Or, nous avons d'après l'équation (1.3) :

$$p_i \oplus \sum_{j=1}^{J} \oplus u_{i-\alpha_j} = 0$$

On en déduit que :

$$s_i = e_i^p \oplus \sum_{j=1}^{J} \oplus e_{i-\alpha_j}^u \tag{1.7}$$

Ces symboles de syndrome, qui ne dépendent que du bruit, sont ensuite stockés dans un registre à décalage de longueur α_J. Ainsi, pour $k = (1, 2, \ldots, \alpha_J)$, nous obtenons :

$$s_{i+k} = e_{i+k}^p \oplus \sum_{j=1}^{J} \oplus e_{i+k-\alpha_j}^u \tag{1.8}$$

Si on s'intéresse uniquement aux syndromes $s_{i+\alpha_k}$, on obtient, pour $k = (1, \ldots, J)$:

$$\begin{aligned} s_{i+\alpha_k} &= e^p_{i+\alpha_k} \oplus \sum_{j=1}^{J} \oplus e^u_{i+\alpha_k-\alpha_j} \\ &= e^p_{i+\alpha_k} \oplus \sum_{j=1}^{k-1} \oplus e^u_{i+\alpha_k-\alpha_j} \oplus e^u_i \oplus \sum_{j=k+1}^{J} \oplus e^u_{i+\alpha_k-\alpha_j} \\ &= e^u_i \oplus e^p_{i+\alpha_k} \oplus \sum_{j=1}^{k-1} \oplus e^u_{i+\alpha_k-\alpha_j} \oplus \sum_{j=k+1}^{J} \oplus e^u_{i+\alpha_k-\alpha_j} \end{aligned} \tag{1.9}$$

Nous pouvons remarquer que, grâce à l'orthogonalité du code, dans les J symboles de syndromes obtenus dans l'équation (1.9), le terme e^u_i est commun aux J équations alors que tous les autres termes n'apparaissent qu'une seule fois, dans un des J syndromes. Nous avons donc obtenu un système de J équations orthogonales à e^u_i, notées $A_{k,i} \triangleq s_{i+\alpha_k}$ et appelées équations de parité :

$$A_{k,i} \triangleq s_{i+\alpha_k} = e^u_i \oplus e^p_{i+\alpha_k} \oplus \sum_{j=1}^{k-1} \oplus e^u_{i+\alpha_k-\alpha_j} \oplus \sum_{j=k+1}^{J} \oplus e^u_{i+\alpha_k-\alpha_j} \tag{1.10}$$

avec $k = 1, \ldots, J$ et $i = 0, 1, 2, \ldots$.

Nous avons alors simplement à effectuer une décision majoritaire sur les équations de parité qui est la suivante : le décodeur décidera $\hat{e}^u_i = 1$ si et seulement si plus de $\lfloor J/2 \rfloor$ équations de parité $A_{k,i}$, avec $k = 1, \ldots, J$, valent 1, où $\lfloor \cdot \rfloor$ représente la partie entière. Sinon, il décidera $\hat{e}^u_i = 0$.

Cependant, nous voyons que dans la deuxième somme de l'équation (1.10), les symboles d'erreur $e^u_{i+\alpha_k-\alpha_j}$, avec $j > k$, auront, à l'instant courant i, déjà été décodés puisque

$\alpha_k - \alpha_j < 0$. Il est alors possible d'introduire une rétroaction de la décision en rajoutant les termes déjà décodés [10]. Nous obtenons alors à partir de l'équation (1.10) :

$$A_{k,i} = e_i^u \oplus e_{i+\alpha_k}^p \oplus \sum_{j=1}^{k-1} \oplus e_{i+\alpha_k-\alpha_j}^u \oplus \sum_{j=k+1}^{J} \oplus \left(e_{i+\alpha_k-\alpha_j}^u \oplus \hat{e}_{i+\alpha_k-\alpha_j}^u \right) \tag{1.11}$$

avec $k = 1, \ldots, J$ et $i = 0, 1, 2, \ldots$

Et si on suppose la décision idéale et donc sans erreur de décodage, il vient :

$$e_{i+\alpha_k-\alpha_j}^u \oplus \hat{e}_{i+\alpha_k-\alpha_j}^u = 0 \tag{1.12}$$

$$\text{avec } k = (1, \ldots, J)\ ,\ j = (k+1, \ldots, J) \text{ et } i = 0, 1, 2, \ldots$$

Ainsi, l'équation (1.11) devient, avec $k = 1, \ldots, J$ et $i = 0, 1, 2, \ldots$:

$$A_{k,i} = e_i^u \oplus e_{i+\alpha_k}^p \oplus \sum_{j=1}^{k-1} \oplus e_{i+\alpha_k-\alpha_j}^u \tag{1.13}$$

Ainsi, si on reprend l'exemple de code $CSOC$ utilisé précédemment à la figure 1.4, les $J = 4$ équations sont :

$$\begin{aligned} A_{1,i} &= e_i^u \oplus e_i^p \\ A_{2,i} &= e_i^u \oplus e_{i+1}^p \oplus e_{i+1}^u \\ A_{3,i} &= e_i^u \oplus e_{i+4}^p \oplus e_{i+4}^u \oplus e_{i+3}^u \\ A_{4,i} &= e_i^u \oplus e_{i+6}^p \oplus e_{i+6}^u \oplus e_{i+5}^u \oplus e_{i+2}^u \end{aligned} \tag{1.14}$$

On voit bien que le terme e_i^u apparaît dans les J équations alors que tous les autres termes n'apparaissent qu'une seule fois : on a bien un système de J équations orthogonales au symbole e_i^u. Le décodeur-type, appelé décodeur à seuil, qu'on utilise en quantification dure pour ces codes est représenté à la figure 1.6.

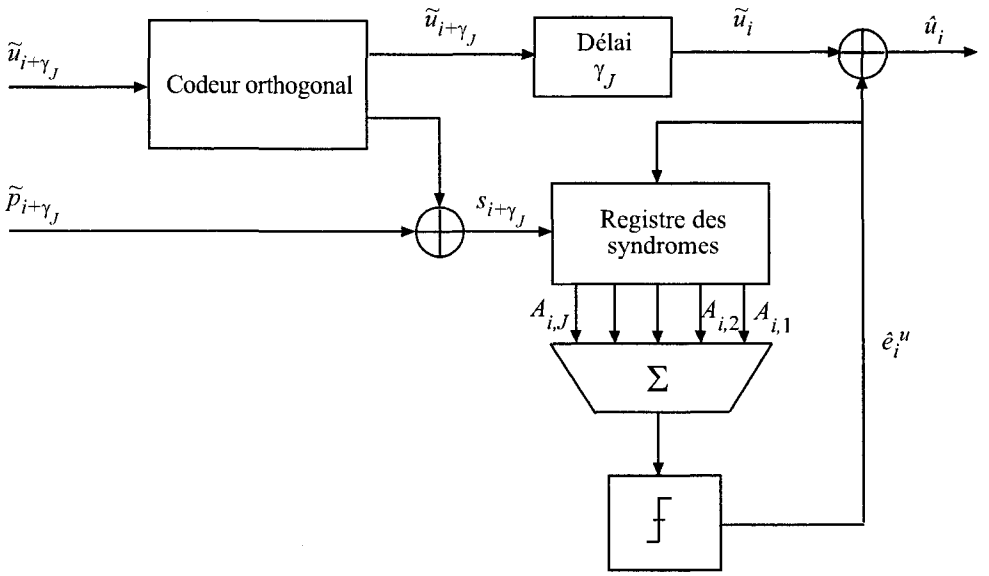

FIG. 1.6 Décodeur à seuil en quantification dure

1.2.4.2 Décodage à seuil à sorties non quantifiées

Dans ce cas, le décodeur va utiliser $J + 1$ équations obtenues à partir des équations $A_{k,i}$ avec $k = 1, \ldots, J$ en leur additionnant le symbole $\tilde{u}_i$. On obtient alors, à partir de l'équation (1.10) :

$$B_{0,i} = \tilde{u}_i \tag{1.15}$$

$$B_{k,i} = \tilde{u}_i \oplus e_i^u \oplus e_{i+\alpha_k}^p \oplus \sum_{j=1}^{k-1} \oplus e_{i+\alpha_k-\alpha_j}^u \oplus \sum_{j=k+1}^{J} \oplus e_{i+\alpha_k-\alpha_j}^u \tag{1.16}$$

Puis en réinjectant l'équation (1.1) dans (1.16) :

$$B_{0,i} = u_i \oplus e_i^u \tag{1.17}$$

$$B_{k,i} = u_i \oplus e_{i+\alpha_k}^p \oplus \sum_{j=1}^{k-1} \oplus e_{i+\alpha_k-\alpha_j}^u \oplus \sum_{j=k+1}^{J} \oplus e_{i+\alpha_k-\alpha_j}^u \tag{1.18}$$

On a ainsi obtenu un système de $J+1$ équations orthogonales au bit u_i. La décision se fait donc maintenant directement sur le bit d'information u_i selon la règle : décider $\hat{u}_i = 1$ si et seulement si plus de $\lfloor (J+1)/2 \rfloor$ équations $B_{k,i}$, avec $k = 0, \ldots, J$, valent 1. Sinon, décider $\hat{u}_i = 0$.

Il a été démontré dans [4], par un raisonnement résumé dans l'Annexe 1, que les $J+1$ équations décrites dans (1.17) et (1.18) reviennent à estimer la valeur λ_i selon :

$$\lambda_i = y_i^u + \sum_{k=1}^{J} \left(y_{i+\alpha_k}^p \diamond \sum_{j=1}^{k-1} \diamond y_{i+\alpha_k-\alpha_j}^u \diamond \sum_{j=k+1}^{J} \diamond \lambda_{i+\alpha_k-\alpha_j} \right) \tag{1.19}$$

$$= y_i^u + \sum_{k=1}^{J} \Psi_{k,i} \tag{1.20}$$

La quantité λ_i est l'approximation du logarithme du rapport de vraisemblance, LRV, du bit d'information u_i [4]. Les termes $\Psi_{k,i}$, avec $k = 1, \ldots, J$, représentent l'estimation de

chacune des équations $B_{k,i}$ dans l'équation (1.18). Nous utilisons aussi un nouvel opérateur représenté par $\diamond$, nommé *add-min*. Il a été défini dans [4], et il représente l'opération suivante :

$$addmin(input1, input2) = -sign(input1)sign(input2) \cdot \min\left(|input1|, |input2|\right)$$

Enfin, lorsque le décodeur a effectué l'estimation de la valeur de λ_i, la règle de décision est :

Décider $\hat{u}_i = 1$ *si et seulement si* $\lambda_i > 0$. *Sinon, décider* $\hat{u}_i = 0$.

Ainsi, si on applique l'équation (1.20) au codeur représenté à la figure 1.4, on obtient :

$$\begin{aligned} \lambda_i &= y_i^u + \sum_{k=1}^{4} \Psi_{k,i} \\ \text{avec} \quad \Psi_{1,i} &= y_i^p \diamond \lambda_{i-1} \diamond \lambda_{i-4} \diamond \lambda_{i-6} \\ \Psi_{2,i} &= y_{i+1}^p \diamond y_{i+1}^u \diamond \lambda_{i-3} \diamond \lambda_{i-5} \\ \Psi_{3,i} &= y_{i+4}^p \diamond y_{i+4}^u \diamond y_{i+3}^u \diamond \lambda_{i-2} \\ \Psi_{4,i} &= y_{i+6}^p \diamond y_{i+6}^u \diamond y_{i+5}^u \diamond y_{i+2}^u \end{aligned} \tag{1.21}$$

Le décodeur à seuil à sorties non quantifiées associé au code ayant $J = 4$ connexions et représenté par le vecteur $\boldsymbol{\alpha} = \{0, 1, 4, 6\}$ est montré à la figure 1.7.

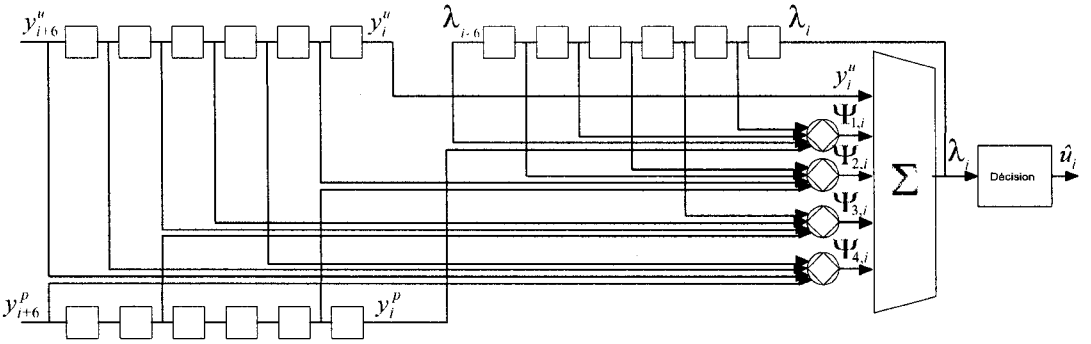

FIG. 1.7 Décodeur à seuil à sorties non quantifiées associé au code convolutionnel $\boldsymbol{\alpha} = \{0\ 1\ 4\ 6\}$

1.3 Codes Convolutionnels récursifs systématiques

1.3.1 Construction à partir de codes convolutionnels non systématiques

Nous avons présenté à la section précédente, à la figure 1.4, un codeur convolutionnel systématique de taux $R = 1/2$. Considérons maintenant un codeur convolutionnel, toujours de taux $R = 1/2$, mais non systématique. Nous aurons donc pour chaque bit u_i en entrée du codeur, deux symboles de parité en sortie, p_{i_1} et p_{i_2}, calculés par des additions modulo-2 sur des bits antérieurs au bit u_i. Un exemple est donné à la figure 1.8 :

En reprenant la notation vectorielle introduite précédemment, ce codeur est représenté par les deux vecteurs $\boldsymbol{\alpha} = \{\alpha_1\ \alpha_2\ \alpha_3\}$ de longueur $J_1 = 3$ et $\boldsymbol{\beta} = \{\beta_1\ \beta_2\}$ de longueur $J_2 = 2$ respectivement égaux à $\{0\ 1\ 2\}$ et $\{0\ 2\}$.

L'idée est alors de prendre le générateur $\boldsymbol{\beta}$ pour effectuer une boucle de retour interne, comme montré à la figure 1.9 :

Nous pouvons observer sur la figure 1.9 que l'additionneur placé avant le registre à

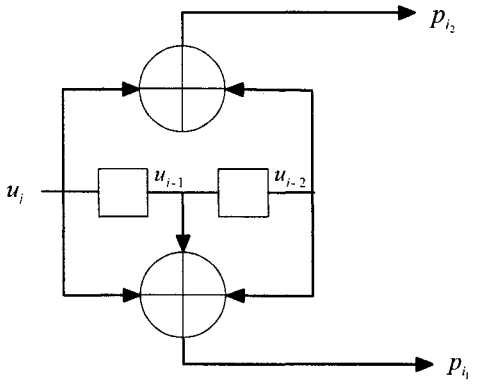

FIG. 1.8 Codeur convolutionnel non systématique de taux R=1/2 représenté par les vecteurs $\alpha = \{0\ 1\ 2\}$ et $\beta = \{0\ 2\}$

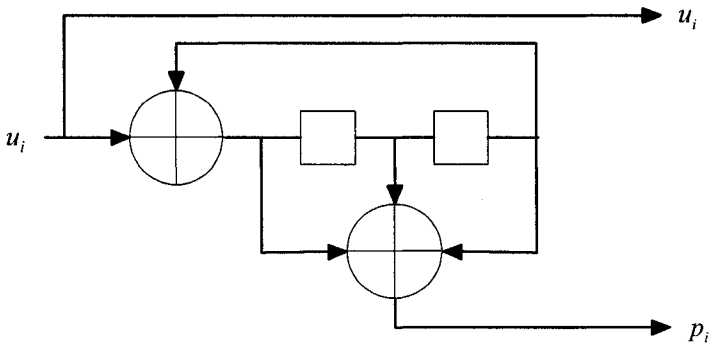

FIG. 1.9 Codeur convolutionnel récursif systématique de taux R=1/2 représenté par les vecteurs $\alpha = \{0\ 1\ 2\}$ et $\beta = \{2\}$

décalage n'est plus relié qu'à une seule case du registre. Ainsi, nous représentons ce code convolutionnel récursif systématique par les deux générateurs $\boldsymbol{\alpha} = \{0\ 1\ 2\}$ et $\boldsymbol{\beta} = \{2\}$ de longueurs respectives $J_1 = 3$ et $J_2 = 1$, $\boldsymbol{\beta}$ étant le générateur utilisé pour la boucle de retour. Cette notation sera conservée dans tout le reste du mémoire.

1.3.2 Propriétés de distance

Nous allons maintenant introduire certaines propriétés de distance qui caractérisent les performances d'un code convolutionnel [13].

La *distance de Hamming*, notée d_H, entre deux mots de codes $\boldsymbol{C_1}$ et $\boldsymbol{C_2}$, est égale au poids de Hamming W_H, et vaut donc le nombre de bits égaux à "1" de la somme $\boldsymbol{C_1} \oplus \boldsymbol{C_2}$:

$$d_H(\boldsymbol{C_1}, \boldsymbol{C_2}) = W_H(\boldsymbol{C_1} \oplus \boldsymbol{C_2})$$

La *fonction de distance des colonnes d'ordre n*, notée $d_c(n)$, est la distance de Hamming minimale entre toutes les paires de mots de code de longueur n branches qui différent dans leur première branche :

$$d_c(n) = min(d_H(\boldsymbol{C_{1_n}}, \boldsymbol{C_{2_n}}))$$

Le *profil de distance*, noté $\boldsymbol{d}$, est constitué de l'ensemble des $d_c(n)$ pour $n = 1, 2, \ldots, k$:

$$\boldsymbol{d} = (d_c(1), d_c(2), \ldots, d_c(k))$$

Enfin, la *distance libre*, notée d_{free}, vaut :

$$d_{free} = \lim_{n \to \infty} d_c(n)$$

En général, les codes les plus puissants sont ceux dont la distance libre est maximale.

1.3.3 Spectres des codes convolutionnels

Le spectre représente l'ensemble des mots de code non nuls en fonction de leur distance de Hamming par rapport à 0. Il est présenté sous forme de tableau de trois colonnes où on y inscrit :

- d : poids des mots de code
- A_d : nombre de mots de code de poids d
- C_d : nombre de bits d'information "1" ayant contribués à l'obtention des A_d mots de code de poids d

Ainsi, chaque triplet $\{d, A_d, C_d\}$ représente une raie du spectre.

Il existe un nombre infini de raies dans un spectre. Cependant, nous verrons dans la prochaine section qu'il n'est pas nécessaire d'obtenir un grand nombre de raies pour pouvoir évaluer la performance d'erreur d'un code. Ainsi, le spectre de L raies se définira comme le dénombrement de tous les chemins de poids inférieurs ou égaux à L, qui débutent par le bit d'information "1" et qui finissent à un noeud d'état 0. Il a été montré que le poids de la première raie non nulle vaut d_{free} [19].

Le spectre du code présenté à la figure 1.9, pour des raies de poids inférieurs ou égaux à 10 est présenté dans le tableau suivant :

d	A_d	C_d
5	1	2
6	2	6
7	4	14
8	8	32
9	16	72
10	32	160

TAB. 1.4 Spectre du code récursif systématique représenté par $\boldsymbol{\alpha} = \{0\ 1\ 2\}$ et $\boldsymbol{\beta} = \{2\}$

1.3.4 Évaluation des performances d'un code à partir de son spectre

Une évaluation de la probabilité d'erreur par bit d'information en utilisant un décodage MAP a été faite dans [19] :

$$P_b \leq \sum_{d=d_{free}}^{\infty} C_d \cdot P_d \tag{1.22}$$

où P_d est la probabilité d'erreur entre deux mots de codes, et vaut, pour un canal à bruit blanc Gaussien additif dont la densité de puissance bilatérale est $N_0/2$ avec une modulation BPSK :

$$P_d = Q(\sqrt{2dR\frac{E_b}{N_0}})$$

$$\text{avec} \quad Q(x) = \int_x^{\infty} \frac{1}{\sqrt{2\pi}} e^{-t^2/2} dt$$

Ainsi, en reprenant le spectre du tableau 1.4, l'estimation de la probabilité d'erreur par bit correspondante est donnée à la figure 1.10, où chaque courbe prend une raie de plus en considération.

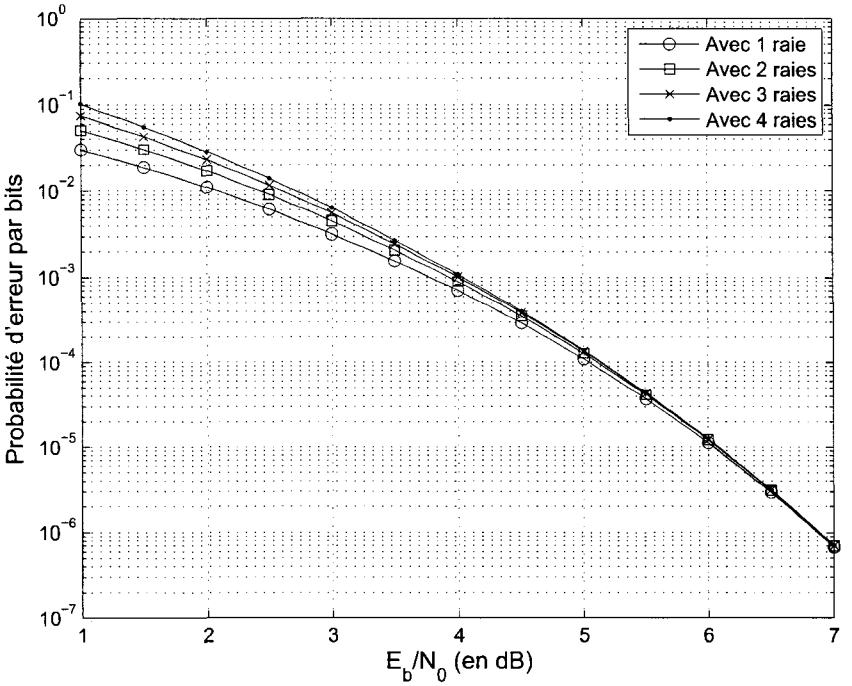

FIG. 1.10 Estimation de la probabilité d'erreur par bit du codeur convolutionnel récursif systématique avec R=1/2, $\alpha = \{0\ 1\ 2\}$ et $\beta = \{2\}$

On remarque que seules les deux premières raies du spectre ont une influence significative, surtout pour de hauts rapports signal sur bruit, sur l'estimation de la probabilité d'erreur.

1.4 Conclusion

Nous avons donc vu dans ce premier chapitre les notions de codage, de bruit, d'orthogonalité, de récursivité et de décodage à seuil. Cependant, la raison pour laquelle les codes Turbo ont d'aussi bons résultats est que le décodage se fait en plusieurs itérations (répétition du processus de décodage) et que, à chaque itération, les observables sont in-

dépendantes grâce aux entrelaceurs mentionnés auparavant. Nous allons définir dans notre prochain chapitre le principe du décodage itératif qui permettra d'aboutir sur la théorie des codes convolutionnels doublement orthogonaux CSO^2C.

CHAPITRE 2

THÉORIE DES CODES CONVOLUTIONNELS DOUBLEMENT ORTHOGONAUX NON RÉCURSIFS

2.1 Décodage à seuil itératif

Nous allons utiliser dans la suite du mémoire un décodeur à seuil itératif composé d'une succession de M décodeurs à sorties non quantifiées semblables au décodeur représenté à la figure 1.7 [5]. Nous représentons ce décodeur à seuil itératif à la figure 2.1, où chaque itération est effectuée par un décodeur distinct.

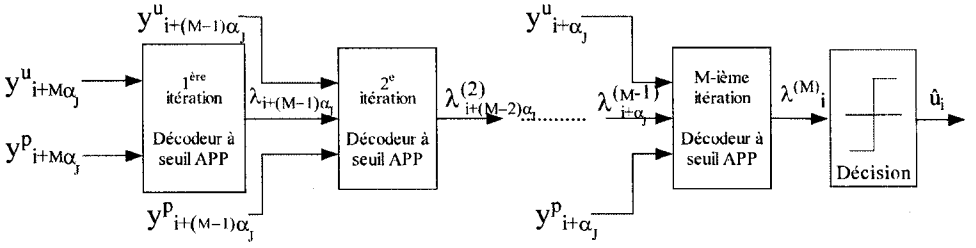

FIG. 2.1 Décodeur à seuil avec M itérations

Á chaque itération μ, $\mu \in (1, \ldots, M)$, le décodeur effectue l'estimation du LRV $\lambda_i^{(\mu)}$ de u_i, en se servant de l'estimation $\lambda_i^{(\mu-1)}$ faite à l'itération précédente ainsi que des symboles provenant du canal, y_i^u et y_i^p, selon l'équation [4] :

$$\lambda_i^{(\mu)} = y_i^u + \sum_{k=1}^{J} \left(y_{i+\alpha_k}^p \diamond \sum_{j=1}^{k-1} \diamond \lambda_{i+\alpha_k-\alpha_j}^{(\mu-1)} \diamond \sum_{j=k+1}^{J} \diamond \lambda_{i+\alpha_k-\alpha_j}^{(\mu)} \right) \tag{2.1}$$

$$= y_i^u + \sum_{k=1}^{J} \Psi_{k,i}^{(\mu)} \tag{2.2}$$

Á la dernière itération, soit pour $\mu = M$, le décodeur effectue une décision finale sur u_i en comparant la valeur $\lambda_i^{(M)}$ à un seuil. La règle de décision est : *Décider* $\hat{u}_i = 1$ *si et seulement si* $\lambda_i^{(M)} > 0$. *Sinon, décider* $\hat{u}_i = 0$.

On voit ainsi que les codes utilisés avec ce type de décodeur doivent, pour que les observables restent indépendantes au fil des itérations, non plus être simplement orthogonaux mais maintenant avoir la propriété de M-orthogonalité. Cependant, il a été montré dans [4] qu'une orthogonalité d'ordre 2 suffit pour atteindre les performances asymptotiques de codes M-orthogonaux. Nous allons donc enfin définir dans la prochaine section les codes doublement orthogonaux CSO^2C, d'abord au sens large, puis au sens strict.

2.2 Codes convolutionnels doublement orthogonaux au sens large

2.2.1 Conditions à respecter

Afin d'obtenir les conditions à respecter pour les codes $CSO^2C - WS$, il suffit de développer l'équation (2.2) pour $\mu = 2$. Nous obtenons alors, en réintégrant l'équation (1.20) :

$$\begin{aligned}
\lambda_i^{(2)} &= y_i^u + \sum_{k=1}^{J} \left(y_{i+\alpha_k}^p \diamond \sum_{j=1}^{k-1} \diamond \lambda_{i+\alpha_k-\alpha_j}^{(1)} \diamond \sum_{j=k+1}^{J} \diamond \lambda_{i+\alpha_k-\alpha_j}^{(2)} \right) \\
&= y_i^u + \sum_{k=1}^{J} \left(y_{i+\alpha_k}^p \diamond \sum_{j=1}^{k-1} \diamond \left(y_{i+\alpha_k-\alpha_j}^u + \sum_{l=1}^{J} \left(y_{i+\alpha_k-\alpha_j+\alpha_l}^p \diamond \sum_{m=1}^{l-1} \diamond y_{i+(\alpha_k-\alpha_j)-(\alpha_m-\alpha_l)}^u \right.\right.\right. \\
&\quad \left.\left.\left. \diamond \sum_{m=l+1}^{J} \diamond \lambda_{i+(\alpha_k-\alpha_j)-(\alpha_m-\alpha_l)}^{(1)} \right) \right) \diamond \sum_{j=k+1}^{J} \diamond \lambda_{i+\alpha_k-\alpha_j}^{(2)} \right) \qquad (2.3)
\end{aligned}$$

Afin de garder l'indépendance des observables sur les deux premières itérations, il faut que tous les termes présents dans l'équation (2.3) soient différents. Nous pouvons alors définir les codes $CSO^2C - WS$ [4].

Définition 2.1 (Codes convolutionnels doublement orthogonaux au sens large) *(CSO^2C-WS) Un code convolutionnel systématique de taux de codage $R = 1/2$ et représenté par un vecteur de J connexions $\boldsymbol{\alpha} = \{\alpha_1\ \alpha_2 \cdots \alpha_J\}$ sera doublement orthogonal au sens large si et seulement si il vérifie les trois conditions suivantes, pour toute combinaison des entiers $k, j, l, m \in (1, \cdots, J)$:*

1. *Les différences simples $(\alpha_k - \alpha_j)$ sont distinctes pour tout $k \neq j$.*
2. *Les différences doubles $(\alpha_k - \alpha_j) - (\alpha_m - \alpha_l)$ sont distinctes pour tout $k \neq j$, $l \neq m$, $k \neq m$, $j \neq l$, sauf pour les différences égales inévitables.*
3. *Les différences simples sont distinctes des différences doubles.*

On remarque, dans la définition 2.1, la présence de différences doubles inévitablement égales. En effet, certaines permutations des indices (k, l) et (j, m) engendrent des différences doubles égales. Par exemple, on aura $(\alpha_1 - \alpha_2) - (\alpha_3 - \alpha_4) = (\alpha_4 - \alpha_2) - (\alpha_3 - \alpha_1)$. C'est la raison pour laquelle ces codes sont dits au *sens large*, puisque l'indépendance des observables à la deuxième itération n'est pas entièrement complète.

Pour remédier à ce problème, il a été défini des codes convolutionnels doublement orthogonaux au *sens strict*, présentés dans la prochaine section, qui garantiront une indépendance totale des observables pour les deux premières itérations.

2.2.2 Coefficients de pondération

On a vu à la partie précédente que les observables ne pouvaient être indépendantes complètement à la deuxième itération. Cette difficulté a été en partie contournée en pondérant par des coefficients $a_k^{(\mu)}$ les équations de contrôle de parité modifiées $\Psi_{k,i}^{(\mu)}$ présentées à l'équation (2.2) [7]. L'idée était en premier lieu de pouvoir réduire l'influence des équations de parité qui contiennent le plus de répétitions de différences. Il en découle que l'estimation effectuée par le décodeur à seuil itératif présenté à la section 2.1 devient :

$$\lambda_i^{(\mu)} = a_0^{(\mu)} y_i^u + \sum_{k=1}^{J} a_k^{(\mu)} \Psi_{k,i}^{(\mu)} \tag{2.4}$$

Cependant, il a été observé dans [4] que les coefficients de pondération peuvent être pris égaux pour un code donné, sans grande dégradation des performances d'erreur. L'algorithme de décodage utilisé effectuera donc l'estimation suivante :

$$\lambda_i^{(\mu)} = a\left(y_i^u + \sum_{k=1}^{J} \Psi_{k,i}^{(\mu)} \right) \tag{2.5}$$

Nous pouvons voir l'influence des coefficients de pondération sur les performances des codes à la figure 2.2. La courbe, tirée de [4], représente les performances du code au sens large avec J=8 et représenté par le vecteur $\alpha = \{0\ 43\ 139\ 322\ 422\ 430\ 441\ 459\}$ à la 8$^{\text{ième}}$ itération, pour $E_b/N_0 = 4dB$.

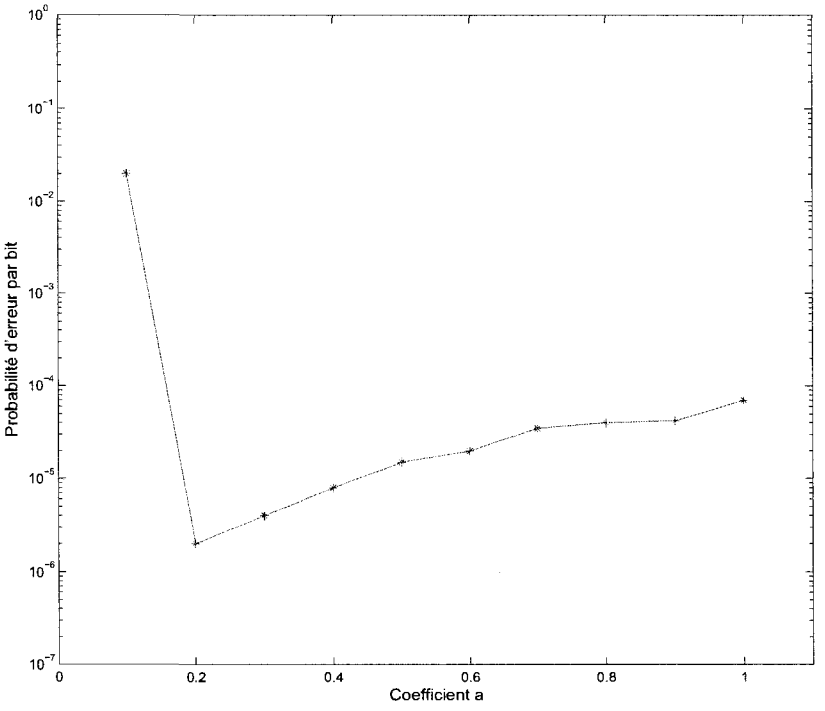

FIG. 2.2 Effet du coefficient de pondération sur la probabilité d'erreur par bit pour le code $CSO^2C - WS$ avec J=8 et représenté par le vecteur $\alpha = \{0\ 43\ 139\ 322\ 422\ 430\ 441\ 459\}$, à la $8^{\text{ième}}$ itération, pour $E_b/N_0 = 4dB$

2.2.3 Codes convolutionnels doublement orthogonaux simplifiés

Il a été introduit dans [16] des codes convolutionnels doublement orthogonaux simplifiés au sens large ($S - CSO^2C - WS$). L'idée est d'accepter un certain nombre de différences doubles égales, en plus des différences égales inévitables à cause des permutations d'indices comme nous l'avons vu à la définition 2.1. Ainsi, si on note N_d le nombre total de différences doubles, en retirant les différences inévitables, on acceptera un nombre N_d^e de différences doubles égales, $N_d^e < N_d$. La définition des codes simplifiés est donnée à la définition 2.2 [16].

Définition 2.2 (Codes convolutionnels doublement orthogonaux simplifiés au sens large) *($S - CSO^2C - WS$) Un code convolutionnel systématique de taux de codage $R = 1/2$ et représenté par un vecteur de J connexions $\boldsymbol{\alpha} = \{\alpha_1\ \alpha_2 \cdots \alpha_J\}$ sera doublement orthogonal simplifié au sens large si et seulement si il vérifie les trois conditions suivantes, pour toute combinaison des entiers $k, j, l, m \in (1, \cdots, J)$:*

1. *Les différences simples $(\alpha_k - \alpha_j)$ sont distinctes pour tout $k \neq j$.*
2. *Les différences doubles $(\alpha_k - \alpha_j) - (\alpha_m - \alpha_l)$ sont distinctes pour tout $k \neq j$, $l \neq m$, $k \neq m$, $j \neq l$, sauf pour les différences égales inévitables, ainsi que pour un nombre N_d^e d'entre elles.*
3. *Les différences simples sont distinctes des différences doubles.*

Le fait de relaxer la deuxième condition de la définition permet de réduire considérablement le span des codes, pour un J donné, par rapport aux codes non simplifiés. De plus,

comme nous le verrons plus loin dans le chapitre, les performances d'erreur n'en sont que très peu dégradées.

2.3 Codes convolutionnels doublement orthogonaux au sens strict

2.3.1 Codeur convolutionnel parallèle

La principale évolution par rapport aux codes définis au sens large est que les bits d'information sont maintenant pris par blocs de J à l'entrée du codeur. On aura alors à la sortie du codeur un ensemble de $2J$ symboles codés, les J premiers étant les bits d'entrée, puisque le code est systématique, et les J suivants, les symboles de parité, $p_{n,i}$, toujours calculés par des additions modulo-2 sur les bits d'information précédant les bits d'information courants $u_{n,i}$ avec $n \in \{1, \ldots, J\}$ et $i = 0, 1, 2, \ldots$. Le taux de codage reste donc égal à $R = J/2J = 1/2$.

Ces codeurs sont représentés par une matrice de taille $J \times J$. L'élément de la matrice situé à l'emplacement $\alpha_{i,j}$ représente le numéro de la cellule de retard du bit d'information u_i connecté à l'additionneur modulo-2 calculant le symbole de parité p_j. Nous donnons un exemple de ce type de codeurs, avec J=3, à la figure 2.3.

La matrice décrivant ce codeur est donc une matrice 3×3 qui vaut :

$$\begin{pmatrix} 3 & 5 & 0 \\ 2 & 0 & 5 \\ 0 & 5 & 2 \end{pmatrix}$$

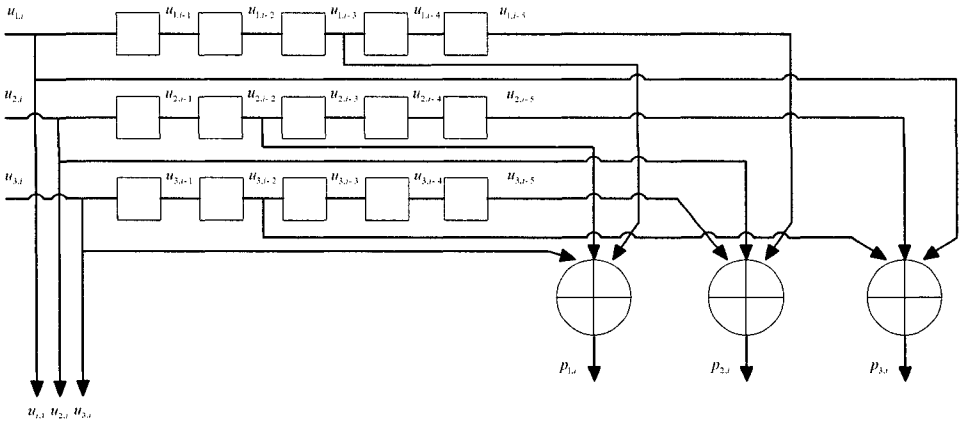

FIG. 2.3 Exemple de codeur convolutionnel en bloc, avec R=1/2, J=3 et m=5

Le span de ce codeur est alors :

$$m = \max_{\substack{i\in\{1,\ldots,J\}\\ j\in\{1,\ldots,J\}}} \{\alpha_{i,j}\} = 5$$

Enfin, avec ces codeurs, la formule pour calculer les symboles de parité est :

$$p_{n,i} = \sum_{k=1}^{J} \oplus u_{k,i-\alpha_{k,n}} \text{ avec } n \in \{1,\ldots,J\}, i = 0,1,2,\ldots \tag{2.6}$$

Si on reprend l'exemple proposé à la figure 2.3, les symboles de parité s'écrivent :

$$\begin{aligned} p_{1,i} &= \sum_{k=1}^{3} \oplus u_{k,i-\alpha_{k,1}} = u_{1,i-3} \oplus u_{2,i-2} \oplus u_{3,i} \\ p_{2,i} &= \sum_{k=1}^{3} \oplus u_{k,i-\alpha_{k,2}} = u_{1,i-5} \oplus u_{2,i} \oplus u_{3,i-5} \\ p_{3,i} &= \sum_{k=1}^{3} \oplus u_{k,i-\alpha_{k,3}} = u_{1,i} \oplus u_{2,i-5} \oplus u_{3,i-2} \end{aligned} \tag{2.7}$$

2.3.2 Conditions à respecter

Par le même raisonnement qui nous a mené à l'équation (2.3), le développement de la deuxième itération pour un code $CSO^2C - SS$ nous donne :

$$\begin{aligned}\lambda_{j,i}^{(2)} = y_{j,i}^{u} + \sum_{n=1}^{J} \Bigg(y_{n,i+\alpha_{j,n}}^{p} \diamond \sum_{k=1,k\neq j}^{J} \diamond \Bigg(y_{k,i+(\alpha_{j,n}-\alpha_{k,n})}^{u} + \sum_{s=1}^{J} \Big(y_{s,i+\alpha_{j,n}-(\alpha_{k,n}-\alpha_{k,s})}^{p} \diamond \\ \sum_{l=1,l\neq k}^{J} \diamond y_{l,i+(\alpha_{j,n}-\alpha_{k,n})-(\alpha_{l,s}-\alpha_{k,s})}^{u} \diamond \sum_{l=1,l\neq k}^{J} \diamond \lambda_{l,i+(\alpha_{j,n}-\alpha_{k,n})-(\alpha_{l,s}-\alpha_{k,s})}^{(1)} \Big) \Bigg) \diamond \\ \sum_{k=1,k\neq j}^{J} \diamond \lambda_{k,i+(\alpha_{j,n}-\alpha_{k,n})}^{(2)} \Bigg) \end{aligned} \tag{2.8}$$

L'équation (2.8) nous permet donc de définir les codes convolutionels doublement orthogonaux au sens strict [4] :

Définition 2.3 (Codes convolutionnels doublement orthogonaux au sens strict) *(CSO^2C-SS) Un code convolutionnel systématique de taux de codage $R = 1/2$ et représenté par une matrice de connexions de taille $J \times J$ dont les éléments sont notés $(\alpha_{j,n})$ sera doublement orthogonal au sens strict si et seulement si il vérifie les trois conditions suivantes, pour toute combinaison des entiers $k, j, l, n, s \in (1, \cdots, J)$:*

1. *Les différences simples $(\alpha_{j,n} - \alpha_{k,n})$ sont distinctes pour tout $k \neq j$*
2. *Les différences doubles $(\alpha_{j,n} - \alpha_{k,n}) - (\alpha_{l,s} - \alpha_{k,s})$ sont distinctes pour tout $k \neq j$, $k \neq l$, $s \neq n$.*
3. *Les différences simples sont distinctes des différences doubles.*

L'utilisation de J registres à décalage, combinée au fait que chaque registre n'est relié

qu'à un unique additionneur modulo-2, a ainsi permis d'obtenir une indépendance totale des observables à la deuxième itération. Il s'en suit une nette amélioration des performances par rapport aux codes $CSO^2C - WS$.

2.4 Conclusion

Nous avons défini les codes convolutionnels doublement orthogonaux au sens large, simplifiés ou non, et au sens strict. Un bon élément de comparaison des trois types de codes pour une valeur de E_b/N_0 donnée, est de regarder les performances en fonction de la latence totale introduite par les codes après convergence. En effet, pour les codes $CSO^2C - WS$, simplifiés ou non, la latence totale engendrée par un code représenté par le vecteur $\boldsymbol{\alpha} = \{\alpha_1\ \alpha_2 \cdots \alpha_J\}$ après M itérations vaut :

$$L = \alpha_J \cdot M$$

En revanche, pour un code $CSO^2C - SS$ représenté par une matrice de taille $J \times J$ qui converge après M itérations, la latence totale sera :

$$L = \alpha_J \cdot M \cdot J$$

Ainsi, les deux prochaines figures, issues des résultats de [4] et de [16], comparent les performances des codes au sens large et au sens strict pour des valeurs de E_b/N_0 de 2.5dB et 3.5dB. Les codes simplifiés, n'étant pas adaptés pour des E_b/N_0 inférieurs à 3dB,

n'apparaissent que sur la courbe représentant les performances d'erreur à 3.5 dB.

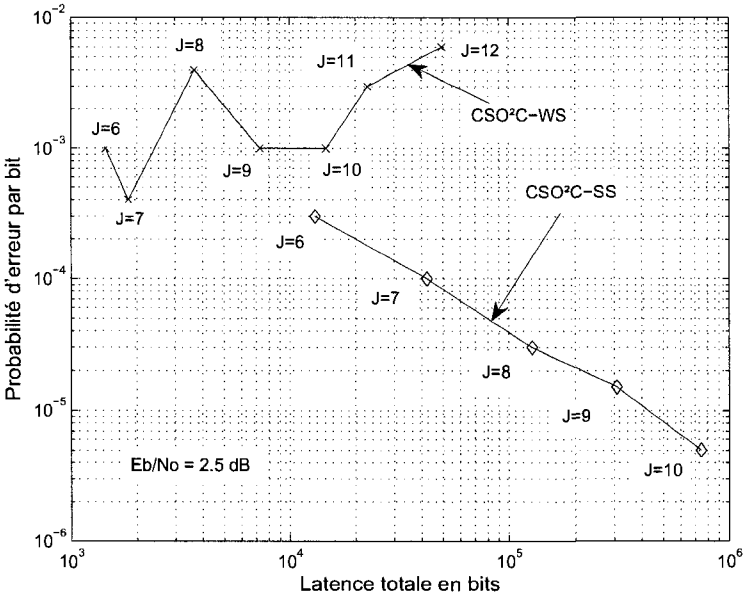

FIG. 2.4 Comparaison entre les performances d'erreur des codes $CSO^2C - WS$ et $CSO^2C - SS$ en fonction de leur latence totale après convergence, $E_b/N_0 = 2.5$ dB [4]

On peut y observer qu'à $E_b/N_0 = 2.5dB$, les codes au sens strict sont nettement plus performants que les codes au sens large. En revanche, pour $E_b/N_0 = 3.5dB$, bien que pour un J donné, les performances au sens strict soit meilleures, nous préférerons utiliser les codes au sens large, en particulier les codes simplifiés au sens large, qui, pour une probabilité d'erreur par bit donnée, engendrent moins de retard et sont bien moins complexes à être implémentés.

Les bons résultats obtenus nous poussent à chercher des variantes de ces codes pour pouvoir travailler à des rapports signal sur bruit les plus faibles possibles. Les codes convolutionnels doublement orthogonaux récursifs ont alors été définis par [4] dans cette optique.

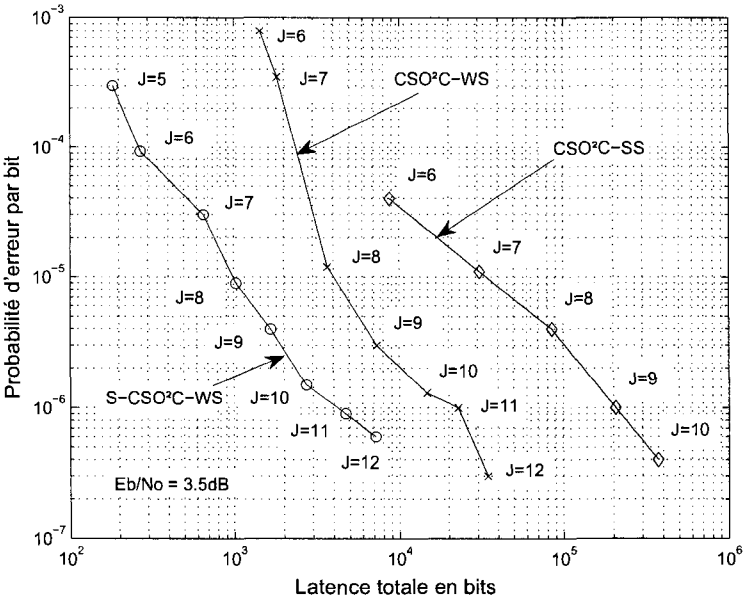

FIG. 2.5 Comparaison entre les performances d'erreur des codes $CSO^2C - WS$ et $CSO^2C - SS$ en fonction de leur latence totale après convergence, $E_b/N_0 = 3.5$ dB [4] [16]

CHAPITRE 3

THÉORIE DES CODES CONVOLUTIONNELS DOUBLEMENT ORTHOGONAUX RÉCURSIFS AU SENS LARGE

3.1 Éléments de théorie

3.1.1 Codeur convolutionnel récursif

Un codeur sera convolutionnel récursif si les opérations modulo-2 pour calculer le symbole de parité courant p_i agissent non seulement sur les bits d'information courant et précédents, mais aussi sur les symboles de parité précédents. Ces codeurs seront par conséquent représentés par deux vecteurs de connexions et ils serviront pour les codes $R - CSO^2C - WS$ [4]. Le vecteur $\boldsymbol{\alpha} = \{\alpha_1\ \alpha_2 \cdots \alpha_{J_1}\}$ indiquera les J_1 emplacements dans le registre à décalage des bits d'information reliés à l'additionneur modulo-2. Le vecteur $\boldsymbol{\beta} = \{\beta_1\ \beta_2 \cdots \beta_{J_2}\}$ donnera les J_2 emplacements des symboles de parité reliés à l'additionneur modulo-2. $\boldsymbol{\beta}$ est le vecteur générateur utilisé pour la boucle de retour interne. Un exemple de codeur, pour J=4, $J = J_1 + J_2$ étant le nombre total de connexions, et un taux de codage R=1/2, est donné aux figures 3.1 et 3.2. Les deux figures représentent le même codeur, la seconde figure étant la représentation canonique.

Dans l'exemple des figures 3.1 et 3.2, nous avons donc un codeur qui sera représenté par les vecteurs $\boldsymbol{\alpha} = \{0\ 1\ 6\}$ et $\boldsymbol{\beta} = \{4\}$. Les nombres de bits reliés à l'additionneur sont respectivement $J_1 = 3$ pour les bits d'information et $J_2 = 1$ pour les symboles de parité.

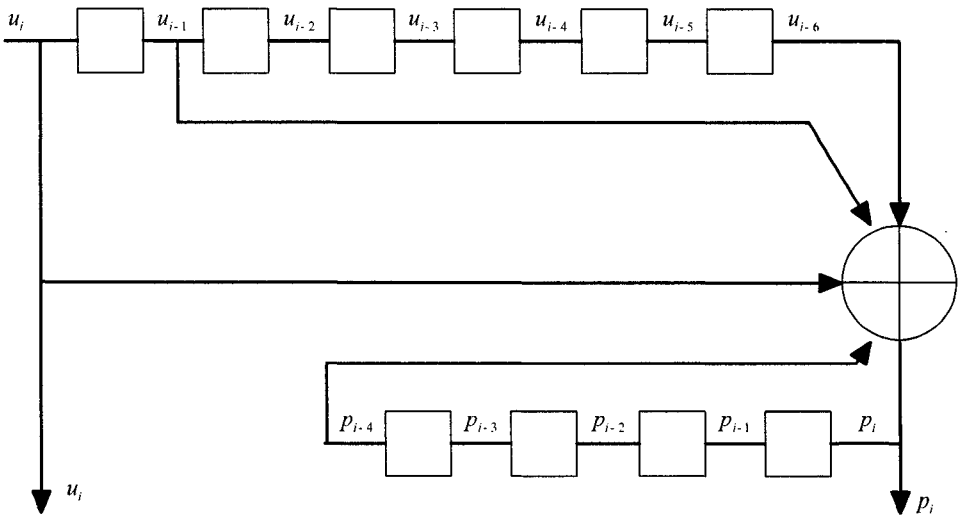

FIG. 3.1 Exemple de codeur convolutionnel récursif de taux R=1/2, représenté par les vecteurs de connexions $\alpha = \{0\ 1\ 6\}$ et $\beta = \{4\}$, avec $J_1 = 3$ et $J_2 = 1$, et $m = 6$

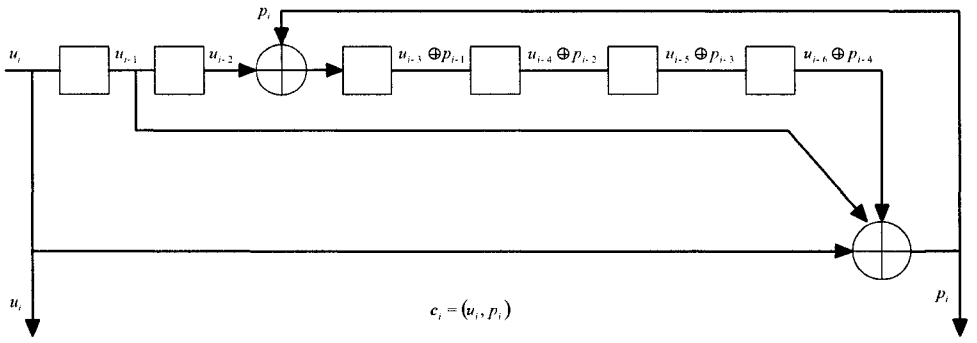

FIG. 3.2 Forme canonique du codeur convolutionnel récursif de la figure 3.1

Enfin, la mémoire du codeur est :

$$m = \max\left\{\max_{i\in\{1,\cdots,J_1\}}\{\alpha_i\}, \max_{j\in\{1,\cdots,J_2\}}\{\beta_j\}\right\} = 6$$

Pour ce type de codeurs, les symboles de parité sont calculés selon la formule :

$$p_i = \sum_{j=1}^{J_1} \oplus u_{i-\alpha_j} \oplus \sum_{k=1}^{J_2} \ominus p_{i-\beta_k} \tag{3.1}$$

On peut noter que le symbole de parité ne peut jamais être additionné à lui même, ce qui se traduit par le fait que tous les β_k, pour $k \in \{1, \ldots, J_2\}$, sont strictement positifs.

Enfin, il existe une troisième manière de représenter les codes convolutionnels récursifs systématiques, qui est celle présentée au premier chapitre à la figure 1.9. Ce codeur était représenté par les deux vecteurs $\boldsymbol{\alpha} = \{0\ 1\ 2\}$ et $\boldsymbol{\beta} = \{2\}$ de longueurs respectives $J_1 = 3$ et $J_2 = 1$. Il est démontré dans l'Annexe 2 qu'il revient au même de représenter ce codeur selon la figure 3.3.

3.1.2 Décodage

De la même manière que pour les codes non récursifs, le décodeur recalcule, à l'instant $i = 0, 1, 2, \ldots$, le symbole de parité p'_i avec les symboles d'information et de parité reçus précédemment, et il le compare avec le symbole $\tilde{p}_i$ reçu [4]. On obtient donc le symbole de syndrome s_i qui vaut :

$$s_i = \tilde{p}_i \oplus p'_i \tag{3.2}$$

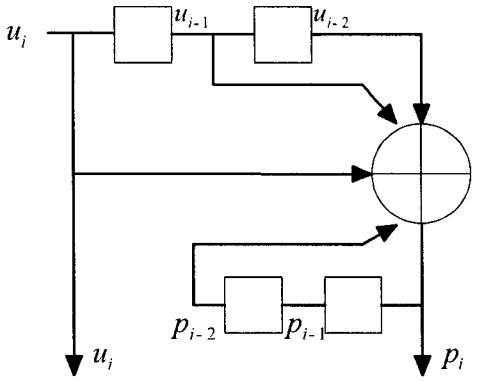

FIG. 3.3 Exemple de codeur convolutionnel récursif de taux R=1/2, représenté par les vecteurs de connexions $\boldsymbol{\alpha} = \{0\ 1\ 2\}$ et $\boldsymbol{\beta} = \{2\}$, avec $J_1 = 3$ et $J_2 = 1$, et $m = 2$

Ainsi, en reprenant les équations (3.1) et (3.2), s_i devient :

$$s_i = \tilde{p}_i \oplus \sum_{j=1}^{J_1} \oplus \tilde{u}_{i-\alpha_j} \oplus \sum_{k=1}^{J_2} \oplus \tilde{p}_{i-\beta_k} \tag{3.3}$$

On considère ensuite ces symboles aux instants $(i+\alpha_l)$ et $(i+\beta_m)$ avec $l = 1, 2, \ldots, J_1$ et $m = 1, 2, \ldots, J_2$. On obtient alors J_1 symboles de syndromes orthogonaux à $\tilde{u}_i$ à l'équation 3.4 et $J_2 + 1$ orthogonaux à $\tilde{p}_i$ à l'équation 3.5 :

$$s_{i+\alpha_l} = \tilde{u}_i \oplus \sum_{\substack{j=1 \\ j \neq l}}^{J_1} \oplus \tilde{u}_{i+\alpha_l-\alpha_j} \oplus \sum_{k=1}^{J_2} \oplus \tilde{p}_{i+\alpha_l-\beta_k} \oplus \tilde{p}_{i+\alpha_l} \tag{3.4}$$

$$\begin{aligned} s_i &= \tilde{p}_i \oplus \sum_{j=1}^{J_1} \oplus \tilde{u}_{i-\alpha_j} \oplus \sum_{k=1}^{J_2} \oplus \tilde{p}_{i-\beta_k} \\ s_{i+\beta_m} &= \tilde{p}_i \oplus \sum_{j=1}^{J_1} \oplus \tilde{u}_{i+\beta_m-\alpha_j} \oplus \sum_{\substack{k=1 \\ k \neq m}}^{J_2} \oplus \tilde{p}_{i+\beta_m-\beta_k} \oplus \tilde{p}_{i+\beta_m} \end{aligned} \tag{3.5}$$

Pour simplifier les expressions des symboles de syndrome, on introduit une connexion virtuelle $\beta_0 = 0$. Les équations (3.4) et (3.5) deviennent donc, pour $l = 1, \ldots, J_1$ et $m = 0, 1, \ldots, J_2$ [4] :

$$s_{i+\alpha_l} = \tilde{u}_i \oplus \sum_{\substack{j=1 \\ j \neq l}}^{J_1} \oplus \tilde{u}_{i+\alpha_l-\alpha_j} \oplus \sum_{k=0}^{J_2} \oplus \tilde{p}_{i+\alpha_l-\beta_k} \tag{3.6}$$

$$s_{i+\beta_m} = \tilde{p}_i \oplus \sum_{j=1}^{J_1} \oplus \tilde{u}_{i+\beta_m-\alpha_j} \oplus \sum_{\substack{k=0 \\ k \neq m}}^{J_2} \oplus \tilde{p}_{i+\beta_m-\beta_k} \tag{3.7}$$

Les équations (3.6) et (3.7) nous donnent une première série de conditions que les codes devront vérifier, et qui définissent les codes convolutionels simplement orthogonaux récursifs.

Définition 3.1 (Codes convolutionnels simplement orthogonaux récursifs ($R - CSOC$)) *Un code convolutionnel systématique récursif de taux de codage $R = 1/2$, et représenté par les vecteurs $\boldsymbol{\alpha} = \{\alpha_1\, \alpha_2 \cdots \alpha_{J_1}\}$ et $\boldsymbol{\beta} = \{\beta_1\, \beta_2 \cdots \beta_{J_2}\}$, respectivement de J_1 et J_2 connexions, ainsi qu'en considérant la connexion virtuelle $\beta_0 = 0$, sera simplement orthogonal si et seulement si il vérifie les trois conditions suivantes, pour toute combinaison des entiers $l, j \in (1, \cdots, J_1)$ et $k, m \in (0, \cdots, J_2)$:*

1. *Les différences simples $(\alpha_l - \alpha_j)$ sont distinctes pour tout $l \neq j$.*
2. *Les différences simples $(\beta_k - \beta_m)$ sont distinctes pour tout $k \neq m$.*
3. *Les différences simples $(\alpha_l - \beta_k)$ sont distinctes.*

Á partir de ces symboles de syndrome, on peut, comme à la section 1.2.4.2, calculer les

équations d'inversion de parité. Nous obtenons ainsi deux ensembles d'équations d'inversion : un ensemble de J_1 équations orthogonales au bit u_i, notées $B^u_{l,i}$ avec $l = 1, \ldots, J_1$, et, $J_2 + 1$ équations orthogonales au symbole p_i, notées $B^p_{m,i}$ avec $m = 0, 1, \ldots, J_2$. Nous les obtenons aux équations (3.10) et (3.11) en suivant le raisonnement suivant :

$$\begin{aligned} B^u_{l,i} &= s_{i+\alpha_l} \oplus \tilde{u}_i \quad , l = 1, \ldots, J_1 \\ &= \sum_{\substack{j=1 \\ j \neq l}}^{J_1} \oplus \tilde{u}_{i+\alpha_l-\alpha_j} \oplus \sum_{k=0}^{J_2} \oplus \tilde{p}_{i+\alpha_l-\beta_k} \\ &= \sum_{\substack{j=1 \\ j \neq l}}^{J_1} \oplus e^u_{i+\alpha_l-\alpha_j} \oplus \sum_{k=0}^{J_2} \oplus e^p_{i+\alpha_l-\beta_k} \oplus \sum_{\substack{j=1 \\ j \neq l}}^{J_1} \oplus u_{i+\alpha_l-\alpha_j} \oplus \sum_{k=0}^{J_2} \oplus p_{i+\alpha_l-\beta_k} \end{aligned} \tag{3.8}$$

Or, on a d'après l'équation (3.1) :

$$p_{i+\alpha_l} = \sum_{j=1}^{J_1} \oplus u_{i+\alpha_l-\alpha_j} \oplus \sum_{k=1}^{J_2} \oplus p_{i+\alpha_l-\beta_k}$$

d'où, en permutant les termes u_i et $p_{i+\alpha_l}$ de part et d'autre du signe égal, et en reprenant $\beta_0 = 0$:

$$u_i = \sum_{\substack{j=1 \\ j \neq l}}^{J_1} \oplus u_{i+\alpha_l-\alpha_j} \oplus \sum_{k=0}^{J_2} \oplus p_{i+\alpha_l-\beta_k} \tag{3.9}$$

On obtient ainsi en reprenant (3.8) :

$$B^u_{l,i} = u_i \oplus \sum_{\substack{j=1 \\ j \neq l}}^{J_1} \oplus e^u_{i+\alpha_l-\alpha_j} \oplus \sum_{k=0}^{J_2} \oplus e^p_{i+\alpha_l-\beta_k} \quad , l = 1, \ldots, J_1 \tag{3.10}$$

Les équations $B^p_{m,i}$, pour $m = 0, 1, \ldots, J_2$ sont obtenues par un raisonnement similaire :

$$\begin{aligned} B^p_{m,i} &= s_{i+\beta_m} \oplus \tilde{p}_i \quad , m = 0, \ldots, J_2 \\ &= \sum_{j=1}^{J_1} \oplus \tilde{u}_{i+\beta_m-\alpha_j} \oplus \sum_{\substack{k=0 \\ k\neq m}}^{J_2} \oplus \tilde{p}_{i+\beta_m-\beta_k} \\ &= p_i \oplus \sum_{j=1}^{J_1} \oplus e^u_{i+\beta_m-\alpha_j} \oplus \sum_{\substack{k=0 \\ k\neq m}}^{J_2} \oplus e^p_{i+\beta_m-\beta_k} \quad , m = 0, \ldots, J_2 \end{aligned} \tag{3.11}$$

La principale différence par rapport aux codes non récursifs étudiés dans le chapitre précédent est qu'on utilise mainenant J_1 équations pour décoder le bit u_i, et J_2+1 équations pour le symbole p_i, contre uniquement $J + 1$ pour le bit u_i pour les codes non récursifs. Les équations (3.10) et (3.11) nous permettent ainsi, pour la première itération, de faire l'estimation du LRV du bit u_i et du symbole p_i selon les équations (3.12) et (3.13) [4] :

$$\lambda_u(i) = y^u_i + \sum_{l=1}^{J_1} \left(\sum_{\substack{j=1 \\ j\neq l}}^{J_1} \diamond y^u_{i+\alpha_l-\alpha_j} \diamond \sum_{k=0}^{J_2} \diamond y^p_{i+\alpha_l-\beta_k} \right) \tag{3.12}$$

$$\lambda_p(i) = y^p_i + \sum_{m=0}^{J_2} \left(\sum_{j=1}^{J_1} \diamond y^u_{i+\beta_m-\alpha_j} \diamond \sum_{\substack{k=0 \\ k\neq m}}^{J_2} \diamond y^p_{i+\beta_m-\beta_k} \right) \tag{3.13}$$

où le symbole $\diamond$ représente l'opération add-min définie dans [4].

Une autre différence par rapport aux codes non récursifs est qu'on n'utilise pas de rétroaction de la décision dans cet algorithme de décodage.

Cependant, nous utilisons toujours un décodeur itératif. On déduit alors à partir des

équations (3.12) et (3.13) l'estimation du LRV de u_i et p_i à l'itération μ [4] :

$$\lambda_u^{(\mu)}(i) = y_i^u + \sum_{r=1}^{J_1} \left(\sum_{\substack{s=1 \\ s\neq r}}^{J_1} \diamond \lambda_u^{(\mu-1)}(i+\alpha_r-\alpha_s) \diamond \sum_{t=0}^{J_2} \diamond \lambda_p^{(\mu-1)}(i+\alpha_r-\beta_t) \right) \quad (3.14)$$

$$\lambda_p^{(\mu)}(i) = y_i^p + \sum_{r=0}^{J_2} \left(\sum_{s=1}^{J_1} \diamond \lambda_u^{(\mu-1)}(i+\beta_r-\alpha_s) \diamond \sum_{\substack{t=0 \\ t\neq r}}^{J_2} \diamond \lambda_p^{(\mu-1)}(i+\beta_r-\beta_t) \right) \quad (3.15)$$

3.1.3 Détermination des conditions à respecter pour les codes $R - CSO^2C - WS$

Pour la détermination des conditions que les codes $R - CSO^2C - WS$ doivent vérifier, j'ai développé l'algorithme de décodage à la deuxième itération. Or, en réinsérant les équations (3.12) et (3.13) dans les équations (3.14) et (3.15) avec $\mu = 2$, nous obtenons :

$$\begin{aligned} \lambda_u^{(2)}(i) = y_i^u + \sum_{r=1}^{J_1} \Bigg(\sum_{\substack{s=1 \\ s\neq r}}^{J_1} \diamond \Bigg(y^u_{i+(\alpha_r-\alpha_s)} + \sum_{\substack{l=1 \\ l\neq s}}^{J_1} \Bigg(\sum_{\substack{j=1 \\ j\neq l}}^{J_1} \diamond y^u_{i+(\alpha_r-\alpha_s)-(\alpha_j-\alpha_l)} \diamond \\ \sum_{k=0}^{J_2} \diamond y^p_{i+(\alpha_r-\alpha_s)-(\beta_k-\alpha_l)} \Bigg) \Bigg) \diamond \sum_{t=0}^{J_2} \diamond \Bigg(y^p_{i+(\alpha_r-\beta_t)} + \sum_{\substack{m=0 \\ m\neq t}}^{J_2} \Bigg(\sum_{j=1}^{J_1} \diamond y^u_{i+(\alpha_r-\beta_t)-(\alpha_j-\beta_m)} \diamond \\ \sum_{\substack{k=0 \\ k\neq m}}^{J_2} \diamond y^p_{i+(\alpha_r-\beta_t)-(\beta_k-\beta_m)} \Bigg) \Bigg) \Bigg) \end{aligned} \quad (3.16)$$

$$\begin{aligned} \lambda_p^{(2)}(i) = y_i^p + \sum_{r=0}^{J_2} \Bigg(\sum_{s=1}^{J_1} \diamond \Bigg(y^u_{i+(\beta_r-\alpha_s)} + \sum_{\substack{l=1 \\ l\neq s}}^{J_1} \Bigg(\sum_{\substack{j=1 \\ j\neq l}}^{J_1} \diamond y^u_{i+(\beta_r-\alpha_s)-(\alpha_j-\alpha_l)} \diamond \\ \sum_{k=0}^{J_2} \diamond y^p_{i+(\beta_r-\alpha_s)-(\beta_k-\alpha_l)} \Bigg) \Bigg) \diamond \sum_{\substack{t=0 \\ t\neq r}}^{J_2} \diamond \Bigg(y^p_{i+(\beta_r-\beta_t)} + \sum_{\substack{m=0 \\ m\neq t}}^{J_2} \Bigg(\sum_{j=1}^{J_1} \diamond y^u_{i+(\beta_r-\beta_t)-(\alpha_j-\beta_m)} \diamond \\ \sum_{\substack{k=0 \\ k\neq m}}^{J_2} \diamond y^p_{i+(\beta_r-\beta_t)-(\beta_k-\beta_m)} \Bigg) \Bigg) \Bigg) \end{aligned} \quad (3.17)$$

Ainsi, les connexions des vecteurs α et β formeront un code $R-CSO^2C-WS$ si elles vérifient, pour que tous les termes de chacune des équations 3.16 et 3.17 soient différents, les quatre ensembles de conditions qu'on énumère ci-bas :

1. Pour toute combinaison des entiers $r, s, l, j \in (1, \ldots, J_1)$ et $t, m \in (0, 1, \ldots, J_2)$, tous les termes suivants doivent être distincts (indices des y^u dans l'équation (3.16)) :
   - $(\alpha_r - \alpha_s)$ pour $r \neq s$.
   - $(\alpha_r - \alpha_s) - (\alpha_j - \alpha_l)$ pour $r \neq s, r \neq j, s \neq l, j \neq l$.
   - $(\alpha_r - \beta_t) - (\alpha_j - \beta_m)$ pour $r \neq j, m \neq t$.
2. Pour toute combinaison des entiers $r, s, l \in (1, \ldots, J_1)$ et $t, m, k \in (0, 1, \ldots, J_2)$, tous les termes suivants doivent être distincts (indices des y^p dans l'équation (3.16)) :
   - $(\alpha_r - \beta_t)$.
   - $(\alpha_r - \alpha_s) - (\beta_k - \alpha_l)$ pour $r \neq s, s \neq l$.
   - $(\alpha_r - \beta_t) - (\beta_k - \beta_m)$ pour $t \neq m, m \neq k$.
3. Pour toute combinaison des entiers $s, l, j \in (1, \ldots, J_1)$ et $r, t, m \in (0, 1, \ldots, J_2)$, tous les termes suivants doivent être distincts (indices des y^u dans l'équation (3.17)) :
   - $(\beta_r - \alpha_s)$.
   - $(\beta_r - \alpha_s) - (\alpha_j - \alpha_l)$ pour $s \neq l, j \neq l$.
   - $(\beta_r - \beta_t) - (\alpha_j - \beta_m)$ pour $r \neq t, m \neq t$.
4. Pour toute combinaison des entiers $s, l \in (1, \ldots, J_1)$ et $r, t, m, k \in (0, 1, \ldots, J_2)$, tous les termes suivants doivent être distincts (indices des y^p dans l'équation (3.17)) :
   - $(\beta_r - \beta_t)$ pour $r \neq t$.

- $(\beta_r - \beta_t) - (\beta_k - \beta_m)$ pour $r \neq t$, $r \neq k$, $t \neq m$, $m \neq k$.
- $(\beta_r - \alpha_s) - (\beta_k - \alpha_l)$ pour $r \neq k$, $s \neq l$.

Les conditions $r \neq j$ dans l'ensemble 1 et $r \neq k$ dans l'ensemble 4 ont été rajoutées par rapport aux équations (3.16) et (3.17). Sinon, on aurait une répétition inévitable de différences. Cependant, ces conditions peuvent être simplifiées. En effet, si on regarde le premier ensemble de conditions, on voit que les différences simples $(\alpha_r - \alpha_s)$ doivent être distinctes, que les différences doubles $(\alpha_r - \alpha_s) - (\alpha_j - \alpha_l)$ doivent être distinctes, et que les différences simples doivent être distinctes des différences doubles. Ceci se résume par le fait que l'ensemble des connexions $\{\alpha_1\ \alpha_2 \cdots \alpha_{J_1}\}$ doit former un code $CSO^2C - WS$ comme défini à la définition 2.1.

De la même manière, le dernier ensemble de conditions implique que les connexions $\{\beta_0\ \beta_1 \cdots \beta_{J_2}\}$ doivent aussi former un code $CSO^2C - WS$. De plus, les deux conditions restantes du premier et dernier ensemble, soit respectivement $(\alpha_r - \beta_t) - (\alpha_j - \beta_m)$ et $(\beta_r - \alpha_s) - (\beta_k - \alpha_l)$ sont identiques et donnent les mêmes différences. Elles peuvent se réécrire selon $(\alpha_r - \alpha_j) - (\beta_t - \beta_m)$ dinstinctes pour $r \neq j$, $t \neq m$. De plus, ces différences doivent être distinctes des différences simples et doubles des connexions $\{\alpha_1\ \alpha_2 \cdots \alpha_{J_1}\}$ et $\{\beta_0\ \beta_1 \cdots \beta_{J_2}\}$.

Enfin, le deuxième ensemble de conditions est identique au troisième ensemble au signe $-$ près. On peut ainsi définir les codes $R - CSO^2C - WS$:

Définition 3.2 (Codes convolutionnels doublement orthogonaux récursifs au sens large) *$(R - CSO^2C - WS)$ Un code convolutionnel systématique récursif de taux de codage*

$R = 1/2$, et représenté par les vecteurs $\boldsymbol{\alpha} = \{\alpha_1\ \alpha_2 \cdots \alpha_{J_1}\}$ et $\boldsymbol{\beta} = \{\beta_1\ \beta_2 \cdots \beta_{J_2}\}$, respectivement de J_1 et J_2 connexions, avec la connexion virtuelle $\beta_0 = 0$, sera doublement orthogonal au sens large si et seulement si il vérifie les quatre conditions suivantes, pour toute combinaison des entiers $l, j \in (1, \cdots, J_1)$ et $k, m \in (0, \cdots, J_2)$:

1. *L'ensemble des connexions $\{\alpha_1\ \alpha_2 \cdots \alpha_{J_1}\}$ forme un code $CSO^2C - WS$.*
2. *L'ensemble des connexions $\{\beta_0\ \beta_1 \cdots \beta_{J_2}\}$ forme un code $CSO^2C - WS$.*
3. *Les différences $(\alpha_r - \alpha_j) - (\beta_t - \beta_m)$ pour $r \neq j$ et $t \neq m$ sont distinctes et différentes des différences simples et doubles engendrées par les connexions $\{\alpha_1\ \alpha_2 \cdots \alpha_{J_1}\}$ et $\{\beta_0\ \beta_1 \cdots \beta_{J_2}\}$.*
4. *Pour toute combinaison des entiers $s, l, j \in (1, \ldots, J_1)$ et $r, t, m \in (0, 1, \ldots, J_2)$, tous les termes suivants doivent être distincts :*
   - *$(\beta_r - \beta_t)$ pour $r \neq t$.*
   - *$(\beta_r - \alpha_s) - (\alpha_j - \alpha_l)$ pour $s \neq l$ et $j \neq l$.*
   - *$(\beta_r - \beta_t) - (\alpha_j - \beta_m)$ pour $r \neq t$ et $m \neq t$.*

Nous avons donc déterminé une définition spécifique aux codes $R - CSO^2C - WS$. L'étape suivante est la génération de nouveaux codes $R - CSO^2C - WS$, expliquée dans la prochaine section. Pour cette recherche, nous avons utilisé trois méthodes de construction de codes, en s'inspirant des méthodes de [1] qui servaient à construire des codes $CSO^2C - WS$.

3.2 Construction des nouveaux codes $R - CSO^2C - WS$

3.2.1 Méthode directe

La première méthode utilisée est la plus simple à concevoir. En effet, on va prendre les entiers dans l'ordre croissant et vérifier à chaque fois si les deux générateurs de connexions vérifient les conditions énoncées dans la définition 3.2. Le principe en est expliqué dans l'organigramme 3.4.

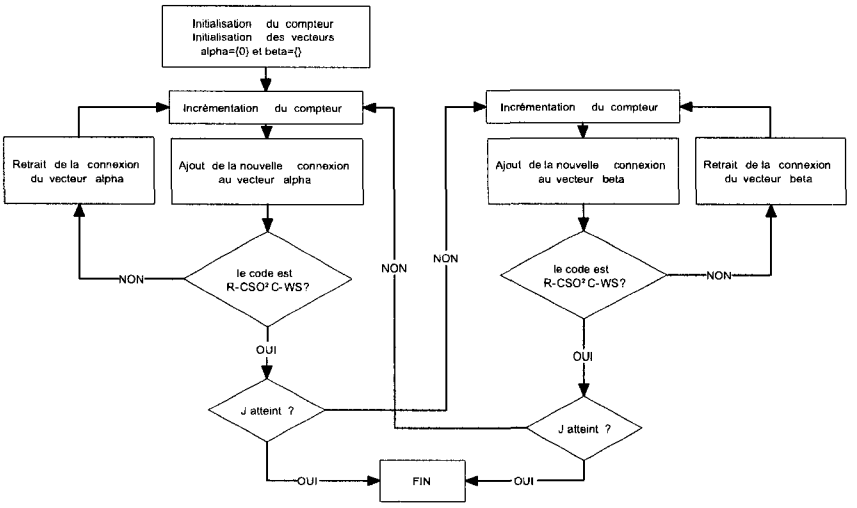

FIG. 3.4 Algorithme de recherche par méthode directe des codes $R - CSO^2C - WS$

On a donc au départ uniquement besoin du nombre de connexions $J = J_1 + J_2$ recherchées. On ajoute alors alternativement une connexion à chacun des générateurs en incrémentant un compteur dans l'ordre des entiers croissants jusqu'à ce que les deux générateurs forment un code $R - CSO^2C - WS$. Un codeur à $J + 1$ connexions est donc simplement déduit du codeur trouvé avec J connexions en rajoutant à un des deux générateurs, selon le cas, la connexion manquante. Les générateurs obtenus par la méthode directe sont indiqués

dans le tableau 3.1 suivant :

J	J_1	J_2	α	β	span
3	2	1	[0 1]	[4]	4
4	3	1	[0 1 11]	[4]	11
5	3	2	[0 1 11]	[4 38]	38
6	4	2	[0 1 11 85]	[4 38]	85
7	4	3	[0 1 11 85]	[4 38 189]	189
8	5	3	[0 1 11 85 401]	[4 38 189]	401
9	5	4	[0 1 11 85 401]	[4 38 189 723]	723
10	6	4	[0 1 11 85 401 1189]	[4 38 189 723]	1189
11	6	5	[0 1 11 85 401 1189]	[4 38 189 723 2068]	2068
12	7	5	[0 1 11 85 401 1189 3392]	[4 38 189 723 2068]	3392
13	7	6	[0 1 11 85 401 1189 3392]	[4 38 189 723 2068 5569]	5569
14	8	6	[0 1 11 85 401 1189 3392 8176]	[4 38 189 723 2068 5569]	8176

TAB. 3.1 Codeurs $R - CSO^2C - WS$ obtenus par la méthode directe.

3.2.2 Méthode pseudo aléatoire

Cette deuxième méthode a été introduite lors de la recherche de codes $CSO^2C - WS$ dans [2]. En effet, dans la génération des codeurs $CSO^2C - WS$, le but est d'en réduire au maximum le span pour un nombre de connexions donné, et la différence par rapport à la méthode directe est que chaque connexion trouvée doit être validée par un test aléatoire de probabilité L/J, où L est le numéro de la connexion courante et J le nombre total de connexions [14]. L'idée est que les générateurs les plus petits trouvables pour J connexions ne correspondent pas au début des générateurs les plus petits trouvables pour $J + 1$ connexions. Ainsi, on rejettera plus de connexions valides au début, alors que, lorsqu'on cherchera la dernière connexion des générateurs, on choisira la première valide sous peine d'augmenter inutilement la mémoire des générateurs. L'algorithme de recherche par

méthode pseudo-aléatoire est présenté à l'organigramme 3.5.

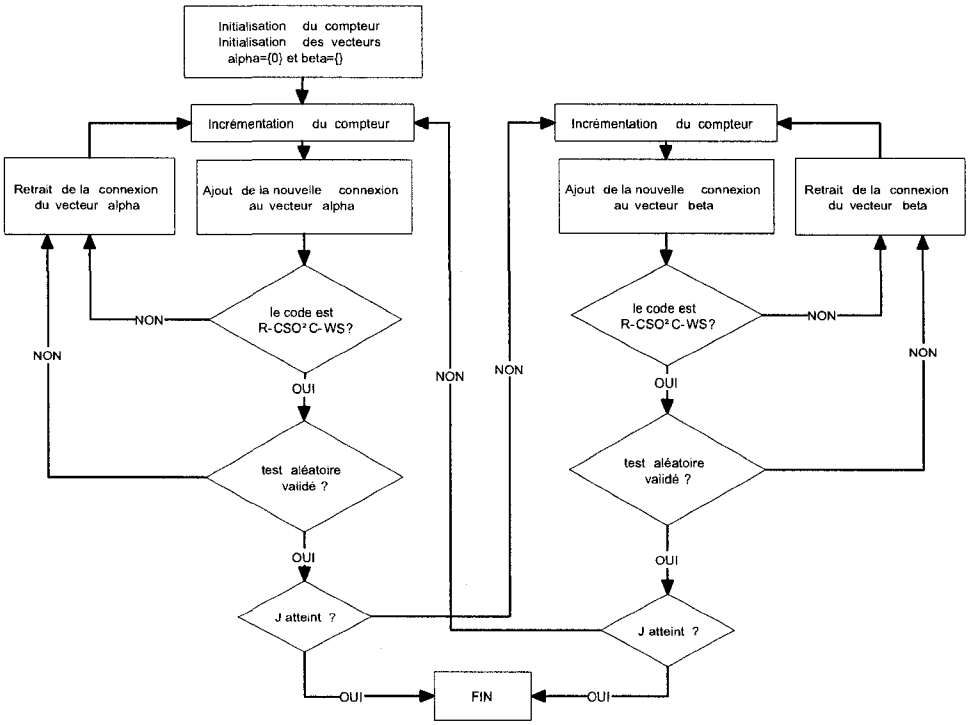

FIG. 3.5 Algorithme de recherche par méthode pseudo-aléatoire des codes $R - CSO^2C - WS$

Pour chaque valeur de J, 20 essais ont été réalisés. Les meilleurs résultats, en termes de span, sont présentés dans le tableau 3.2.

On peut remarquer que les mémoires des codes obtenus par la méthode de recherche pseudo-aléatoire sont inférieures à celles qu'on avait trouvées avec la méthode directe. Le tableau 3.3 compare, pour chaque valeur de J, les spans obtenus par les deux méthodes.

J	J_1	J_2	α	β	span
3	2	1	[0 2]	[3]	3
4	3	1	[0 2 10]	[7]	10
5	3	2	[0 19 22]	[20 27]	27
6	4	2	[0 8 15 67]	[13 55]	67
7	4	3	[0 3 17 88]	[5 35 149]	149
8	5	3	[0 3 30 104 296]	[7 35 214]	296
9	5	4	[0 2 12 103 385]	[3 40 226 539]	539
10	6	4	[0 3 14 92 455 905]	[10 47 216 595]	905
11	6	5	[0 3 21 116 417 1146]	[10 50 204 683 1831]	1831
12	7	5	[0 11 19 90 421 1288 3128]	[12 29 194 755 2098]	3128

TAB. 3.2 Meilleurs codeurs $R - CSO^2C - WS$ obtenus par la méthode pseudo-aléatoire.

J / Méthode	3	4	5	6	7	8	9	10	11	12
directe	4	11	38	85	189	401	723	1189	2068	3392
ps. aléatoire	3	10	27	67	149	296	539	905	1831	3128

TAB. 3.3 Comparaison des spans des codes $R - CSO^2C - WS$ obtenus par les méthodes de construction directe et pseudo-aléatoire.

3.2.3 Méthode exhaustive

La dernière méthode utilisée est la méthode exhaustive. On aura, grâce à cette méthode, les générateurs les plus courts possibles en terme de span, et qui sont donc optimaux. Cependant, il a été impossible de trouver les générateurs optimaux avec un nombre total de connexions supérieur à $J = 7$, le temps de simulation devenant trop important. Il a en effet fallu un mois de simulation pour obtenir les codeurs les plus courts à $J = 7$.

L'algorithme de recherche par méthode exhaustive est présenté à la figure 3.6. Le but est d'essayer toutes les combinaisons possibles pour obtenir le générateur le plus court pour un nombre de connexions J donné.

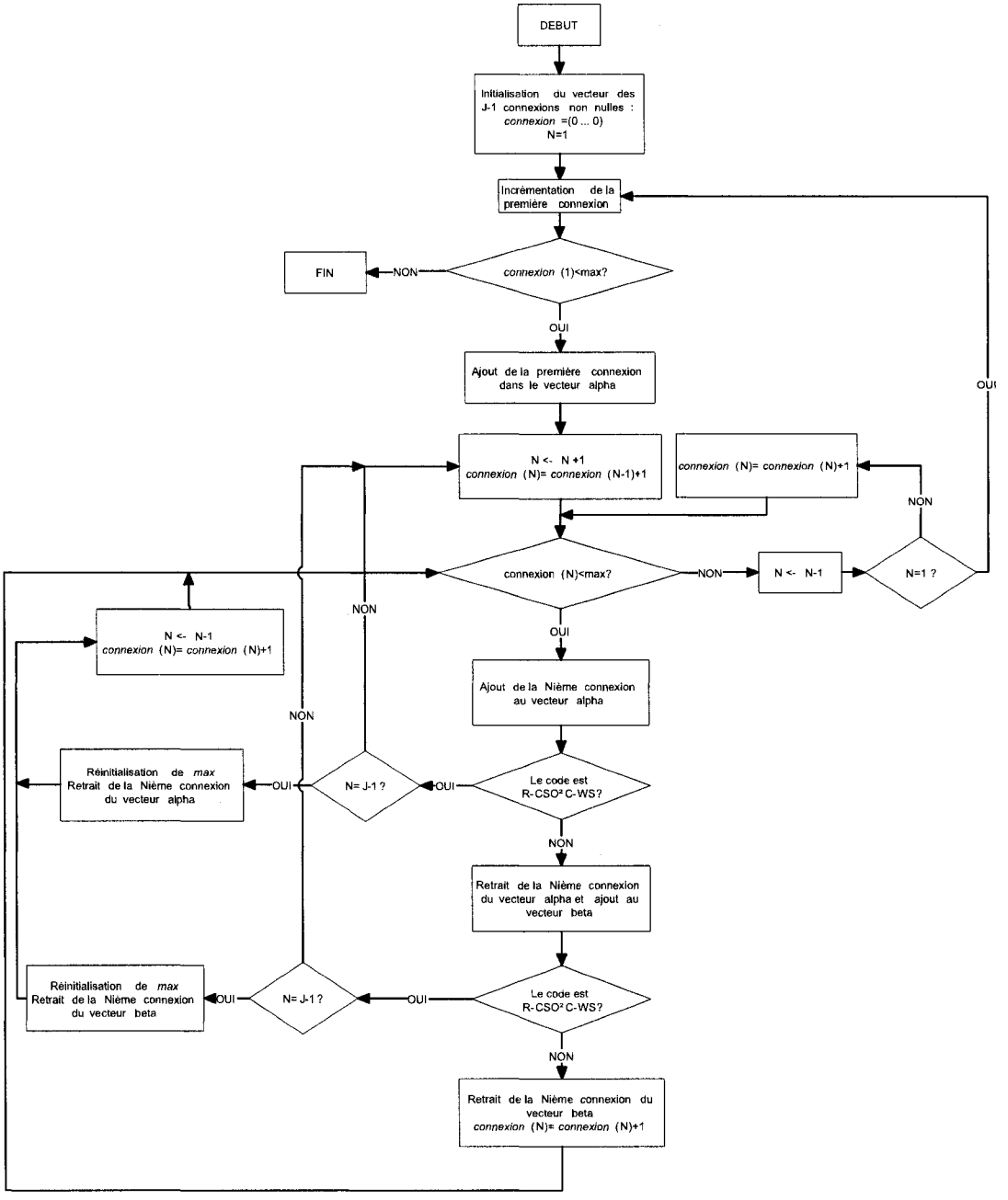

FIG. 3.6 Algorithme de recherche par méthode exhaustive des codes $R - CSO^2C - WS$

3.2.3.1 Description de l'algorithme

L'algorithme effectue une recherche exhaustive, permettant de trouver les codes $R - CSO^2C - WS$ de plus petit span, pour un J donné. Aucune restriction sur les valeurs de J_1 ou J_2 n'a été prise lors de la recherche. L'algorithme se sert d'un vecteur $connexion$ qui stocke toutes les valeurs, dans l'ordre croissant, des connexions non nulles des vecteurs $\boldsymbol{\alpha}$ et $\boldsymbol{\beta}$. Il y a donc $J-1$ connexions non nulles car le vecteur $\boldsymbol{\alpha}$ contient toujours la valeur 0. Le numéro de la connexion non nulle en cours de recherche est noté N : on a donc $N \in \{1, \ldots, J-1\}$. Enfin, la plus petite valeur du span obtenue pour un J donné est notée max. C'est cette valeur qu'on cherche à réduire. Au départ, le vecteur $connexion$ vaut 0, $N = 1$, et max est initialisé à l'infini.

On incrémente la valeur de la première connexion et on la place dans $\boldsymbol{\alpha}$. Le code est alors de taille $J = 2$, et vérifie toujours les conditions d'un code $R - CSO^2C - WS$. On passe alors à la connexion suivante, qui prend la valeur de la première connexion à laquelle on ajoute une unité. On la place successivement dans les vecteurs $\boldsymbol{\alpha}$ et $\boldsymbol{\beta}$. Si, dans un des deux cas, le code forme un code $R - CSO^2C - WS$, on passe à la connexion suivante, qui prend la valeur de la connexion qui vient de remplir les conditions à laquelle on ajoute une unité. On continue ainsi jusqu'à ce qu'on obtienne un code avec J connexions. Alors, on peut réinitialiser la valeur de max qui vaut le dernier terme du vecteur $connexion$, et on reprend la recherche en incrémentant l'avant dernière connexion.

Dans le cas où la connexion ne satisfait pas les conditions, en étant ni dans le vecteur $\boldsymbol{\alpha}$, ni dans le vecteur $\boldsymbol{\beta}$, on l'incrémente d'une unité, et on reteste dans chacun des vecteurs.

Quand cette connexion a été incrémentée jusqu'à la valeur courante de max, on revient à la connexion précédente car nous ne pourrons améliorer les résultats avec cette connexion.

Á la fin, quand la première connexion est incrémentée jusqu'à max, on sait que toutes les possiblités ont été testées. La valeur max correspond alors, pour un J donné, à la plus petite valeur possible de span pour un code $R - CSO^2C - WS$.

3.2.3.2 Résultats

Les résultats obtenus par la méthode exhaustive sont présentés dans le tableau 3.4.

J	J_1	J_2	α	β	span
3	2	1	[0 2]	[3]	3
4	3	1	[0 1 10]	[6]	10
5	3	2	[0 1 24]	[4 17]	24
6	4	2	[0 1 49 53]	[13 42]	53
7	4	3	[0 1 78 110]	[4 66 91]	110

TAB. 3.4 Meilleurs codeurs $R - CSO^2C - WS$ obtenus par la méthode exhaustive.

La méthode exhaustive donne les codes les plus courts possibles. Les spans des codes obtenus sont donc, comme on le voit dans le tableau 3.5, plus petits que ceux qu'on a eu par les méthodes de construction directe et pseudo-aléatoire.

J / Méthode	3	4	5	6	7	8	9	10	11	12
directe	4	11	38	85	189	401	723	1189	2068	3392
ps. aléatoire	3	10	27	67	149	296	539	905	1831	3128
exhaustive	3	10	24	53	110					

TAB. 3.5 Comparaison des spans des codes $R - CSO^2C - WS$ obtenus par les trois méthodes de construction.

On peut toutefois noter que la méthode pseudo-aléatoire avait permis de trouver les

codes optimaux pour J=3 et J=4. De plus, on remarque que plus J augmente, et plus la méthode exhaustive permet de réduire la mémoire des codes.

3.3 Performances d'erreur obtenues par simulations

Les trois méthodes de génération de codes $R - CSO^2C - WS$ décrites dans la section précédente nous ont permis de construire des nouveaux codes. De plus, la méthode exhaustive nous assure d'avoir les codes optimaux jusqu'à J=7. Nous allons maintenant présenter les résultats des simulations de ces nouveaux codes en se servant de l'algorithme de décodage itératif présenté aux équations (3.12) à (3.15).

3.3.1 Amélioration de l'algorithme de décodage

Les lecteurs avertis auront remarqué que l'algorithme de décodage présenté dans cette recherche, et défini par les équations (3.12) à (3.15), diffère de l'algorithme de décodage utilisé dans [4] qui était similaire au décodage itératif de faible complexité des codes LDPC présenté dans [9]. La raison est simplement qu'après plusieurs simulations, les performances d'erreur des codes $R - CSO^2C - WS$ se sont avérées meilleures en se servant du décodeur présenté à la section 3.1.2 .

Une autre amélioration a été possible en réinsérant dans le calcul des équations (3.12) à (3.15) les valeurs déjà décodées. Reprenons par exemple l'équation (3.12) :

$$\lambda_u(i) = y_i^u + \sum_{l=1}^{J_1} \left(\sum_{\substack{j=1 \\ j \neq l}}^{J_1} \diamond y_{i+\alpha_l-\alpha_j}^u \diamond \sum_{k=0}^{J_2} \diamond y_{i+\alpha_l-\beta_k}^p \right)$$

Cette équation représente l'estimation du LRV du bit u_i, à l'instant i, pour $i = 0, 1, \ldots$, pour la première itération. Ainsi, les termes $\lambda(i+\alpha_l-\alpha_j)$ avec $\alpha_l-\alpha_j < 0$ et $\lambda(i+\alpha_l-\beta_k)$ avec $\alpha_l - \beta_k < 0$ ont déjà été décodés. Il est donc possible de remplacer les termes $y^u_{i+\alpha_l-\alpha_j}$, avec $\alpha_l - \alpha_j < 0$, et $y^p_{i+\alpha_l-\beta_k}$, avec $\alpha_l - \beta_k < 0$, respectivement par $\lambda_u(i + \alpha_l - \alpha_j)$ et $\lambda_p(i + \alpha_l - \beta_k)$. En effectuant, par le même principe, ces modifications pour chacune des équations (3.12) à (3.15), on a réussi à améliorer les performances d'erreur.

3.3.2 Détermination des coefficients de pondération

Comme pour les codes $CSO^2C - WS$ non récursifs présentés au chapitre précédent, il n'y a pas une indépendance totale des observables à la deuxième itération à cause de certaines répétitions inévitables, dûes aux permutations d'indices. Afin de réduire l'intercorrélation des observables à la deuxième itération, nous utilisons à nouveau des coefficients de pondération. Mais nous faisons maintenant, à chaque itération, l'estimation des LRV de u_i et de p_i. Nous introduisons donc deux coefficients, a_u et a_p, qui agiront respectivement sur les λ_u et les λ_p. Comme pour les codes non récursifs, ces coefficients seront pris identiques

à chaque itération. L'algorithme de décodage devient donc :

pour la première itération :

$$\lambda_u(i) = a_u \left(y_i^u + \sum_{l=1}^{J_1} \left(\sum_{\substack{j=1 \\ j \neq l}}^{J_1} \diamond y_{i+\alpha_l-\alpha_j}^u \diamond \sum_{k=0}^{J_2} \diamond y_{i+\alpha_l-\beta_k}^p \right) \right)$$

$$\lambda_p(i) = a_p \left(y_i^p + \sum_{m=0}^{J_2} \left(\sum_{j=1}^{J_1} \diamond y_{i+\beta_m-\alpha_j}^u \diamond \sum_{\substack{k=0 \\ k \neq m}}^{J_2} \diamond y_{i+\beta_m-\beta_k}^p \right) \right)$$

pour l'itération $\mu > 1$:

$$\lambda_u^{(\mu)}(i) = a_u \left(y_i^u + \sum_{r=1}^{J_1} \left(\sum_{\substack{s=1 \\ s \neq r}}^{J_1} \diamond \lambda_u^{(\mu-1)}(i+\alpha_r-\alpha_s) \diamond \sum_{t=0}^{J_2} \diamond \lambda_p^{(\mu-1)}(i+\alpha_r-\beta_t) \right) \right)$$

$$\lambda_p^{(\mu)}(i) = a_p \left(y_i^p + \sum_{r=0}^{J_2} \left(\sum_{s=1}^{J_1} \diamond \lambda_u^{(\mu-1)}(i+\beta_r-\alpha_s) \diamond \sum_{\substack{t=0 \\ t \neq r}}^{J_2} \diamond \lambda_p^{(\mu-1)}(i+\beta_r-\beta_t) \right) \right)$$

Ainsi, si on considère le code du tableau 3.2, pour $J_1 = 3$ et $J_2 = 1$, représenté par les vecteurs $\boldsymbol{\alpha} = \{0\ 2\ 10\}$ et $\boldsymbol{\beta} = \{7\}$, nous pouvons, pour un rapport signal sur bruit donné, simuler le code en faisant varier alternativement chacun des coefficients a_u et a_p de 0.1 à 1 par pas de 0.1 pour trouver le meilleur duo de coefficients. Les résultats présentés à la figure 3.7 montrent la probabilité d'erreur par bit obtenue dans chacun des cas possibles pour $E_b/N_0 = 5\ dB$. Pour des raisons de clarté, uniquement les résultats des simulations pour des coefficients tels que $a_u > 0.2$ et $a_p > 0.3$, qui donnent les meilleurs résultats, sont présentés.

Nous pouvons ainsi lire que les meilleures performances, pour $E_b/N_0 = 5dB$, seront obtenues pour les coefficients $a_u = 0.5$ et $a_p = 0.6$, avec une probabilité d'erreur par

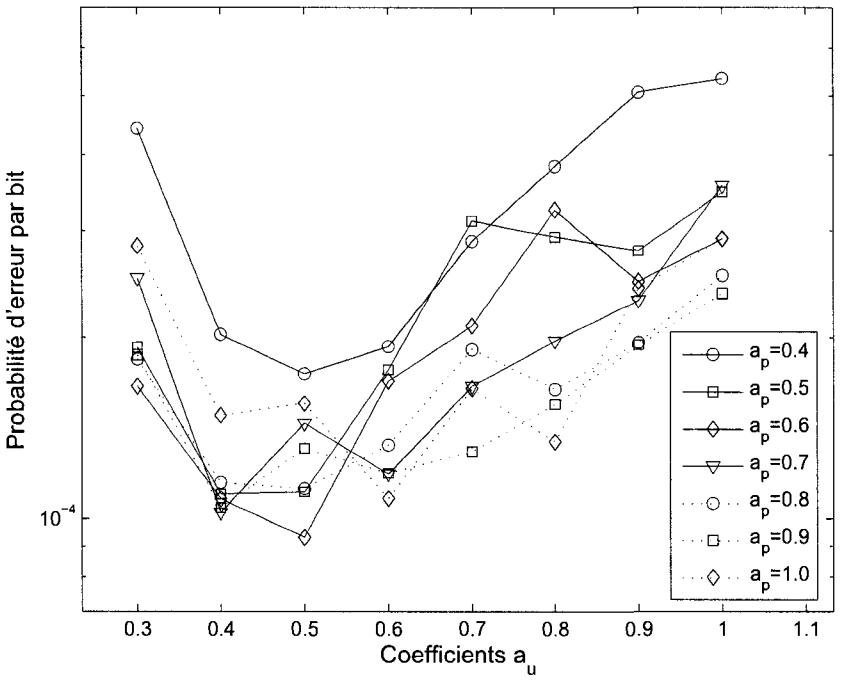

FIG. 3.7 Effet des coefficients de pondération sur les performances d'erreur du code $R - CSO^2C - WS$ avec R=1/2 et représenté par les vecteurs $\boldsymbol{\alpha} = \{0\ 2\ 10\}$ et $\boldsymbol{\beta} = \{7\}$ après 10 itérations, $E_b/N_0 = 5dB$

bit d'environ $8 \cdot 10^{-5}$. Il est bien évidemment possible de calculer tous les coefficients optimaux pour chaque valeur de E_b/N_0. Cependant, les performances sont peu sensibles aux variations des coefficients. En effet, toujours pour le code $R-CSO^2C-WS$ représenté par les vecteurs $\boldsymbol{\alpha} = \{0\ 2\ 10\}$ et $\boldsymbol{\beta} = \{7\}$, la recherche des meilleurs coefficients de pondération à chaque valeur de E_b/N_0 donne les résultats présentés dans le tableau 3.6 :

$E_b/N_0\ (dB)$	1	1.5	2	2.5	3	3.5	4	4.5	5
a_u	0.9	0.7	0.7	0.6	0.9	0.7	0.7	0.6	0.5
a_p	0.5	0.7	0.6	0.5	0.4	0.4	0.5	0.4	0.6

TAB. 3.6 Meilleurs coefficients de pondération, en fonction de E_b/N_0, obtenus pour le code $R - CSO^2C - WS$ représenté par les vecteurs $\boldsymbol{\alpha} = \{0\ 2\ 10\}$ et $\boldsymbol{\beta} = \{7\}$

Les performances d'erreur de ce code, pour un rapport signal sur bruit variant de 1 à 5 dB, après 10 itérations, et en utilisant les meilleurs coefficients de pondération à chaque valeur de E_b/N_0, sont présentées à la figure 3.8. On y indique aussi les performances d'erreur obtenues pour les 10 premières itérations si on garde uniquement les coefficients optimaux à $5dB$, soit $a_u = 0.5$ et $a_p = 0.6$, comme trouvés à la figure 3.7.

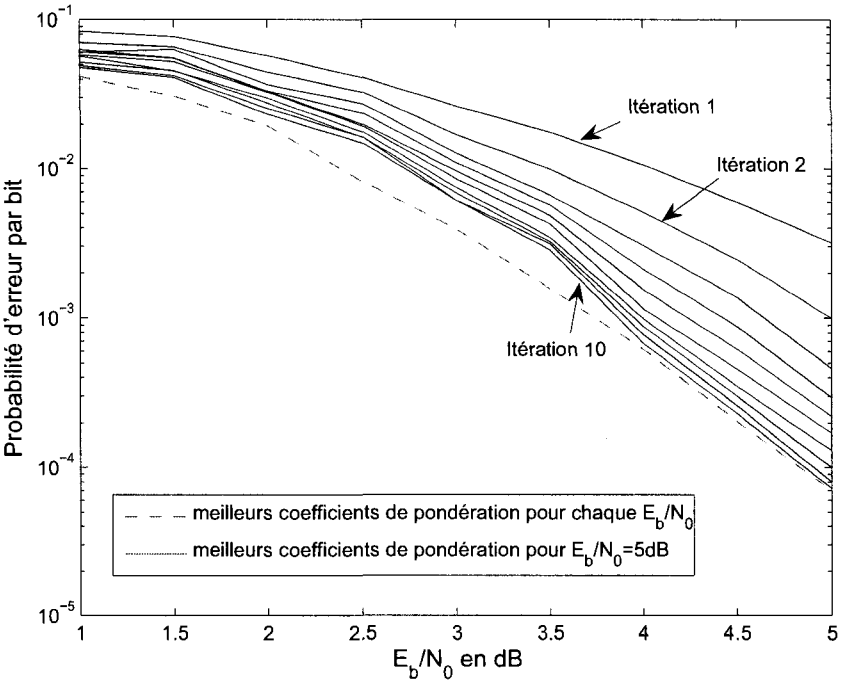

FIG. 3.8 Influence des coefficients de pondération utilisés sur la probabilité d'erreur

On peut ainsi observer qu'il est inutile de chercher les coefficients de pondération pour chaque valeur de E_b/N_0, puisque les performances d'erreur restent sensiblement les mêmes pour les plus faibles SNR. On cherchera ainsi les coefficients de pondération pour des valeurs de E_b/N_0 où les probabilités d'erreur par bit seront de l'ordre de 1×10^{-5}.

3.3.3 Spectres des codes $R - CSO^2C - WS$

Nous avons vu au premier chapitre qu'il était possible de calculer le spectre des distances des codes convolutionnels récursifs afin d'évaluer des bornes supérieures sur les performances d'erreur des codes. Les spectres indiquent, comme mentionné à la section 1.3.3, le nombre de mots de code A_d de poids d, ainsi que le nombre C_d de bit d'information valant "1" ayant contribués à l'obtention des A_d mots de code de poids d. Bien que nous n'utilisons pas le même type de décodeur dans cette recherche que celui qui a servi à obtenir l'estimation de la probabilité d'erreur dans l'équation (1.22), nous pouvons quand même nous donner une bonne idée sur les performances des codes.

Ainsi, si on considère le code $R - CSO^2C - WS$ pour $J = J_1 + J_2 = 5$ et représenté par les vecteurs $\alpha = \{0\ 1\ 24\}$ et $\beta = \{14\ 20\}$, les 15 premières raies du spectre sont représentées dans le tableau 3.7 :

d	A_d	C_d
1	0	0
2	0	0
3	0	0
4	0	0
5	0	0
6	1	3
7	0	0
8	0	0
9	0	0
10	6	30
11	0	0
12	20	120
13	0	0
14	54	378
15	0	0

TAB. 3.7 Spectre du code $R - CSO^2C - WS$ représenté par $\alpha = \{0\ 1\ 24\}$ et $\beta = \{14\ 20\}$

Ainsi, les deux premières raies non nulles du spectre sont de poids $d_{free} = J+1 = 6$ et $2J = 10$. Nous pouvons vérifier que ces observations sont généralisables à tous les codes $R - CSO^2C - WS$. En effet, si on regarde le spectre du code pour $J = 4$ représenté par les vecteurs $\boldsymbol{\alpha} = \{0\ 2\ 10\}$ et $\boldsymbol{\beta} = \{7\}$, nous obtenons, sans représenter les raies de poids inférieurs au poids de la première raie non nulle, appellée raie solide :

d	A_d	C_d
5	1	2
6	0	0
7	0	0
8	4	14
9	1	6
10	6	24
11	22	104
12	16	102
13	60	336
14	154	962
15	223	1618

TAB. 3.8 Spectre du code $R - CSO^2C - WS$ représenté par $\boldsymbol{\alpha} = \{0\ 2\ 10\}$ et $\boldsymbol{\beta} = \{7\}$

Á nouveau, les deux premières raies ont des poids respectifs de $d_{free} = J + 1 = 5$ et $2J = 8$. Nous pouvons aussi remarquer que, pour nos codes $RCSO^2C - WS$, il y a toujours un seul mot de code de poids $J + 1$ et que le coefficient C_d associé à la raie solide du spectre, vaut $J_2 + 1$ dans les deux tableaux 3.7 et 3.8.

Nous pouvons calculer un autre spectre pour confirmer ces informations. Considérons le code représenté par les deux vecteurs $\boldsymbol{\alpha} = \{0\ 1\ 20\ 24\}$ et $\boldsymbol{\beta} = \{13\}$. On a donc $J = J_1 + J_2 = 4 + 1 = 5$. Nous nous attendons donc, d'après les observations précédentes, à avoir une première raie pour un poids de $d_{free} = J + 1 = 6$. Pour ce poids, il ne devrait y avoir qu'un mot de code, obtenu à l'aide de $J_2 + 1 = 2$ bits d'information "1" en entrée du

codeur. Enfin, la deuxième raie doit être au poids $2J = 10$. Le spectre est donné au tableau 3.9.

d	A_d	C_d
6	1	2
7	0	0
8	0	0
9	0	0
10	7	26
11	0	0
12	22	100
13	0	0
14	67	350
15	0	0

TAB. 3.9 Spectre du code $R - CSO^2C - WS$ représenté par $\alpha = \{0\ 1\ 20\ 24\}$ et $\beta = \{13\}$

Ainsi, si on trace l'estimation de la probabilité d'erreur par bit, selon l'équation (1.22), pour les trois codes dont on a calculé les spectres, nous obtenons les performances d'erreur représentées à la figure 3.9 :

On voit sur la courbe que les codes avec $J = 5$ ont de meilleures performances d'erreur que le code avec $J = 4$, ce qui pouvait se prévoir puisque d'après les spectres calculés, on a pour le code avec $J = 4$ une distance libre de 5, alors qu'elle est de 6 pour les deux autres codes. Aussi, si on regarde les deux codes avec $J = 5$, on voit qu'on a une faible amélioration des performances d'erreur en maximisant J_1 par rapport à J_2. Cependant, pour $J > 5$, le fait de prendre J_1 le plus proche possible de J entraînera une augmentation du span pour des performances qui resteront du même ordre de grandeur. Ainsi, la recherche des codes a été faite sans chercher à maximiser J_1.

Nous pouvons confirmer ces résultats par des simulations. Les trois codes présentés

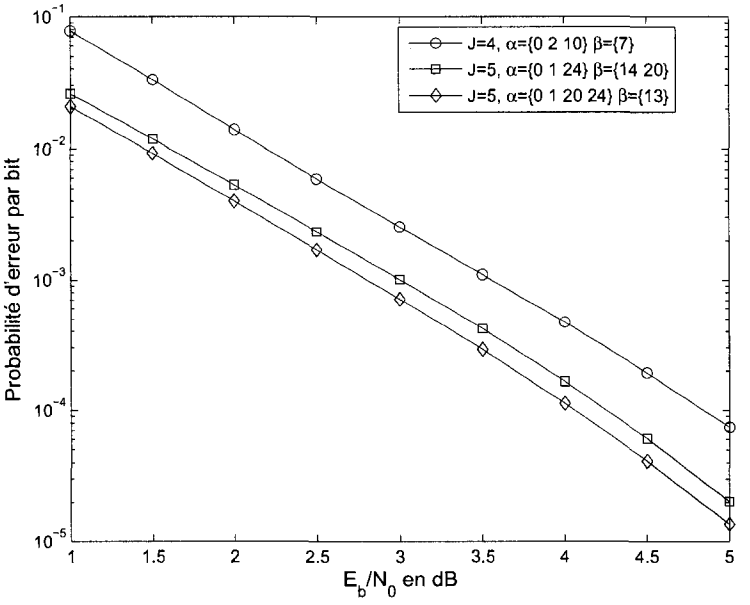

FIG. 3.9 Estimation de la probabilité d'erreur de 3 codes à partir de leur spectres

dans le tableau 3.10 sont tous tels que $J = J_1 + J_2 = 5$. Mais ils ont tous des valeurs de J_1 et J_2 différentes.

J	J_1	J_2	$\boldsymbol{\alpha}$	$\boldsymbol{\beta}$
5	2	3	[0 13]	[1 20 24]
5	3	2	[0 1 24]	[14 20]
5	4	1	[0 1 20 24]	[13]

TAB. 3.10 Codes $R - CSO^2C - WS$ avec différents J_1 et J_2 pour $J = 5$ fixé

Les performances d'erreur des codes sont données à la figure 3.10. On indique également sur la figure 3.10 les performances des codes du tableau 3.4 pour $J = 4$ et $J = 6$.

On observe que les courbes obtenues sont en accord avec ce que les spectres avaient prédits, du moins pour $E_b/N_0 > 3.5dB$. En dessous de cette valeur, les différents codes

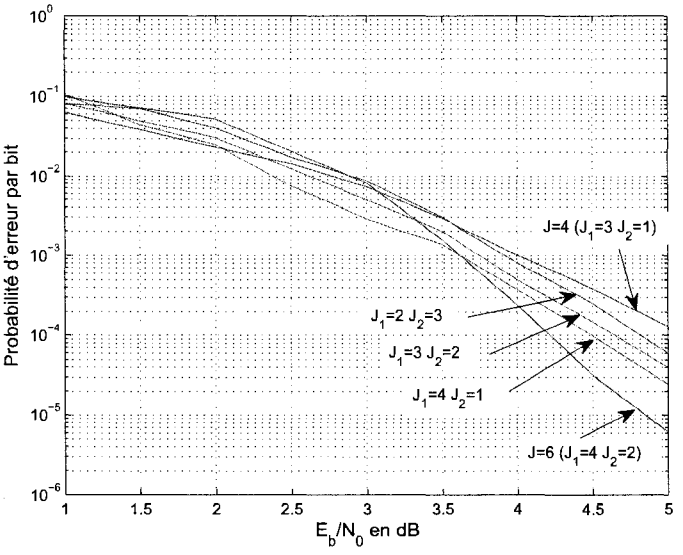

FIG. 3.10 Influence de J_1 et J_2 pour $J = 5$ fixé, et comparaison des performances d'erreur avec des codes ayant J=4 et J=6, après 6 itérations

donnent les mêmes ordres de grandeur de performances d'erreur. Nous verrons dans la prochaine section que cette remarque semble toujours valable. Mais pour $E_b/N_0 > 3.5dB$, entre les trois codes avec $J = 5$, celui qui a le plus grand J_1 offre les meilleures performances. Cependant, nous pouvons voir que le code avec $J = 6$ est meilleur que les trois codes avec $J = 5$, qui sont eux mêmes plus performants que le code avec $J = 4$. Ainsi, bien qu'à J fixé, il soit préférable d'avoir J_1 le plus grand possible, nous chercherons avant tout à réduire au maximum le span des codes obtenus, et ainsi la latence au décodage. La prochaine section présente ainsi les performances d'erreur des codes $R - CSO^2C - WS$ pour J variant de 3 à 11, avec les spans minimum obtenus.

3.3.4 Performances d'erreur obtenues pour les codes $R - CSO^2C - WS$

Les codes simulés sont présentés dans le tableau 3.11. Ce sont les codes les plus courts que nous avons obtenus. Nous indiquons aussi le span des plus petits codes $CSO^2C - WS$ connus [14] pour comparer. Jusqu'à $J = 7$, les codes ont été obtenus par la méthode ex-

J	J_1	J_2	α	β	$CSO^2C - WS$
3	2	1	[0 2]	[3]	5
4	3	1	[0 2 10]	[7]	15
5	3	2	[0 1 24]	[4 17]	33
6	4	2	[0 1 49 53]	[13 42]	98
7	4	3	[0 1 78 110]	[4 66 91]	222
8	5	3	[0 3 30 104 296]	[7 35 214]	459
9	5	4	[0 2 12 103 385]	[3 40 226 539]	912
10	6	4	[0 3 14 92 455 905]	[10 47 216 595]	1698
11	6	5	[0 3 21 116 417 1146]	[10 50 204 683 1831]	3490

TAB. 3.11 Codes $R - CSO^2C - WS$ les plus courts, simulés pour J variant de 3 à 11

haustive. Les autres codes ont été obtenus par la méthode de construction pseudo-aléatoire. Nous pouvons remarquer que pour un nombre de connexions donné, les codes récursifs sont beaucoup plus courts que les codes non récursifs, surtout que les span pour les codes récursifs pour $J > 7$ ne sont pas optimum.

Pour chacun des codes, une recherche des meilleurs coefficients de pondération a été effectuée au préalable à $E_b/N_0 = 5dB$ pour les codes ayant un J entre 4 et 7, $E_b/N_0 = 4dB$ pour J=8 et J=9 et $E_b/N_0 = 3.5dB$ pour J=10 et J=11. Les résultats de simulation sont présentés à la figure 3.11. Nous n'y représentons que les performances obtenues après convergence pour chacun des codes, soit après 6 itérations pour J<8, 10 itérations pour J=8, 7 itérations pour J=9 et J=11 et 8 itérations pour J=10. Toutes les courbes complètes

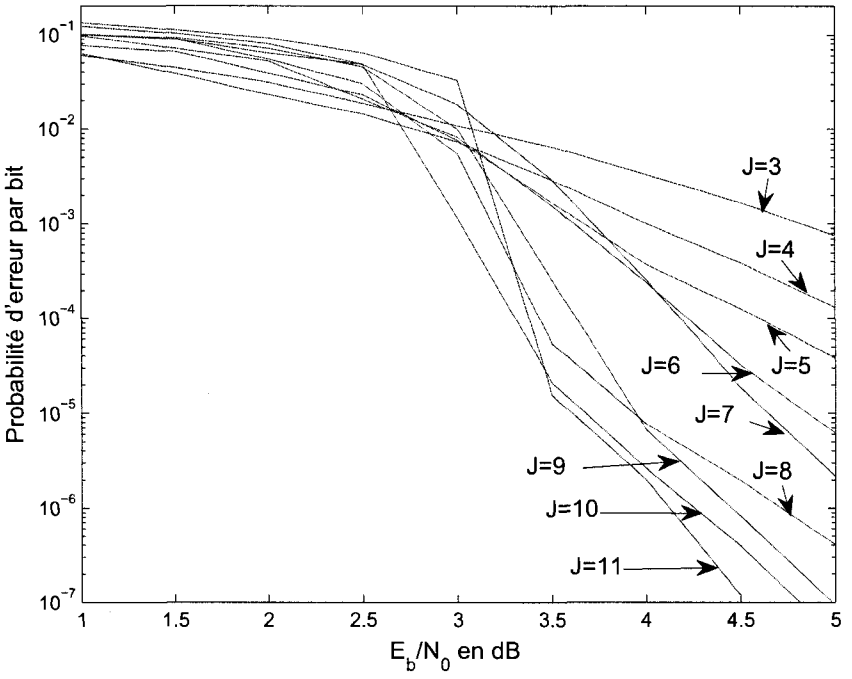

FIG. 3.11 Simulation des performances d'erreur des codes $R - CSO^2C - WS$ pour J variant de 3 à 11

pour chaque code simulé sont présentées dans l'Annexe 3.

Nous pouvons observer que les performances, à haut rapport signal sur bruit, s'améliorent quand J augmente. En revanche les codes $R - CSO^2C - WS$ ne sont clairement pas adaptés pour des valeurs de E_b/N_0 inférieures à 3.5dB. Nous verrons dans le prochain chapitre qu'il est préférable d'utiliser les codes convolutionnels doublement orthogonaux récursifs au sens strict pour travailler en dessous de 3dB.

3.3.5 Comparaison des performances par rapport aux codes connus

Les deux courbes suivantes vont nous permettre d'avoir un élément de comparaison entre les codes $R - CSO^2C - WS$ et les codes doublement orthogonaux non récursifs à des valeurs de E_b/N_0 de 2.5 et 3.5 dB. On y représente les performances d'erreur par bit en fonction de la latence totale, calculée en bits.

Les tableaux 3.12 et 3.13 résument le calcul de la latence pour les codes $R-CSO^2C-WS$ simulés. On y indique, pour chaque code, la latence en bits nécessaire par itération, le nombre total d'itérations pour atteindre la convergence, la latence totale en bit ainsi que la probabilté d'erreur par bit correspondante. Il est important de noter que les coefficients de pondération ont été recalculés pour chacun des codes aux valeurs de E_b/N_0 correspondantes, afin d'avoir les meilleures performances possibles.

Comme nous l'avions observé sur la figure 3.11, la figure 3.12 confirme que les codes $R - CSO^2C - WS$ ne sont pas performants pour des valeurs de E_b/N_0 inférieures à 3dB, et ce, même en calculant les meilleurs duos de coefficients de pondération.

J	lat. / it.	nb it.	latence totale (bits)	P_b
3	3	6	18	1.3×10^{-2}
4	10	8	80	9×10^{-3}
5	24	10	240	7×10^{-3}
6	53	8	424	6×10^{-3}
7	110	10	1110	4×10^{-3}
8	296	10	2960	1×10^{-2}
9	539	10	5390	1.2×10^{-2}
10	905	10	9050	2.2×10^{-2}
11	1831	10	18310	2×10^{-2}

TAB. 3.12 Calcul de la latence des codes $R - CSO^2C - WS$ simulés pour J variant de 3 à 11 avec $E_b/N_0 = 2.5dB$

J	lat. / it.	nb it.	latence totale	P_b
3	3	5	15	5×10^{-3}
4	10	7	70	2×10^{-3}
5	24	10	240	8×10^{-4}
6	53	10	530	2.2×10^{-4}
7	110	8	880	1.1×10^{-4}
8	296	8	2368	4.5×10^{-5}
9	539	8	4312	4×10^{-5}
10	905	8	7240	2.3×10^{-5}
11	1831	8	14648	1.5×10^{-5}

TAB. 3.13 Calcul de la latence des codes $R - CSO^2C - WS$ simulés pour J variant de 3 à 11 avec $E_b/N_0 = 3.5dB$

Nous observons qu'à J fixé, les codes récursifs produisent une latence inférieure aux codes non récursifs non simplifiés. De plus, jusqu'à $J = 7$, la figure 3.13 montre, qu'à un SNR de $3.5dB$, il est plus intéressant d'utiliser les codes récursifs qui offrent de meilleures performances pour une latence moindre. Cependant, les codes non récursifs et simplifiés sont plus avantageux, puisque pour une probabilité d'erreur par bit donnée, ce sont les codes qui engendrent la plus petite latence. De plus, les probabilités d'erreur par bit décroissent plus rapidement, quand on augmente J, pour les codes non récursifs, si bien qu'il n'est plus

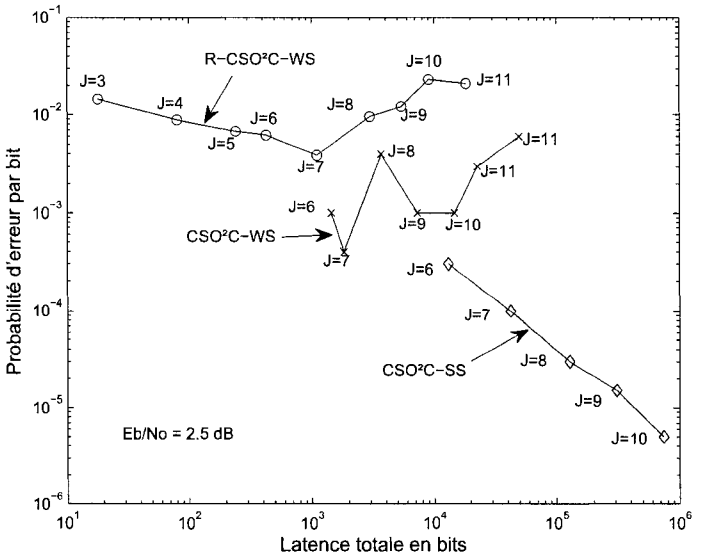

FIG. 3.12 Comparaison des performances d'erreur des codes CSO^2C et $R-CSO^2C-WS$ en fonction de la latence totale après convergence en bits avec $E_b/N_0 = 2.5dB$

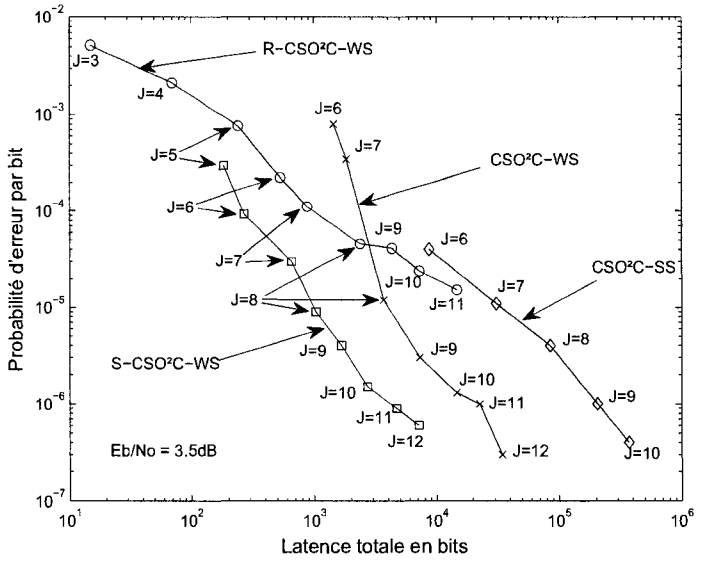

FIG. 3.13 Comparaison des performances d'erreur des codes CSO^2C et $R-CSO^2C-WS$ en fonction de la latence totale après convergence en bits avec $E_b/N_0 = 3.5dB$

intéressant de se servir de la récursivité pour les codes ayant $J > 7$.

3.4 Conclusion

Les codes récursifs au sens large sont de bons codes si on cherche à travailler pour des rapports signal sur bruit élevés. Ils sont, de manière générale, plus courts que les codes doublement orthogonaux au sens large non récursifs non simplifiés, pour des performances comparables. Cependant, les répétitions inévitables des différences à la deuxième itération du processus de décodage itératif nous empêchent d'avoir de bonnes performances d'erreur pour de faibles rapports signal sur bruit. C'est pourquoi nous allons étudier dans le prochain chapitre les codes convolutionnels doublement orthogonaux récursifs au sens strict qui permettront d'avoir une indépendance totale des observables à la deuxème itération. Nous pourrons alors travailler à de faibles valeurs de E_b/N_0, ce qui est le but de cette recherche.

CHAPITRE 4

THÉORIE DES CODES CONVOLUTIONNELS DOUBLEMENT ORTHOGONAUX RÉCURSIFS AU SENS STRICT

4.1 Éléments de théorie

Les codes $R - CSO^2C - WS$, présentés dans le chapitre précédent, n'offrent pas de bonnes performances d'erreur pour des valeurs de E_b/N_0 inférieures à 3dB. Le principal problème est que les observables, à la deuxième itération, ne sont pas complètement indépendantes, à cause des différences doubles égales inévitables, dûes à la permutation des indices. Pour remédier à ceci, l'idée est, comme pour les codes non récursifs, d'utiliser plusieurs registres à décalage qui seront connectés qu'une fois à chaque additionneur modulo-2. Ainsi, nous allons envoyer les bits dans le codeur par blocs de J. Le taux de codage reste de 1/2, puisque que pour les J bits d'information envoyés dans le codeur, on calculera J symboles de parité.

4.1.1 Codeur convolutionnel récursif parallèle

Ce dernier type de codeur sera réservé pour les codes $R - CSO^2C - SS$. En effet, les bits vont entrer dans le codeur par blocs de J bits, et les symboles de parité seront calculés, à l'instant $i = 0, 1, 2, \ldots$, avec des additions modulo-2, non seulement sur les bits d'information précédant les bits d'information courants $u_{n,i}$ avec $n \in \{1, \ldots, J\}$, mais aussi sur les symboles de parité précédant les symboles de parités courants $p_{n,i}$ avec

$n \in \{1, \ldots, J\}$.

Pour représenter ces codeurs, on utilise deux matrices, chacune de taille $J \times J$, et notées α et β. Les éléments de la première matrice sont notés $\alpha_{k,l}$ avec $(k, l) \in \{1, \ldots, J\}$, et donnent, à l'instant $i = 0, 1, 2, \ldots$, le numéro de la case de retard du bit $u_{k,i}$ connectée à l'additionneur modulo-2 correspondant au symbole de parité $p_{l,i}$. Les éléments de la seconde matrice sont notés $\beta_{k,l}$, et donnent le numéro de la case de retard du symbole $p_{k,i}$ connectée à l'additionneur modulo-2 correspondant au symbole de parité $p_{l,i}$. On peut tout de suite noter que les éléments de la matrice β qui valent 0 ne représentent pas de connexion, et ne seront donc pas pris en compte. Un exemple de ce type de codeur est donné aux figures 4.1 et 4.2.

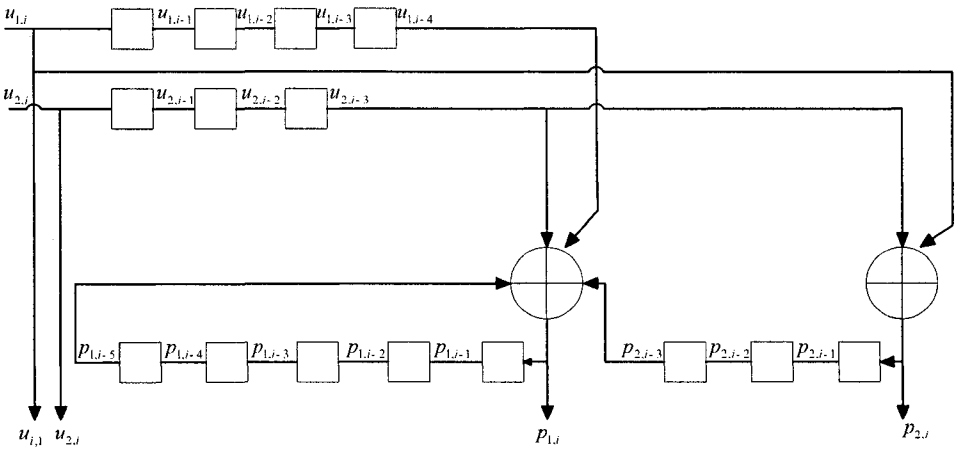

FIG. 4.1 Exemple de codeur convolutionnel récursif en bloc avec J=2, R=1/2 et m=5

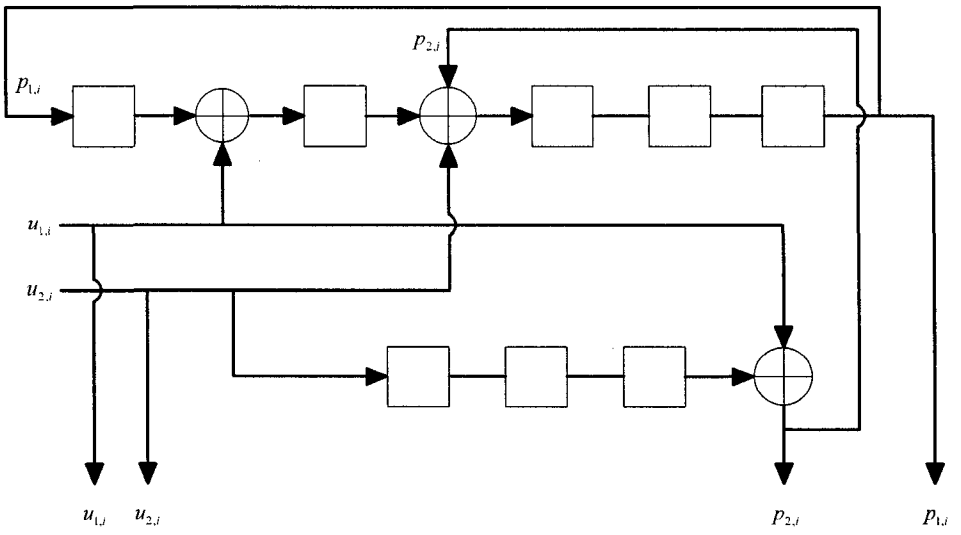

FIG. 4.2 Forme canonique du codeur convolutionnel récursif en bloc de la figure 4.1

Les matrices décrivant ce codeur sont donc :

$$\boldsymbol{\alpha} = \begin{pmatrix} 4 & 0 \\ 3 & 3 \end{pmatrix} \boldsymbol{\beta} = \begin{pmatrix} 5 & 0 \\ 3 & 0 \end{pmatrix}$$

Le span du codeur est alors :

$$m = \max_{\substack{i\in\{1,\ldots,J\} \\ j\in\{1,\ldots,J\}}} \{\alpha_{i,j}, \beta_{i,j}\} = 5$$

Pour ce type de codeurs, les symboles de parité courants $p_{n,i}$, avec $n \in \{1, \ldots, J\}$, à l'instant $i = 0, 1, 2, \ldots$, sont calculés selon la formule :

$$p_{n,i} = \sum_{j=1}^{J} \oplus u_{j,i-\alpha_{j,n}} \oplus \sum_{\substack{k=1 \\ \beta_{k,n}>0}}^{J} \oplus p_{k,i-\beta_{k,n}} \tag{4.1}$$

4.1.2 Décodage

Le décodeur est encore un décodeur à seuil, et il effectue toujours un calcul de syndrome [4]. On obtient ainsi, à l'instant $i = 0, 1, 2, \ldots$, pour $n \in \{1, \ldots, J\}$:

$$s_n(i) = \tilde{p}_{n,i} \oplus p'_{n,i} \tag{4.2}$$

Ainsi, en reprenant les équations (4.1) et (4.2), $s_n(i)$ devient :

$$s_n(i) = \tilde{p}_{n,i} \oplus \sum_{j=1}^{J} \oplus \tilde{u}_{j,i-\alpha_{j,n}} \oplus \sum_{\substack{k=1 \\ \beta_{k,n}>0}}^{J} \oplus \tilde{p}_{k,i-\beta_{k,n}} \tag{4.3}$$

On peut alors prendre l'équation (4.3) aux instants $i + \alpha_{l,n}$ pour $l \in \{1, \ldots, J\}$ afin d'obtenir les systèmes d'équations orthogonales aux bits $\tilde{u}_{l,i}$. Il vient alors, à l'instant i, $i = 0, 1, 2, \ldots$, pour $l \in \{1, \ldots, J\}$:

$$\begin{aligned} s_n(i + \alpha_{l,n}) &= \tilde{u}_{l,i} \oplus \tilde{p}_{n,i+\alpha_{l,n}} \oplus \sum_{\substack{j=1 \\ j \neq l}}^{J} \oplus \tilde{u}_{j,i+(\alpha_{l,n}-\alpha_{j,n})} \oplus \sum_{\substack{k=1 \\ \beta_{k,n}>0}}^{J} \oplus \tilde{p}_{k,i+(\alpha_{l,n}-\beta_{k,n})} \\ &\text{avec } n \in \{1, \ldots, J\} \end{aligned} \tag{4.4}$$

De la même manière, pour le décodage des symboles $\tilde{p}_{l,i}$ avec $l \in \{1, \ldots, J\}$, on observe l'équation (4.3) aux instants $i + \beta_{l,n}$. Il vient alors à l'instant $i = 0, 1, 2, \ldots$, pour chaque

$l \in \{1, \ldots, J\}$ tel que $\beta_{l,n} > 0$:

$$
\begin{aligned}
s_l(i) &= \tilde{p}_{l,i} \oplus \sum_{j=1}^{J} \oplus \tilde{u}_{j,i-\alpha_{j,l}} \oplus \sum_{\substack{k=1 \\ \beta_{k,l}>0}}^{J} \oplus \tilde{p}_{k,i-\beta_{k,l}} \\
s_n(i+\beta_{l,n}) &= \tilde{p}_{l,i} \oplus \tilde{p}_{n,i+\beta_{l,n}} \oplus \sum_{j=1}^{J} \oplus \tilde{u}_{j,i+(\beta_{l,n}-\alpha_{j,n})} \oplus \sum_{\substack{k=1 \\ \beta_{k,n}>0 \\ k\neq l}}^{J} \oplus \tilde{p}_{k,i+(\beta_{l,n}-\beta_{k,n})} \quad (4.5) \\
&\quad \text{avec } n \in \{1, \ldots, J\}
\end{aligned}
$$

On déduit des équations précédentes qu'à chaque instant i, $i = 0, 1, 2, \ldots$, les bits $\tilde{u}_{l,i}$, avec $l \in \{1, \ldots, J\}$, sont décodés grâce à J équations, notées $\Psi_{u,l,m}$, avec $m \in \{1, \ldots, J\}$. En revanche, pour les symboles $\tilde{p}_{l,i}$, nous aurons un nombre variable d'équations pour effectuer leur décodage. Ainsi, si on note N_l le nombre d'éléments non nuls dans la matrice $\boldsymbol{\beta}$ à la ligne $l \in \{1, \ldots, J\}$, on aura, à chaque instant i, $N_l + 1$ équations orthogonales, notées $\Psi_{p,l,0}$ et $\Psi_{p,l,n}$ avec $n \in \{1, \ldots, J\}$, pour décoder les symboles $p_{l,i}$, avec $l \in \{1, \ldots, J\}$. Notons que seules les équations $\Psi_{p,l,n}$ pour $\beta_{l,n} > 0$ sont calculées.

De plus, nous obtenons, à partir des équations (4.4) et (4.5), les équations utilisées à la première itération par le décodeur pour décoder les symboles $\tilde{u}_{l,i}$ et $\tilde{p}_{l,i}$ pour $l \in \{1, \ldots, J\}$,

à l'instant $i = 0, 1, 2, \ldots$. Elles s'écrivent comme suit [4] :

$$\begin{aligned}
\lambda_{u,l}^{(1)}(i) &= y_{l,i}^{u} + \sum_{m=1}^{J} \left(y_{m,i+\alpha_{l,m}}^{p} \diamond \sum_{\substack{j=1 \\ j \neq l}}^{J} \diamond y_{j,i+(\alpha_{l,m}-\alpha_{j,m})}^{u} \diamond \sum_{\substack{k=1 \\ \beta_{k,m}>0}}^{J} \diamond y_{k,i+(\alpha_{l,m}-\beta_{k,m})}^{p} \right) \\
&= y_{l,i}^{u} + \sum_{m=1}^{J} \Psi_{u,l,m}^{(1)} \qquad (4.6) \\
\lambda_{p,l}^{(1)}(i) &= y_{l,i}^{p} + \left(\sum_{j=1}^{J} \diamond y_{j,i-\alpha_{j,l}}^{u} \diamond \sum_{\substack{k=1 \\ \beta_{k,l}>0}}^{J} \diamond y_{k,i-\beta_{k,l}}^{p} \right) + \\
&\quad \sum_{\substack{n=1 \\ \beta_{l,n}>0}}^{J} \left(y_{n,i+\beta_{l,n}}^{p} \diamond \sum_{j=1}^{J} \diamond y_{j,i+(\beta_{l,n}-\alpha_{j,n})}^{u} \diamond \sum_{\substack{k=1 \\ k \neq l \\ \beta_{k,n}>0}}^{J} \diamond y_{k,i+(\beta_{l,n}-\beta_{k,n})}^{p} \right) \\
&= y_{l,i}^{p} + \Psi_{p,l,0}^{(1)} + \sum_{\substack{n=1 \\ \beta_{l,n}>0}}^{J} \Psi_{p,l,n}^{(1)} \qquad (4.7)
\end{aligned}$$

Cependant, l'algorithme de décodage utilisé est semblable à celui servant au décodage des codes LDPC défini dans [12] et amélioré dans [9]. La conséquence de l'utilisation de cet algorithme est qu'au lieu d'effectuer directement l' approximation du LRV représenté par $\lambda_{u,l}^{(1)}(i)$ qui contient J équations, l'algorithme va calculer J termes $\lambda_{u,l,h}^{(1)}(i)$, avec $h \in \{1, \ldots, J\}$, en utilisant à chaque fois $J-1$ équations, soit toutes sauf $\Psi_{u,l,h}^{(1)}$. Pour $\lambda_{p,l}^{(1)}(i)$, il calcule $N_l + 1$ termes $\lambda_{p,l,v}^{(1)}(i)$, avec $v \in \{0, \ldots, J\}$, faisant intervenir N_l équations, soit toutes sauf $\Psi_{p,l,v}^{(1)}$. On obtient donc pour le décodage de la première itération, pour $l \in \{1, \ldots, J\}$ et $h \in \{1, \ldots, J\}$ [4] :

$$\begin{aligned}\lambda^{(1)}_{u,l,h}(i) &= y^u_{l,i} + \sum_{\substack{m=1\\m\neq h}}^{J}\Big(y^p_{m,i+\alpha_{l,m}} \diamond \sum_{\substack{j=1\\j\neq l}}^{J} \diamond y^u_{j,i+(\alpha_{l,m}-\alpha_{j,m})} \diamond \sum_{\substack{k=1\\\beta_{k,m}>0}}^{J} \diamond y^p_{k,i+(\alpha_{l,m}-\beta_{k,m})}\Big)\\ &= y^u_{l,i} + \sum_{\substack{m=1\\m\neq h}}^{J} \Psi^{(1)}_{u,l,m} \qquad (4.8)\end{aligned}$$

De la même manière, pour $l \in \{1,\ldots,J\}$ et $v \in \{1,\ldots,J\}$ [4] :

$$\begin{aligned}\lambda^{(1)}_{p,l,v}(i) &= y^p_{l,i} + \Big(\sum_{j=1}^{J} \diamond y^u_{j,i-\alpha_{j,l}} \diamond \sum_{\substack{k=1\\\beta_{k,l}>0}}^{J} \diamond y^p_{k,i-\beta_{k,l}}\Big) +\\ &\qquad \sum_{\substack{n=1\\n\neq v,\beta_{l,n}>0}}^{J}\Big(y^p_{n,i+\beta_{l,n}} \diamond \sum_{j=1}^{J} \diamond y^u_{j,i+(\beta_{l,n}-\alpha_{j,n})} \diamond \sum_{\substack{k=1\\k\neq l\\\beta_{k,n}>0}}^{J} \diamond y^p_{k,i+(\beta_{l,n}-\beta_{k,n})}\Big)\\ &= y^p_{l,i} + \Psi^{(1)}_{p,l,0} + \sum_{\substack{n=1\\n\neq v\,,\,\beta_{l,n}>0}}^{J} \Psi^{(1)}_{p,l,n} \qquad (4.9)\\ \lambda^{(1)}_{p,l,0}(i) &= y^p_{l,i} + \sum_{\substack{n=1\\\beta_{l,n}>0}}^{J}\Big(y^p_{n,i+\beta_{l,n}} \diamond \sum_{j=1}^{J} \diamond y^u_{j,i+(\beta_{l,n}-\alpha_{j,n})} \diamond \sum_{\substack{k=1\\k\neq l\\\beta_{k,n}>0}}^{J} \diamond y^p_{k,i+(\beta_{l,n}-\beta_{k,n})}\Big)\\ &= y^p_{l,i} + \sum_{\substack{n=1\\\beta_{l,n}>0}}^{J} \Psi^{(1)}_{p,l,n} \qquad (4.10)\end{aligned}$$

Nous avons ainsi obtenu l'algorithme de décodage pour la première itération. Afin de généraliser l'algorithme pour une itération quelconque $\mu > 1$, nous transformons les équations (4.8) à (4.10) de la façon suivante, pour $l \in \{1,\ldots,J\}$ et $h \in \{1,\ldots,J\}$ [4] :

$$\begin{aligned}\lambda^{(\mu)}_{u,l,h}(i) &= y^u_{l,i} + \sum_{\substack{r=1\\ r\neq h}}^{J}\Big(\lambda^{(\mu-1)}_{p,r,0}(i+\alpha_{l,r}) \diamond \sum_{\substack{s=1\\ s\neq l}}^{J} \diamond \lambda^{(\mu-1)}_{u,s,r}(i+(\alpha_{l,r}-\alpha_{s,r})) \diamond \cdots \\ &\qquad \cdots \sum_{\substack{t=1\\ \beta_{t,r}>0}}^{J} \diamond \lambda^{(\mu-1)}_{p,t,r}(i+(\alpha_{l,r}-\beta_{t,r}))\Big) \\ &= y^u_{l,i} + \sum_{\substack{r=1\\ r\neq h}}^{J} \Psi^{(\mu)}_{u,l,r} \end{aligned} \tag{4.11}$$

De la même manière, pour $l \in \{1,\ldots,J\}$ et $v \in \{1,\ldots,J\}$ [4] :

$$\begin{aligned}\lambda^{(\mu)}_{p,l,v}(i) &= y^p_{l,i} + \Big(\sum_{s=1}^{J} \diamond \lambda^{(\mu-1)}_{u,s,l}(i-\alpha_{s,l}) \diamond \sum_{\substack{t=1\\ \beta_{t,l}>0}}^{J} \diamond \lambda^{(\mu-1)}_{p,t,l}(i-\beta_{t,l})\Big) + \cdots \\ &\qquad \cdots \sum_{\substack{r=1\\ r\neq v,\beta_{l,r}>0}}^{J} \Big(\lambda^{(\mu-1)}_{p,r,0}(i+\beta_{l,r}) \diamond \sum_{s=1}^{J} \diamond \lambda^{(\mu-1)}_{u,s,r}(i+(\beta_{l,r}-\alpha_{s,r})) \diamond \cdots \\ &\qquad \cdots \sum_{\substack{t=1\\ t\neq l\\ \beta_{t,r}>0}}^{J} \diamond \lambda^{(\mu-1)}_{p,t,r}(i+(\beta_{l,r}-\beta_{t,r}))\Big) \\ &= y^p_{l,i} + \Psi^{(\mu)}_{p,l,0} + \sum_{\substack{r=1\\ r\neq v,\beta_{l,r}>0}}^{J} \Psi^{(\mu)}_{p,l,r} \end{aligned} \tag{4.12}$$

$$\begin{aligned}\lambda^{(\mu)}_{p,l,0}(i) &= y^p_{l,i} + \sum_{\substack{r=1\\ \beta_{l,r}>0}}^{J}\Big(\lambda^{(\mu-1)}_{p,r,0}(i+\beta_{l,r}) \diamond \sum_{s=1}^{J} \diamond \lambda^{(\mu-1)}_{u,s,r}(i+(\beta_{l,r}-\alpha_{s,r})) \diamond \cdots \\ &\qquad \cdots \sum_{\substack{t=1\\ t\neq l\\ \beta_{t,r}>0}}^{J} \diamond \lambda^{(\mu-1)}_{p,t,r}(i+(\beta_{l,r}-\beta_{t,r}))\Big) \\ &= y^p_{l,i} + \sum_{\substack{r=1\\ \beta_{l,r}>0}}^{J} \Psi^{(\mu)}_{p,l,r} \end{aligned} \tag{4.13}$$

Enfin, pour la dernière itération $\mu = M$, la décision finale sur l'estimation $\lambda_{u,l}^{(\mu)}(i)$ du LRV de $\tilde{u}_{l,i}$ se fait en se servant de toutes les équations disponibles :

$$\lambda_{u,l}^{(M)}(i) = y_{l,i}^{u} + \sum_{r=1}^{J} \Psi_{u,l,r}^{(M)} \tag{4.14}$$

L'algorithme de décodage est donc entièrement décrit par les équations (4.8) à (4.14).

Nous allons maintenant définir, de manière spécifique, les codes $R - CSO^2C - SS$ en faisant ressortir tous les critères qu'ils doivent respecter.

4.1.3 Détermination des conditions à respecter par les codes $R - CSO^2C - SS$

De la même manière que pour les codes récursifs au sens large, nous devons développer la deuxième itération de l'algorithme de décodage afin de déterminer l'ensemble des conditions que les codes $R - CSO^2C - SS$ doivent satisfaire. En effet, à chaque approximation $\lambda_{u,l,h}^{(2)}(i)$ et $\lambda_{p,l,v}^{(2)}(i)$ effectuées à la deuxième itération, pour $l, h, v \in \{1, \ldots, J\}$, tous les termes utilisés doivent être différents. Or, nous avons, à partir des équations (4.11) à (4.13) décrivant l'algorithme de décodage :

$$\lambda^{(2)}_{u,l,h}(i) = y^u_{l,i} + \sum_{\substack{r=1\\ r\neq h}}^{J} \Big(\lambda^{(1)}_{p,r,0}(i+\alpha_{l,r}) \diamond \sum_{\substack{s=1\\ s\neq l}}^{J} \diamond \lambda^{(1)}_{u,s,r}(i+(\alpha_{l,r}-\alpha_{s,r})) \diamond \cdots$$

$$\cdots \sum_{\substack{t=1\\ \beta_{t.r}>0}}^{J} \diamond \lambda^{(1)}_{p,t,r}(i+(\alpha_{l,r}-\beta_{t,r})) \Big)$$

$$\lambda^{(2)}_{p,l,v}(i) = y^p_{l,i} + \Big(\sum_{s=1}^{J} \diamond \lambda^{(1)}_{u,s,l}(i-\alpha_{s,l}) \diamond \sum_{\substack{t=1\\ \beta_{t.l}>0}}^{J} \diamond \lambda^{(1)}_{p,t,l}(i-\beta_{t,l}) \Big) + \cdots$$

$$\cdots \sum_{\substack{r=1\\ r\neq v, \beta_{l.r}>0}}^{J} \Big(\lambda^{(1)}_{p,r,0}(i+\beta_{l,r}) \diamond \sum_{s=1}^{J} \diamond \lambda^{(1)}_{u,s,r}(i+(\beta_{l,r}-\alpha_{s,r})) \diamond \cdots$$

$$\cdots \sum_{\substack{t=1\\ t\neq l\\ \beta_{t.r}>0}}^{J} \diamond \lambda^{(1)}_{p,t,r}(i+(\beta_{l,r}-\beta_{t,r})) \Big)$$

$$\lambda^{(2)}_{p,l,0}(i) = y^p_{l,i} + \sum_{\substack{r=1\\ \beta_{l.r}>0}}^{J} \Big(\lambda^{(1)}_{p,r,0}(i+\beta_{l,r}) \diamond \sum_{s=1}^{J} \diamond \lambda^{(1)}_{u,s,r}(i+(\beta_{l,r}-\alpha_{s,r})) \diamond \cdots$$

$$\cdots \sum_{\substack{t=1\\ t\neq l\\ \beta_{t,r}>0}}^{J} \diamond \lambda^{(1)}_{p,t,r}(i+(\beta_{l,r}-\beta_{t,r})) \Big)$$

Ainsi, en réinjectant les équations (4.8) à (4.10), on obtient à la deuxième itération :

$$
\begin{aligned}
\lambda^{(2)}_{u,l,h}(i) &= y^u_{l,i} + \sum_{\substack{r=1\\ r\neq h}}^{J} \Bigg(\Bigg(y^p_{r,i+\alpha_{l,r}} + \sum_{\substack{n=1\\ \beta_{r,n}>0}}^{J} \Big(y^p_{n,i+\alpha_{l,r}+\beta_{r,n}} \diamond \sum_{j=1}^{J} \diamond y^u_{j,i+\alpha_{l,r}+(\beta_{r,n}-\alpha_{j,n})} \diamond \\
&\quad \sum_{\substack{k=1\\ k\neq r\\ \beta_{k,n}>0}}^{J} \diamond y^p_{k,i+\alpha_{l,r}+(\beta_{r,n}-\beta_{k,n})} \Big) \Bigg) \diamond \sum_{\substack{s=1\\ s\neq l}}^{J} \diamond \Bigg(y^u_{s,i+(\alpha_{l,r}-\alpha_{s,r})} + \sum_{\substack{m=1\\ m\neq r}}^{J} \Big(y^p_{m,i+(\alpha_{l,r}-\alpha_{s,r})+\alpha_{s,m}} \diamond \\
&\quad \sum_{\substack{j=1\\ j\neq s}}^{J} \diamond y^u_{j,i+(\alpha_{l,r}-\alpha_{s,r})+(\alpha_{s,m}-\alpha_{j,m})} \diamond \sum_{\substack{k=1\\ \beta_{k,m}>0}}^{J} \diamond y^p_{k,i+(\alpha_{l,r}-\alpha_{s,r})+(\alpha_{s,m}-\beta_{k,m})} \Big) \Bigg) \diamond \\
&\quad \sum_{\substack{t=1\\ \beta_{t,r}>0}}^{J} \diamond \Bigg(y^p_{t,i+(\alpha_{l,r}-\beta_{t,r})} + \Big(\sum_{j=1}^{J} \diamond y^u_{j,i+(\alpha_{l,r}-\beta_{t,r})-\alpha_{j,t}} \diamond \sum_{\substack{k=1\\ \beta_{k,t}>0}}^{J} \diamond y^p_{k,i+(\alpha_{l,r}-\beta_{t,r})-\beta_{k,t}} \Big) + \\
&\quad \sum_{\substack{n=1\\ n\neq r,\beta_{t,n}>0}}^{J} \Big(y^p_{n,i+(\alpha_{l,r}-\beta_{t,r})+\beta_{t,n}} \diamond \sum_{j=1}^{J} \diamond y^u_{j,i+(\alpha_{l,r}-\beta_{t,r})+(\beta_{t,n}-\alpha_{j,n})} \diamond \\
&\quad \sum_{\substack{k=1\\ k\neq t\\ \beta_{k,n}>0}}^{J} \diamond y^p_{k,i+(\alpha_{l,r}-\beta_{t,r})+(\beta_{t,n}-\beta_{k,n})} \Big) \Big) \Bigg) \qquad (4.15) \\
&= y^u_{l,i} + \sum_{\substack{r=1\\ r\neq h}}^{J} \Psi^{(2)}_{u,l,r}
\end{aligned}
$$

Cependant, l'équation (4.15), pour l fixé, montre qu'il faut, pour chaque $h \in \{1, \ldots, J\}$ que tous les termes engendrés par les équations $\Psi^{(2)}_{u,l,r}$, avec $r \in \{1, \ldots, J\}$, sauf $\Psi^{(2)}_{u,l,h}$, soient différents. Finalement, cela revient à dire, si $J > 2$, que tous les termes introduits par les équations $\Psi^{(2)}_{u,l,r}$ doivent être différents. Nous déduisons donc qu'il n'est plus nécessaire de garder la restriction $r \neq h$ pour la recherche de conditions.

Nous avons également pour la deuxième itération, pour $v \in \{1, \ldots, J\}$:

$$
\begin{aligned}
\lambda^{(2)}_{p,l,v}(i) &= y^p_{l,i} + \Bigg(\sum_{s=1}^{J} \diamond \Bigg(y^u_{s,i-\alpha_{s,l}} + \sum_{\substack{m=1 \\ m\neq l}}^{J} \Big(y^p_{m,i-\alpha_{s,l}+\alpha_{s,m}} \diamond \sum_{\substack{j=1 \\ j\neq s}}^{J} \diamond y^u_{j,i-\alpha_{s,l}+(\alpha_{s,m}-\alpha_{j,m})} \diamond \\
&\quad \sum_{\substack{k=1 \\ \beta_{k,m}>0}}^{J} \diamond y^p_{k,i-\alpha_{s,l}+(\alpha_{s,m}-\beta_{k,m})} \Big) \Bigg) \diamond \sum_{\substack{t=1 \\ \beta_{t,l}>0}}^{J} \diamond \Bigg(y^p_{t,i-\beta_{t,l}} + \Big(\sum_{j=1}^{J} \diamond y^u_{j,i-\beta_{t,l}-\alpha_{j,t}} \diamond \\
&\quad \sum_{\substack{k=1 \\ \beta_{k,t}>0}}^{J} \diamond y^p_{k,i-\beta_{t,l}-\beta_{k,t}} \Big) + \sum_{\substack{n=1 \\ n\neq l, \beta_{t,n}>0}}^{J} \Big(y^p_{n,i-\beta_{t,l}+\beta_{t,n}} \diamond \sum_{j=1}^{J} \diamond y^u_{j,i-\beta_{t,l}+(\beta_{t,n}-\alpha_{j,n})} \diamond \\
&\quad \sum_{\substack{k=1 \\ k\neq t \\ \beta_{k,n}>0}}^{J} \diamond y^p_{k,i-\beta_{t,l}+(\beta_{t,n}-\beta_{k,n})} \Big) \Bigg) \Bigg) + \sum_{\substack{r=1 \\ r\neq v, \beta_{l,r}>0}}^{J} \Bigg(\Big(y^p_{r,i+\beta_{l,r}} + \sum_{\substack{n=1 \\ \beta_{r,n}>0}}^{J} \Big(y^p_{n,i+\beta_{l,r}+\beta_{r,n}} \diamond \\
&\quad \sum_{j=1}^{J} \diamond y^u_{j,i+\beta_{l,r}+(\beta_{r,n}-\alpha_{j,n})} \diamond \sum_{\substack{k=1 \\ k\neq r \\ \beta_{k,n}>0}}^{J} \diamond y^p_{k,i+\beta_{l,r}+(\beta_{r,n}-\beta_{k,n})} \Big) \Big) \diamond \sum_{s=1}^{J} \diamond \Big(y^u_{s,i+(\beta_{l,r}-\alpha_{s,r})} + \\
&\quad \sum_{\substack{m=1 \\ m\neq r}}^{J} \Big(y^p_{m,i+(\beta_{l,r}-\alpha_{s,r})+\alpha_{s,m}} \diamond \sum_{\substack{j=1 \\ j\neq s}}^{J} \diamond y^u_{j,i+(\beta_{l,r}-\alpha_{s,r})+(\alpha_{s,m}-\alpha_{j,m})} \diamond \\
&\quad \sum_{\substack{k=1 \\ \beta_{k,m}>0}}^{J} \diamond y^p_{k,i+(\beta_{l,r}-\alpha_{s,r})+(\alpha_{s,m}-\beta_{k,m})} \Big) \Big) \diamond \sum_{\substack{t=1 \\ t\neq l \\ \beta_{t,r}>0}}^{J} \diamond \Big(y^p_{t,i+(\beta_{l,r}-\beta_{t,r})} + \\
&\quad \Big(\sum_{j=1}^{J} \diamond y^u_{j,i+(\beta_{l,r}-\beta_{t,r})-\alpha_{j,t}} \diamond \sum_{\substack{k=1 \\ \beta_{k,t}>0}}^{J} \diamond y^p_{k,i+(\beta_{l,r}-\beta_{t,r})-\beta_{k,t}} \Big) + \sum_{\substack{n=1 \\ n\neq r, \beta_{t,n}>0}}^{J} \Big(y^p_{n,i+(\beta_{l,r}-\beta_{t,r})+\beta_{t,n}} \diamond \\
&\quad \sum_{j=1}^{J} \diamond y^u_{j,i+(\beta_{l,r}-\beta_{t,r})+(\beta_{t,n}-\alpha_{j,n})} \diamond \sum_{\substack{k=1 \\ k\neq t \\ \beta_{k,n}>0}}^{J} \diamond y^p_{k,i+(\beta_{l,r}-\beta_{t,r})+(\beta_{t,n}-\beta_{k,n})} \Big) \Big) \Bigg) \qquad (4.16) \\
&= y^p_{l,i} + \Psi^{(\mu)}_{p,l,0} + \sum_{\substack{r=1 \\ r\neq v, \beta_{l,r}>0}}^{J} \Psi^{(\mu)}_{p,l,r} \\
\lambda^{(2)}_{p,l,0}(i) &= y^p_{l,i} + \sum_{\substack{r=1 \\ \beta_{l,r}>0}}^{J} \Psi^{(\mu)}_{p,l,r} \qquad \text{pour } v = 0
\end{aligned}
$$

Pour des raisons identiques à celles qui nous ont amenées à ne plus considérer la condition $r \neq h$ dans l'équation (4.15), il n'est plus nécessaire de restreindre les calculs pour $r \neq v$. Nous obtenons alors deux équations, dont l'ensemble des termes doivent être différents pour que le code soit $R - CSO^2C - SS$. Elles sont définies aux équations (4.17) et (4.18) :

$$
\begin{aligned}
\lambda^{(2)}_{u,l}(i) \;=\; & y^u_{l,i} + \sum_{r=1}^{J} \Bigg(\Bigg(y^p_{r,i+\alpha_{l,r}} + \sum_{\substack{n=1 \\ \beta_{r,n}>0}}^{J} \Big(y^p_{n,i+\alpha_{l,r}+\beta_{r,n}} \diamond \sum_{j=1}^{J} \diamond y^u_{j,i+\alpha_{l,r}+(\beta_{r,n}-\alpha_{j,n})} \diamond \\
& \sum_{\substack{k=1 \\ k\neq r \\ \beta_{k,n}>0}}^{J} \diamond y^p_{k,i+\alpha_{l,r}+(\beta_{r,n}-\beta_{k,n})} \Big) \Bigg) \diamond \sum_{\substack{s=1 \\ s\neq l}}^{J} \diamond \Bigg(y^u_{s,i+(\alpha_{l,r}-\alpha_{s,r})} + \sum_{\substack{m=1 \\ m\neq r}}^{J} \Big(y^p_{m,i+(\alpha_{l,r}-\alpha_{s,r})+\alpha_{s,m}} \diamond \\
& \sum_{\substack{j=1 \\ j\neq s}}^{J} \diamond y^u_{j,i+(\alpha_{l,r}-\alpha_{s,r})+(\alpha_{s,m}-\alpha_{j,m})} \diamond \sum_{\substack{k=1 \\ \beta_{k,m}>0}}^{J} \diamond y^p_{k,i+(\alpha_{l,r}-\alpha_{s,r})+(\alpha_{s,m}-\beta_{k,m})} \Big) \Bigg) \diamond \\
& \sum_{\substack{t=1 \\ \beta_{t,r}>0}}^{J} \diamond \Bigg(y^p_{t,i+(\alpha_{l,r}-\beta_{t,r})} + \Big(\sum_{j=1}^{J} \diamond y^u_{j,i+(\alpha_{l,r}-\beta_{t,r})-\alpha_{j,t}} \diamond \sum_{\substack{k=1 \\ \beta_{k,t}>0}}^{J} \diamond y^p_{k,i+(\alpha_{l,r}-\beta_{t,r})-\beta_{k,t}} \Big) + \\
& \sum_{\substack{n=1 \\ n\neq r, \beta_{t,n}>0}}^{J} \Big(y^p_{n,i+(\alpha_{l,r}-\beta_{t,r})+\beta_{t,n}} \diamond \sum_{j=1}^{J} \diamond y^u_{j,i+(\alpha_{l,r}-\beta_{t,r})+(\beta_{t,n}-\alpha_{j,n})} \diamond \\
& \sum_{\substack{k=1 \\ k\neq t \\ \beta_{k,n}>0}}^{J} \diamond y^p_{k,i+(\alpha_{l,r}-\beta_{t,r})+(\beta_{t,n}-\beta_{k,n})} \Big) \Bigg) \Bigg)
\end{aligned}
\tag{4.17}
$$

$$
\begin{aligned}
\lambda^{(2)}_{p,l}(i) \;=\; & y^p_{l,i} + \Bigg(\sum_{s=1}^{J} \diamond \Bigg(y^u_{s,i-\alpha_{s,l}} + \sum_{\substack{m=1 \\ m\neq l}}^{J} \Big(y^p_{m,i-\alpha_{s,l}+\alpha_{s,m}} \diamond \sum_{\substack{j=1 \\ j\neq s}}^{J} \diamond y^u_{j,i-\alpha_{s,l}+(\alpha_{s,m}-\alpha_{j,m})} \diamond \\
& \sum_{\substack{k=1 \\ \beta_{k,m}>0}}^{J} \diamond y^p_{k,i-\alpha_{s,l}+(\alpha_{s,m}-\beta_{k,m})} \Big) \Bigg) \diamond \sum_{\substack{t=1 \\ \beta_{t,l}>0}}^{J} \diamond \Bigg(y^p_{t,i-\beta_{t,l}} + \Big(\sum_{j=1}^{J} \diamond y^u_{j,i-\beta_{t,l}-\alpha_{j,t}} \diamond \\
& \sum_{\substack{k=1 \\ \beta_{k,t}>0}}^{J} \diamond y^p_{k,i-\beta_{t,l}-\beta_{k,t}} \Big) + \sum_{\substack{n=1 \\ n\neq l, \beta_{t,n}>0}}^{J} \Big(y^p_{n,i-\beta_{t,l}+\beta_{t,n}} \diamond \sum_{j=1}^{J} \diamond y^u_{j,i-\beta_{t,l}+(\beta_{t,n}-\alpha_{j,n})} \diamond \\
& \sum_{\substack{k=1 \\ k\neq t \\ \beta_{k,n}>0}}^{J} \diamond y^p_{k,i-\beta_{t,l}+(\beta_{t,n}-\beta_{k,n})} \Big) \Big) \Bigg) + \sum_{\substack{r=1 \\ \beta_{l,r}>0}}^{J} \Bigg(\Big(y^p_{r,i+\beta_{l,r}} + \sum_{\substack{n=1 \\ \beta_{r,n}>0}}^{J} \Big(y^p_{n,i+\beta_{l,r}+\beta_{r,n}} \diamond \\
& \sum_{j=1}^{J} \diamond y^u_{j,i+\beta_{l,r}+(\beta_{r,n}-\alpha_{j,n})} \diamond \sum_{\substack{k=1 \\ k\neq r \\ \beta_{k,n}>0}}^{J} \diamond y^p_{k,i+\beta_{l,r}+(\beta_{r,n}-\beta_{k,n})} \Big) \Big) \diamond \sum_{s=1}^{J} \diamond \Big(y^u_{s,i+(\beta_{l,r}-\alpha_{s,r})} + \\
& \sum_{\substack{m=1 \\ m\neq r}}^{J} \Big(y^p_{m,i+(\beta_{l,r}-\alpha_{s,r})+\alpha_{s,m}} \diamond \sum_{\substack{j=1 \\ j\neq s}}^{J} \diamond y^u_{j,i+(\beta_{l,r}-\alpha_{s,r})+(\alpha_{s,m}-\alpha_{j,m})} \diamond \\
& \sum_{\substack{k=1 \\ \beta_{k,m}>0}}^{J} \diamond y^p_{k,i+(\beta_{l,r}-\alpha_{s,r})+(\alpha_{s,m}-\beta_{k,m})} \Big) \Big) \diamond \sum_{\substack{t=1 \\ t\neq l \\ \beta_{t,r}>0}}^{J} \diamond \Big(y^p_{t,i+(\beta_{l,r}-\beta_{t,r})} + \\
& \Big(\sum_{j=1}^{J} \diamond y^u_{j,i+(\beta_{l,r}-\beta_{t,r})-\alpha_{j,t}} \diamond \sum_{\substack{k=1 \\ \beta_{k,t}>0}}^{J} \diamond y^p_{k,i+(\beta_{l,r}-\beta_{t,r})-\beta_{k,t}} \Big) + \sum_{\substack{n=1 \\ n\neq r, \beta_{t,n}>0}}^{J} \Big(y^p_{n,i+(\beta_{l,r}-\beta_{t,r})+\beta_{t,n}} \diamond \\
& \sum_{j=1}^{J} \diamond y^u_{j,i+(\beta_{l,r}-\beta_{t,r})+(\beta_{t,n}-\alpha_{j,n})} \diamond \sum_{\substack{k=1 \\ k\neq t \\ \beta_{k,n}>0}}^{J} \diamond y^p_{k,i+(\beta_{l,r}-\beta_{t,r})+(\beta_{t,n}-\beta_{k,n})} \Big) \Big) \Bigg)
\end{aligned}
\tag{4.18}
$$

Ainsi, nous voulons que tous les indices des termes en y^u soient différents, tout comme les indices des termes en y^p, dans chacune des équations (4.17) et (4.18). Il y aura donc

quatre ensembles de conditions, qui sont présentés dans les tableaux 4.1 à 4.4. Il faut, pour tous l et j fixés dans $\{1,\ldots,J\}$, pour toute combinaison des entiers $r,n,t \in \{1,\ldots,J\}$ sous les restrictions indiquées, que tous les termes inscrits dans chaque tableau soient différents entre eux et avec les autres termes du tableau, en supposant que tous les $\beta_{l,j}$, avec $(l,j) \in \{1,\ldots,J\}$ soient positifs. L'ordre des conditions suit l'ordre d'apparition des termes dans les équations (4.17) et (4.18).

Conditions	terme	restriction
1	$\alpha_{l,r} + \beta_{r,n} - \alpha_{j,n}$	
2	$\alpha_{l,r} - \alpha_{j,r}$	$j \neq l$
3	$\alpha_{l,r} - \alpha_{t,r} + \alpha_{t,n} - \alpha_{j,n}$	$n \neq r\ t \neq l\ t \neq j$
4	$\alpha_{l,r} - \beta_{n,r} - \alpha_{j,n}$	
5	$\alpha_{l,r} - \beta_{t,r} + \beta_{t,n} - \alpha_{j,n}$	$n \neq r$

TAB. 4.1 Indices des termes en y^u dans l'équation (4.17)

Conditions	terme	restriction
1	$\alpha_{l,j}$	
2	$\alpha_{l,r} + \beta_{r,j}$	
3	$\alpha_{l,r} + \beta_{r,n} - \beta_{j,n}$	$r \neq j$
4	$\alpha_{l,r} - \alpha_{n,r} + \alpha_{n,j}$	$n \neq l\ r \neq j$
5	$\alpha_{l,r} - \alpha_{t,r} + \alpha_{t,n} - \beta_{j,n}$	$n \neq r\ t \neq l$
6	$\alpha_{l,r} - \beta_{j,r}$	
7	$\alpha_{l,r} - \beta_{n,r} - \beta_{j,n}$	
8	$\alpha_{l,r} - \beta_{n,r} + \beta_{n,j}$	$r \neq j$
9	$\alpha_{l,r} - \beta_{t,r} + \beta_{t,n} - \beta_{j,n}$	$n \neq r\ t \neq j$

TAB. 4.2 Indices des termes en y^p dans l'équation (4.17)

Conditions	terme	restriction
1	$-\alpha_{j,l}$	
2	$-\alpha_{n,l} + \alpha_{n,r} - \alpha_{j,r}$	$n \neq j \; r \neq l$
3	$-\beta_{r,l} - \alpha_{j,r}$	
4	$-\beta_{n,l} + \beta_{n,r} - \alpha_{j,r}$	$r \neq l$
5	$\beta_{l,n} + \beta_{n,r} - \alpha_{j,r}$	
6	$\beta_{l,r} - \alpha_{j,r}$	
7	$\beta_{l,n} - \alpha_{t,n} + \alpha_{t,r} - \alpha_{j,r}$	$n \neq r \; t \neq j$
8	$\beta_{l,n} - \beta_{r,n} - \alpha_{j,r}$	$r \neq l$
9	$\beta_{l,n} - \beta_{t.n} + \beta_{t,r} - \alpha_{j,r}$	$n \neq r \; t \neq l$

TAB. 4.3 Indices des termes en y^u dans l'équation (4.18)

Conditions	terme	restriction
1	$-\alpha_{r,l} + \alpha_{r,j}$	$j \neq l$
2	$-\alpha_{r,l} + \alpha_{r,n} - \beta_{j,n}$	$n \neq l$
3	$-\beta_{j,l}$	
4	$-\beta_{r,l} - \beta_{j,r}$	
5	$-\beta_{r,l} + \beta_{r,j}$	$j \neq l$
6	$-\beta_{r,l} + \beta_{r,n} - \beta_{j,n}$	$r \neq j \; n \neq l$
7	$\beta_{l,j}$	
8	$\beta_{l,r} + \beta_{r,j}$	
9	$\beta_{l,r} + \beta_{r,n} - \beta_{j,n}$	$r \neq j$
10	$\beta_{l,r} - \alpha_{n,r} + \alpha_{n,j}$	$r \neq j$
11	$\beta_{l,r} - \alpha_{t,r} + \alpha_{t,n} - \beta_{j,n}$	$r \neq n$
12	$\beta_{l,r} - \beta_{j,r}$	$l \neq j$
13	$\beta_{l,r} - \beta_{n,r} - \beta_{j,n}$	$n \neq l$
14	$\beta_{l,r} - \beta_{n,r} + \beta_{n,j}$	$r \neq j \; n \neq l$
15	$\beta_{l,r} - \beta_{t,r} + \beta_{t,n} - \beta_{j,n}$	$r \neq n \; t \neq l \; t \neq j$

TAB. 4.4 Indices des termes en y^p dans l'équation (4.18)

Cependant, certaines de ces conditions sont redondantes et peuvent donc être simplifiées. Tout d'abord, si nous réorganisons les tableaux 4.2 et 4.3, nous remarquons qu'ils donnent les mêmes conditions au signe - près et en inversant les indices l et j :

Conditions du tableau 4.2			Conditions du tableau 4.3		
N°	terme	restriction	N°	terme	restriction
1	$\alpha_{l,j}$		1	$-\alpha_{j,l}$	
2	$\alpha_{l,r} + \beta_{r,j}$		3	$-\beta_{r,l} - \alpha_{j,r}$	
3	$\alpha_{l,r} + \beta_{r,n} - \beta_{j,n}$	$r \neq j$	8	$\beta_{l,n} - \beta_{r,n} - \alpha_{j,r}$	$r \neq l$
4	$\alpha_{l,r} - \alpha_{n,r} + \alpha_{n,j}$	$n \neq l\ r \neq j$	2	$-\alpha_{n,l} + \alpha_{n,r} - \alpha_{j,r}$	$n \neq j\ r \neq l$
5	$\alpha_{l,r} - \alpha_{t,r} + \alpha_{t,n} - \beta_{j,n}$	$n \neq r\ t \neq l$	7	$\beta_{l,n} - \alpha_{t,n} + \alpha_{t,r} - \alpha_{j,r}$	$n \neq r\ t \neq j$
6	$\alpha_{l,r} - \beta_{j,r}$		6	$\beta_{l,r} - \alpha_{j,r}$	
7	$\alpha_{l,r} - \beta_{n,r} - \beta_{j,n}$		5	$\beta_{l,n} + \beta_{n,r} - \alpha_{j,r}$	
8	$\alpha_{l,r} - \beta_{n,r} + \beta_{n,j}$	$r \neq j$	4	$-\beta_{n,l} + \beta_{n,r} - \alpha_{j,r}$	$r \neq l$
9	$\alpha_{l,r} - \beta_{t,r} + \beta_{t,n} - \beta_{j,n}$	$n \neq r\ t \neq j$	9	$\beta_{l,n} - \beta_{t,n} + \beta_{t,r} - \alpha_{j,r}$	$n \neq r\ t \neq l$

TAB. 4.5 Comparaison des termes des tableaux 4.2 et 4.3

Nous pouvons donc éliminer les conditions du tableau 4.3.

Les autres simplifications sont plus délicates à démontrer. Par exemple, si on regarde les conditions 2 et 3 du tableau 4.2, nous pouvons prouver que si tous les termes de la condition 3 sont différents, alors tous les termes de la condition 2 seront obligatoirement différents :

Nous procédons par l'absurde. Supposons donc que la condition 3 est vérifiée et que la condition 2 ne l'est pas. Il existe donc r_1 et r_2 dans $\{1, \ldots, J\}$ avec $r_1 \neq r_2$ tels que pour l et j dans $\{1, \ldots, J\}$,

$$\alpha_{l,r_1} + \beta_{r_1,j} = \alpha_{l,r_2} + \beta_{r_2,j}$$

Pour $J > 2$, il est possible de fixer un j_2 dans $\{1, \ldots, J\}$ avec $j_2 \neq r_1$ et $j_2 \neq r_2$. Alors, en

retranchant le terme $\beta_{j_2,j}$ de chaque côté de l'égalité précédente, nous obtenons :

$$\alpha_{l,r_1} + \beta_{r_1,j} - \beta_{j_2,j} = \alpha_{l,r_2} + \beta_{r_2,j} - \beta_{j_2,j}$$

En résumé, pour l et j_2 fixés dans $\{1, \ldots, J\}$, nous avons trouvé r_1 et r_2 avec $r_1 \neq j_2$ et $r_2 \neq j_2$ tels qu'il existe un j dans $\{1, \ldots, J\}$ tels que

$$\alpha_{l,r_1} - \beta_{j_2,j} + \beta_{r_1,j} = \alpha_{l,r_2} - \beta_{j_2,j} + \beta_{r_2,j}$$

ce qui est en contradiction avec la condition 3. Ainsi, si tous les termes de la condition 3 sont différents, tous les termes de la condition 2 le seront aussi. Mais cette démonstration ne peut suffir pour retirer la condition 2 du tableau, puisqu'il faudrait démontrer que tous les autres termes du tableau 4.2 sont aussi différents des termes engendrés par la condition 2.

Une méthode simplificatrice a été utilisée en construisant des codes $R - CSO^2C - SS$ en enlevant une à une les conditions des tableaux. Ensuite, les codes obtenus sont soumis à la validation de l'ensemble des conditions. Ainsi, si ces codes vérifient toujours l'ensemble des conditions, nous pouvons déduire que la condition retirée n'était pas nécessaire. Cette méthode n'est pas très formelle, mais tous les codes obtenus dans la suite du mémoire vérifient bien toutes les conditions. Les conditions simplifiées sont présentées dans le tableau 4.6, où tous les termes provenant des conditions de chacun des 3 ensembles doivent être différents entre eux.

Ensemble	Conditions	terme	restriction
1	1	$\alpha_{l,r} - \alpha_{j,n} - \beta_{n,r}$	
	2	$\alpha_{l,r} - \alpha_{j,n} + \alpha_{t,n} - \alpha_{t,r}$	$n \neq r\ t \neq l\ t \neq j$
2	1	$\alpha_{l,r} + \beta_{r,j}$	
	2	$\alpha_{l,r} - \beta_{j,n} - \beta_{n,r}$	
	3	$\alpha_{l,r} - \beta_{j,n} + \beta_{r,n}$	$r \neq j$
	4	$\alpha_{l,r} + \alpha_{n,j} - \alpha_{n,r}$	$n \neq l\ r \neq j$
	5	$\alpha_{l,r} - \beta_{j,n} + \alpha_{t,n} - \alpha_{t,r}$	$n \neq r\ t \neq l$
	6	$\alpha_{l,r} - \beta_{j,n} + \beta_{t,n} - \beta_{t,r}$	$n \neq r\ t \neq j$
3	1	$\beta_{l,r} + \beta_{r,j}$	
	2	$-\alpha_{r,l} - \beta_{j,n} + \alpha_{r,n}$	$n \neq l$
	3	$\beta_{l,r} + \alpha_{n,j} - \alpha_{n,r}$	$r \neq j$
	4	$-\beta_{r,l} - \beta_{j,n} + \beta_{r,n}$	$r \neq j\ n \neq l$
	5	$\beta_{l,r} - \beta_{j,n} + \beta_{r,n}$	$r \neq j$
	6	$\beta_{l,r} - \beta_{j,n} - \beta_{n,r}$	$n \neq l$
	7	$\beta_{l,r} - \beta_{j,n} + \alpha_{t,n} - \alpha_{t,r}$	$r \neq n$
	8	$\beta_{l,r} - \beta_{j,n} + \beta_{t,n} - \beta_{t,r}$	$r \neq n\ t \neq l\ t \neq j$

TAB. 4.6 Conditions d'un code $R - CSO^2C - SS$

Il aurait été possible de simplifier les conditions d'une manière différente. Cependant, cette configuration a donné le plus petit nombre de conditions et ne peux pas être réduite d'avantage. Il est important de noter que nous avons supposé tous les β strictement positifs. Si ce n'était pas le cas, les conditions sont toujours valides, il suffirait tout simplement de ne pas considérer les termes faisant intervenir les β valant 0.

4.2 Construction de nouveaux codes $R - CSO^2C - SS$

4.2.1 Recherche de codes

L'algorithme de construction des codes $R - CSO^2C - SS$ utilisé est un algorithme aléatoire. Il faut l'initialiser en indiquant la taille des matrices, J, la valeur maximale autorisée des connexions, max, ainsi qu'un vecteur $position$ qui regroupe les emplacements

des connexions dans les matrices, puisque les meilleures performances d'erreur ne seront pas obtenues avec des matrices pleines. On notera $\boldsymbol{H}$ la matrice des connexions, qui sont numérotées dans le vecteur *position* en lisant le long des colonnes comme suit :

$$\boldsymbol{H} = \left(\frac{\boldsymbol{\alpha}}{\boldsymbol{\beta}}\right) = \left(\begin{array}{ccccc} 1 & \vdots & \cdots & \vdots & 2\cdot J\cdot(J-1)+1 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ J & \vdots & \cdots & \vdots & 2\cdot J\cdot(J-1)+J \\ \hline J+1 & \vdots & \cdots & \vdots & 2\cdot J\cdot(J-1)+J+1 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ 2\cdot J & \vdots & \cdots & \vdots & 2\cdot J\cdot J \end{array}\right)$$

Par exemple, si $J = 3$, pour avoir une matrice où les connexions doivent être à l'emplacement des $\times$ dans $\boldsymbol{H}$ telle que

$$\boldsymbol{H} = \left(\frac{\boldsymbol{\alpha}}{\boldsymbol{\beta}}\right) = \left(\begin{array}{ccc} \times & \circ & \times \\ \times & \times & \times \\ \circ & \circ & \times \\ \hline \circ & \times & \circ \\ \times & \circ & \times \\ \circ & \circ & \times \end{array}\right)$$

le vecteur *position* sera $\{1\ 2\ 5\ 8\ 10\ 13\ 14\ 15\ 17\ 18\}$. La longueur du vecteur *position*, valant le nombre total de connexions, sera notée *total*. On se sert aussi d'un compteur,

noté cpt, indiquant le numéro de la connexion en cours de recherche située dans la matrice $\boldsymbol{H}$ à l'emplacement $position(cpt)$. Enfin, on place dans le vecteur $liste$ toutes les valeurs que la connexion en cours de recherche peut prendre. Ainsi, $liste$ vaudra $\{0, \ldots, max\}$ si on cherche une connexion dans la sous matrice $\boldsymbol{\alpha}$ et $\{1, \ldots, max\}$ si on est dans la sous matrice $\boldsymbol{\beta}$, où les connexions ne peuvent valoir 0.

Le principe de l'algorithme est décrit à la figure 4.3. Nous partons d'une matrice $\boldsymbol{H}$

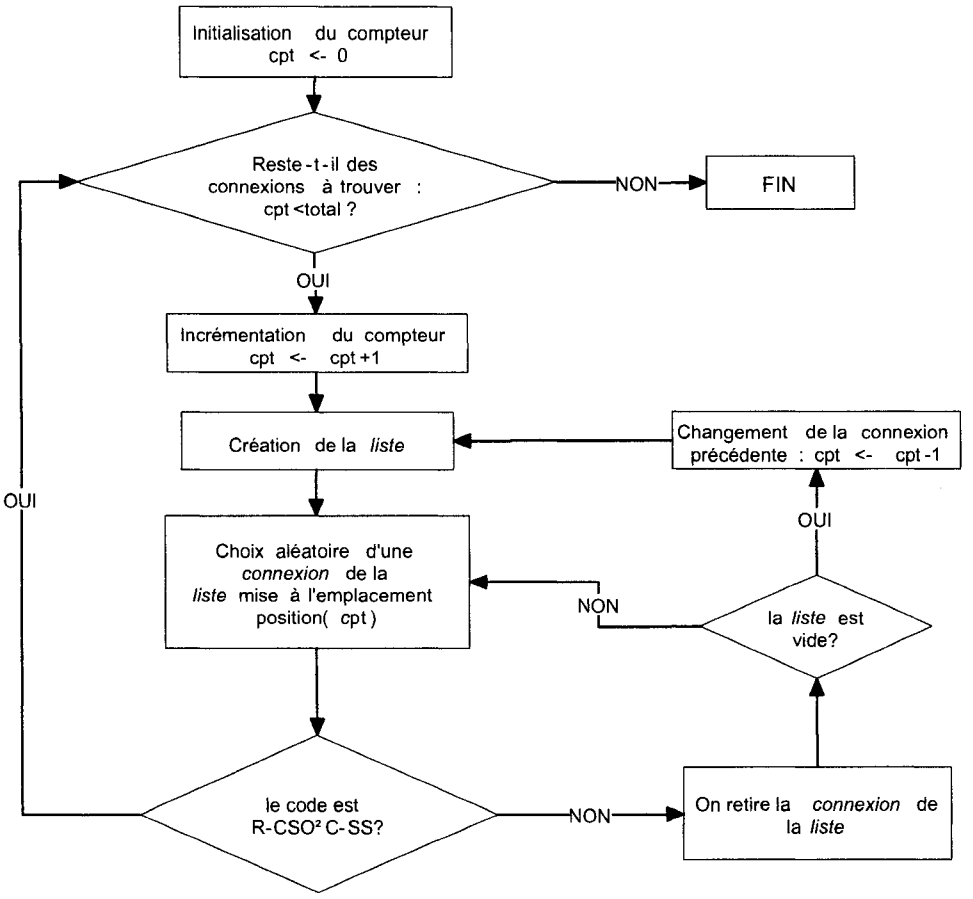

FIG. 4.3 Algorithme de recherche aléatoire des codes $R - CSO^2C - SS$

vide et incrémentons le compteur cpt à 1. On construit alors la $liste$ des valeurs possibles,

et on en prend une au hasard, appelée $connexion$, qu'on met à $position(cpt)$, représentant sa place dans la matrice $\boldsymbol{H}$. Si la matrice ainsi définie vérifie les conditions d'un code $R - CSO^2C - SS$, on incrémente cpt, et on recommence jusqu'à ce qu'on ait toutes les connexions désirées. Si le code ne vérifie pas les conditions du tableau 4.6, on retire la $connexion$ de la $liste$, et on revérifie avec une autre valeur prise aléatoirement dans ce qu'il reste de la $liste$. Dans le cas où toutes les valeurs de la $liste$ ont été testées sans succès, on revient en arrière pour changer les connexions précédentes.

Précisons que nous nous sommes restreints dans cette recherche à des codes dont la sous matrice α est pleine, c'est à dire que lors de la recherche des codes pour un J quelconque, le vecteur $position$ contient tous les éléments $j + 2J(k-1)$ avec $(k, j) \in \{1, \ldots, J\}$.

4.2.2 Réduction du span

Un autre algorithme a été utilisé afin de réduire la latence d'un code obtenu par la méthode de construction décrite à la section précédente. Il est montré à la figure 4.4 et avait initialement été proposé dans [1] pour réduire la longueur des codes $CSO^2C - SS$.

On commence par mettre en mémoire la matrice $\boldsymbol{H}$ initiale sous $\boldsymbol{H'}$. L'idée est de tester chacun des $2 \cdot J^2$ éléments de la matrice $\boldsymbol{H}$ en les remplacant par un compteur qui commence à la valeur 0 (ou 1 si l'élément est dans la sous matrice β) et qui s'incrémente. Á chaque valeur du compteur, on effectue un test pour savoir si la nouvelle matrice $\boldsymbol{H}$ décrit un code $R - CSO^2C - SS$. On gardera comme connexion valide la plus petite valeur trouvée du compteur qui remplisse les conditions du tableau 4.6. Dans le pire des

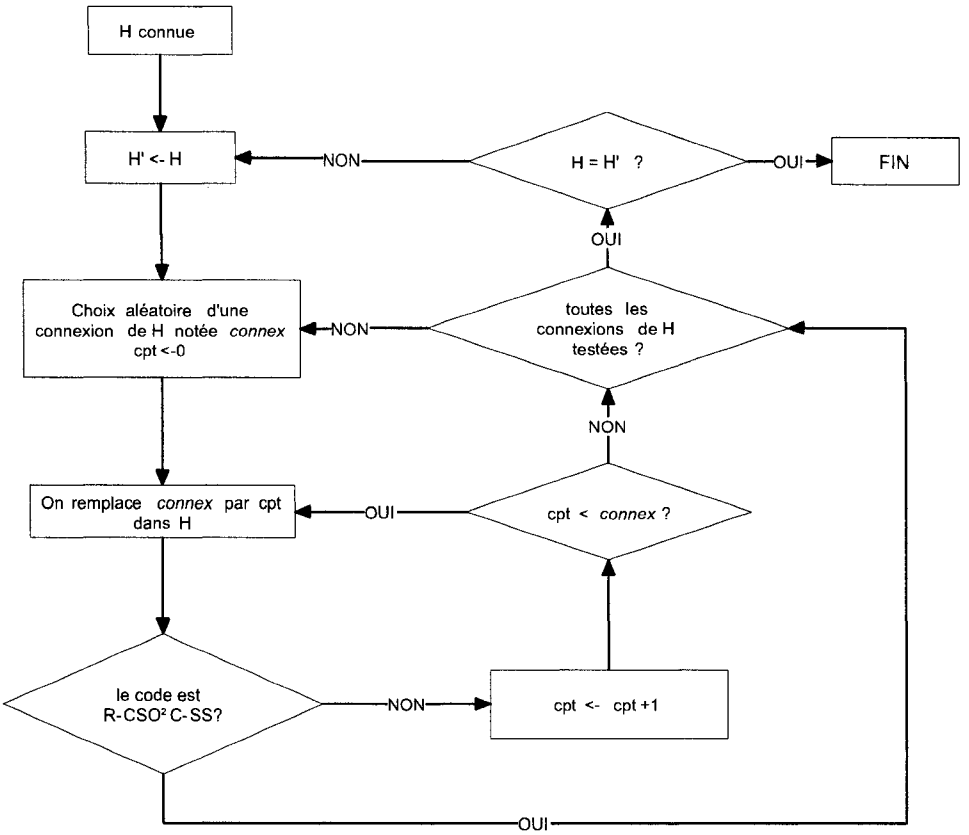

FIG. 4.4 Algorithme de réduction du span des codes $R - CSO^2C - SS$

cas, cette valeur du compteur vaut la valeur d'origine de la matrice $\boldsymbol{H}$. Enfin, quand ces étapes ont été complétées pour toutes les connexions de la matrice $\boldsymbol{H}$, on regarde si $\boldsymbol{H}$ et $\boldsymbol{H'}$ sont égales, auquel cas aucune simplification n'a été possible. Sinon, on recommence tout le processus jusqu'à ce qu'il n'y ait plus de réduction possible.

Notons que l'ordre dans lequel on tente de réduire la valeur des éléments de la matrice $\boldsymbol{H}$ est pris aléatoirement. En effet, le fait de réduire une valeur plutôt qu'une autre affecte le reste de la recherche. Ainsi, pour réduire efficacement le span d'un code représenté par sa matrice $\boldsymbol{H}$, il est important de répéter plusieurs fois l'algorithme à partir de la matrice de

départ car les résultats sont rarement identiques.

Ainsi, le code testé dans [4] pour $J = 5$, avec un span de 1135, a pu être réduit à un span de 247 selon :

$$\boldsymbol{H1} = \left(\begin{array}{ccccc} 0 & 299 & 0 & 142 & 1135 \\ 0 & 320 & 146 & 407 & 794 \\ 0 & 679 & 5 & 977 & 172 \\ 87 & 333 & 0 & 106 & 310 \\ 0 & 418 & 60 & 123 & 99 \\ \hline 0 & 945 & 0 & 0 & 92 \\ 1135 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1003 & 0 \\ 0 & 0 & 193 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right) \equiv \boldsymbol{H2} = \left(\begin{array}{ccccc} 0 & 30 & 0 & 0 & 247 \\ 0 & 2 & 146 & 107 & 39 \\ 0 & 19 & 3 & 9 & 172 \\ 87 & 4 & 0 & 106 & 1 \\ 0 & 184 & 4 & 123 & 99 \\ \hline 0 & 225 & 0 & 0 & 3 \\ 16 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 34 & 0 \\ 0 & 0 & 71 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

Cependant, nous verrons dans la prochaine partie que réduire la latence d'un code connu peut avoir des répercutions sur ses performances d'erreur.

4.3 Simulations

Nous présentons dans cette section plusieurs résultats obtenus pour les codes $R - CSO^2C - SS$. Différentes pistes ont été suivies afin d'obtenir les meilleures performances d'erreur possibles. Nous nous sommes tout d'abord intéressés à étudier, pour une confi-

guration de matrice donnée, l'influence du span sur les performances d'erreur. Ensuite, comme il avait été observé dans [4] que ces codes nécessitaient un grand nombre d'itérations pour converger, la deuxième idée a été d'essayer de réduire au maximum le span de différents codes pour J=3. Enfin, en restant à span constant, nous avons fait varier le nombre de connexions et leurs emplacements dans la partie récursive du codeur.

4.3.1 Importance du span des codeurs

4.3.1.1 Exemple avec J=5

Les meilleurs résultats qui ont été publiés dans [4] utilisent la matrice $\boldsymbol{H}1$, avec J=5 et un span de 1135, présentée dans la section précédente. On présente les performances de ce code à la figure 4.5.

On peut voir sur la figure 4.5 l'amélioration des performances d'erreur avec le nombre d'itérations utilisées. Par exemple, après 10 itérations, on obtient une probabilité d'erreur par bit de 1×10^{-4} pour $E_b/N_0 = 1.95dB$. En revanche, après 50 itérations, on atteint la même probabilité pour $E_b/N_0 = 1.2dB$, ce qui représente un gain de codage de 0.75 dB. Cependant, plus on utilise d'itérations, plus la latence au décodage est importante. En effet, pour les codes $R - CSO^2C - SS$, chaque itération produit une latence de $J \cdot m$. Ainsi, dans le cas du code représenté par la matrice $\boldsymbol{H}1$, un décodage avec 50 itérations engendre un retard de $50 \cdot 5 \cdot 1135 = 283750$ bits. Un décodage de 10 itérations, lui, produira une latence de "seulement" 56750 bits.

Pour tous les codes doublement orthogonaux connus, les performances de correction

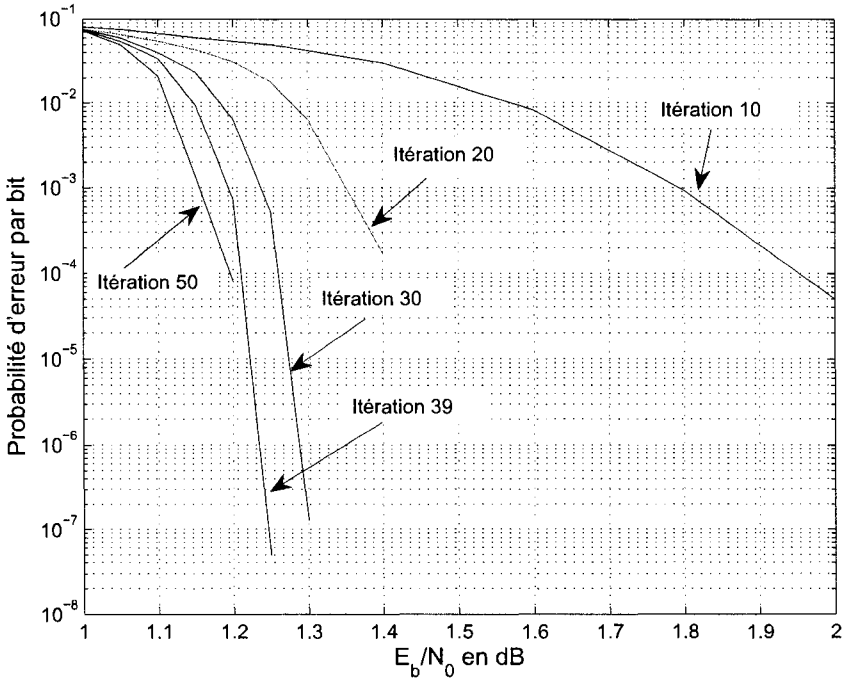

FIG. 4.5 Simulation du code $R - CSO^2C - SS$ avec R=1/2, J=5 et m=1135 représenté par la matrice $\boldsymbol{H}1$

d'erreurs dépendaient uniquement du nombre de connexions utilisées. Nous avons donc toujours cherché à minimiser le span des codeurs, ce qui ne détériorait pas les résultats. Ainsi, nous sommes parvenus, pour J=5, à réduire le span à 247, sans changer la place des connexions. Le code est représenté par la matrice $\boldsymbol{H2}$ de la section précédente. Les performances d'erreur sont indiquées à la figure 4.6. On observe une faible détérioration des

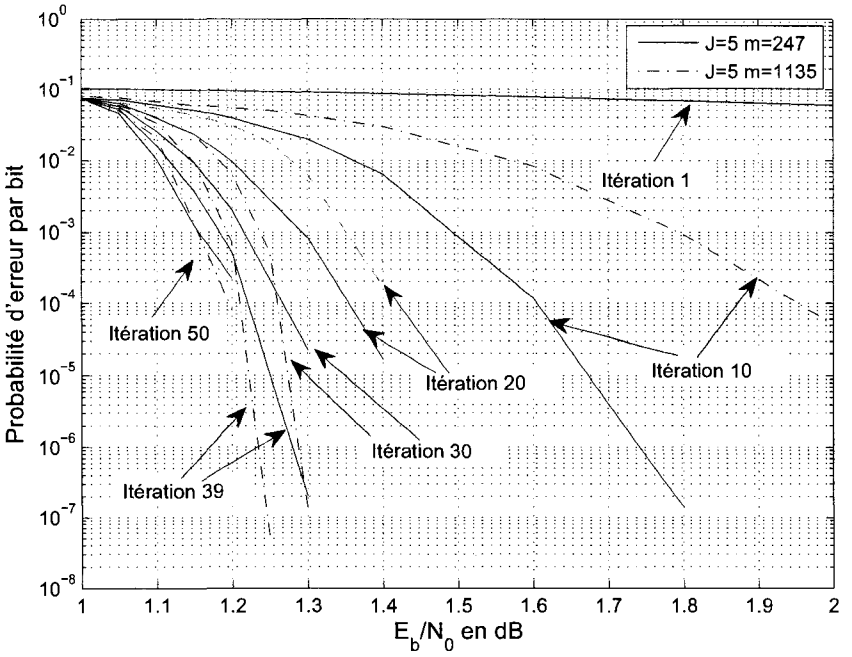

FIG. 4.6 Simulation du code $R - CSO^2C - SS$ avec R=1/2, J=5 et m=247 représenté par la matrice $\boldsymbol{H2}$

performances d'erreur au fil des itérations. Le tableau 4.7 compare les probabilités d'erreur par bit obtenues par chacun des codes simulés pour J=5, pour les différentes itérations représentées à la figure 4.6.

On voit que les performances du code avec le plus petit span sont meilleures pour les

Nombre d'itérations	10	20	30	39	50
E_b/N_0 (dB)	1.8	1.4	1.3	1.25	1.2
span de 1135	9×10^{-4}	1.7×10^{-4}	1.3×10^{-7}	5×10^{-8}	8×10^{-5}
span de 247	1.3×10^{-7}	1.7×10^{-5}	2×10^{-5}	9×10^{-6}	2×10^{-4}
Rapport des performances	6.9×10^{3}	10	6.5×10^{-3}	5.5×10^{-3}	4×10^{-1}

TAB. 4.7 Comparaison des performances au fil des itérations de 2 codes $R-CSO^2C-SS$ avec R=1/2 pour J=5 avec des span de 1135 et 247.

premières itérations. Cependant, au fur et à mesure que les itérations augmentent, c'est le code avec le plus grand span qui obtient les plus faibles probabilités d'erreur par bit. Ces différences de performance sont probablement dûes à l'impact du span sur les cycles. Toutefois, si on compare les performances des deux codes en terme de latence, c'est le code le plus court qui est le plus intéressant. En effet, le rapport des spans étant de 4.6, le décodage du code avec le span de 1135 après 10 itérations produit presque la même latence que le décodage du code avec le span de 247 après 50 itérations, pour des performances d'erreur bien moins bonnes. Mais le code le plus long pourra donner des performances inatteignables pour le code le plus court. Tout dépend des contraintes d'utilisation. Si la latence est une contrainte importante, on préfèrera le code le plus court. Si c'est la performance qui compte, on se servira du code le plus long.

4.3.1.2 Exemple avec J=3

Pour confirmer les résultats observés pour J=5, nous avons effectué un autre test à J=3, avec des matrices qui contiennent toutes les connexions dans la partie récursive. Les deux codeurs, respectivement de spans maximaux 119 et 195, sont représentés par les matrices

$\boldsymbol{H3}$ et $\boldsymbol{H4}$ suivantes :

$$\boldsymbol{H3} = \left(\begin{array}{ccc} 1 & 12 & 85 \\ 106 & 63 & 8 \\ 16 & 61 & 78 \\ \hline 21 & 115 & 103 \\ 112 & 119 & 40 \\ 111 & 63 & 14 \end{array}\right) \text{ et } \boldsymbol{H4} = \left(\begin{array}{ccc} 193 & 101 & 96 \\ 26 & 0 & 97 \\ 49 & 44 & 20 \\ \hline 182 & 195 & 155 \\ 135 & 172 & 5 \\ 124 & 2 & 151 \end{array}\right)$$

Les simulations effectuées avec ces deux codes ont donné les résultats présentés aux figures 4.7 et 4.8.

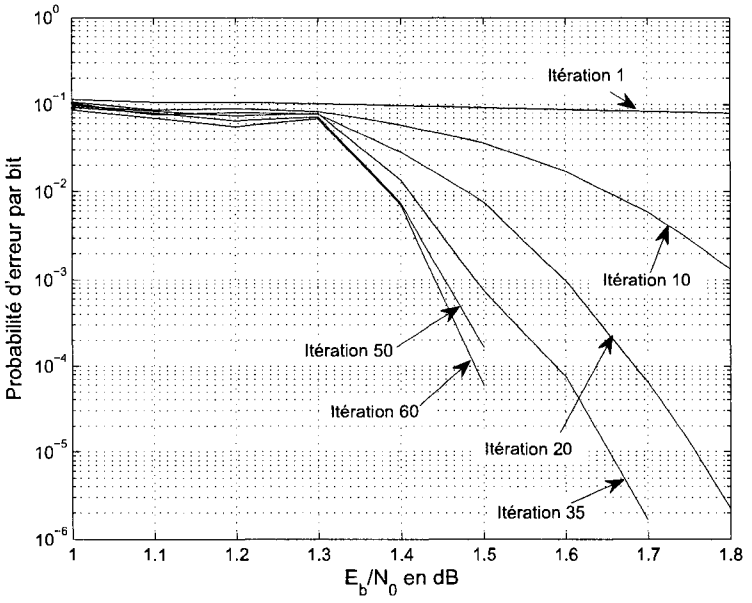

FIG. 4.7 Simulation du code $R - CSO^2C - SS$ avec R=1/2, J=3 et m=119 représenté par la matrice $\boldsymbol{H3}$

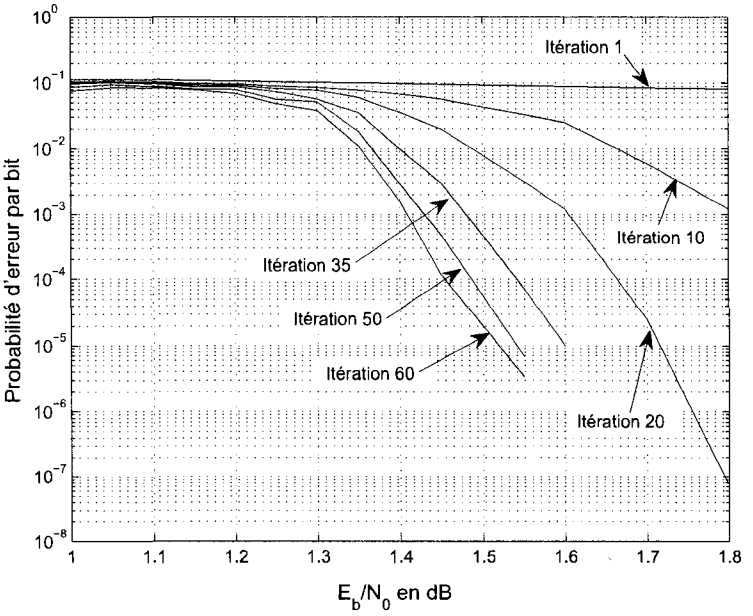

FIG. 4.8 Simulation du code $R - CSO^2C - SS$ avec R=1/2, J=3 et m=195 représenté par la matrice $\boldsymbol{H4}$

Ce qu'on y observe est que les deux codes ont les mêmes performances pour les 10 premières itérations. Ensuite, comme pour la partie précédente, le code de plus petite longueur obtient, pour une valeur de E_b/N_0 donnée, de moins bonnes performances. Par exemple, si on se place à un rapport $E_b/N_0 = 1.5dB$, nous obtenons, après 50 itérations de décodage, une probabilité d'erreur par bit de 2.0×10^{-4} pour le code de plus petit span représenté par la matrice $\boldsymbol{H3}$. Pour l'autre code, de span 195, au même rapport signal sur bruit, la probabilité d'erreur par bit après 50 itérations est de 7.0×10^{-5}. Le gain de codage résultant est de l'ordre de 0.3dB.

En conclusion à cette première partie de résultats, nous pouvons affirmer que les spans des codes influencent les performances de correction d'erreurs. Nous verrons d'autres

exemples plus frappants de ce phénomène plus loin dans cette section. Cette propriété est intéressante car nouvelle pour les codes convolutionnels doublement orthogonaux, dont les performances d'erreur ne dépendaient jusqu'ici que du nombre de connexions du codeur. Il en résulte que la recherche des codes $R - CSO^2C - SS$ doit être différente de celle des autre codes. En effet, bien qu'il puisse être intéressant d'étudier le comportement des codes avec un petit span, comme nous le verrons dans la prochaine partie, les meilleures performances d'erreur seront obtenues avec des spans plus grands.

4.3.2 Plus petits spans trouvés pour J=3

Les observations de la partie précédente montrent que les performances d'erreur s'améliorent avec la longueur des spans des codes pour les grandes valeurs d'itérations. Mais nous avons quand même cherché à réduire au maximum la longueur de différents codes pour J=3. Ainsi, la plus petite latence obtenue pour une matrice pleine est de 119 et correspond à la matrice $\boldsymbol{H3}$ présentée à la section précédente. Les performances sont présentées à la figure 4.7. Pour pouvoir diminuer cette latence, il a fallu rechercher des codes dont la partie récursive, c'est à dire la sous matrice β, n'est pas complète. Nous nous sommes intéressés aux deux cas suivants : avec 1 connexion par ligne/colonne dans la sous matrice β et avec 2 connexions par ligne/colonne.

4.3.2.1 Avec 1 connexion par ligne/colonne dans la sous matrice β

Avec 1 seule connexion par ligne et colonne dans la partie récursive de la matrice, le plus petit span obtenu est de 10. La matrice de représentation du code est :

$$\boldsymbol{H5} = \left(\begin{array}{ccc} 7 & 3 & 1 \\ 3 & 9 & 0 \\ 0 & 0 & 10 \\ \hline 0 & 0 & 8 \\ 9 & 0 & 0 \\ 0 & 10 & 0 \end{array}\right)$$

Cette diminution de plus de 90% du span est dûe au fait que tous les termes qui faisaient intervenir les connexions retirées dans la vérification des conditions du tableau 4.6 ont été enlevés. Ainsi, le nombre de différences devant être distinctes a été considérablement réduit, d'où cette réduction du span.

Les performances d'erreur de ce code sont données à la figure 4.9.

Un ajustement des coefficients de pondération a été nécessaire. On utilise $a_u = 0.8$ et $a_p = 0.9$. On peut observer que les performances sont nettement moins bonnes que pour les autres codes $R-CSO^2C-SS$ vus jusqu'ici. Cependant, la latence totale d'une itération est de 30 bits, ce qui est très peu. De plus, le nombre total d'itérations nécessaires au décodage est faible, puisque l'algorithme de décodage a presque atteint sa convergence après 10 itérations seulement. Pour être sûr de la validité des résultats, nous considérons qu'il faut

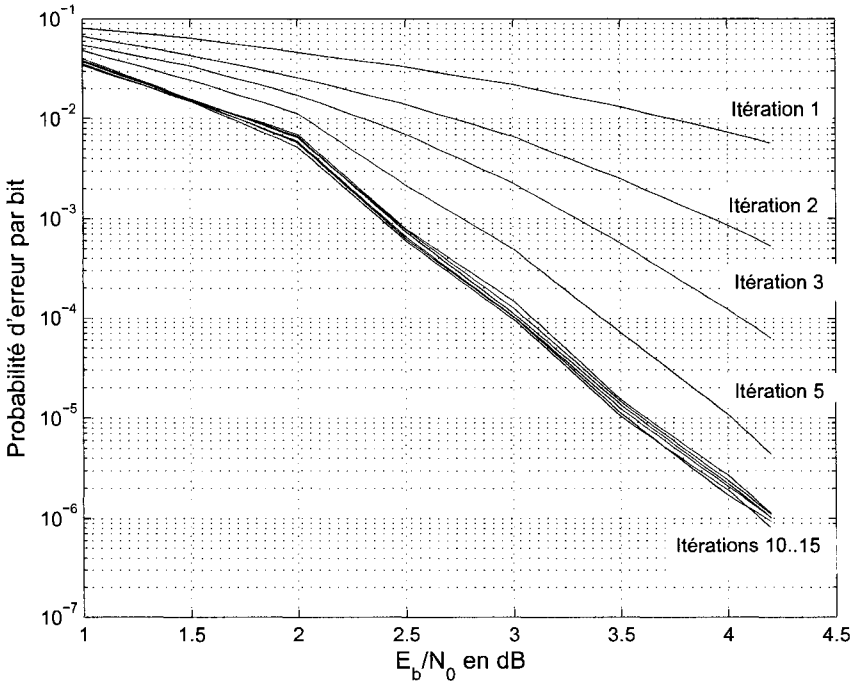

FIG. 4.9 Simulation du code $R - CSO^2C - SS$ avec R=1/2, J=3 et m=10 représenté par la matrice $\boldsymbol{H5}$

15 itérations pour faire le décodage. Ceci reste faible par rapport aux codes de plus grandes longueurs, comme ceux représentés par les matrice $\boldsymbol{H1}$ à $\boldsymbol{H4}$, qui nécessitent au moins 50 itérations pour obtenir les meilleures performances possibles. Ainsi, pour $E_b/N_0 = 3.5dB$, on obtient une probabilité d'erreur par bit de 1.0×10^{-6} après 15 itérations, ce qui signifie que le nombre de bits que le décodeur a besoin de garder en mémoire pour effectuer le décodage est seulement de 450.

On peut se donner une idée plus précise de ces performances en le comparant aux codes vus précédemment dans ce mémoire à la figure 4.10 :

Il est clair qu'aucun des autres codes CSO^2C ne parvient, pour $E_b/N_0 = 3.5dB$, à de

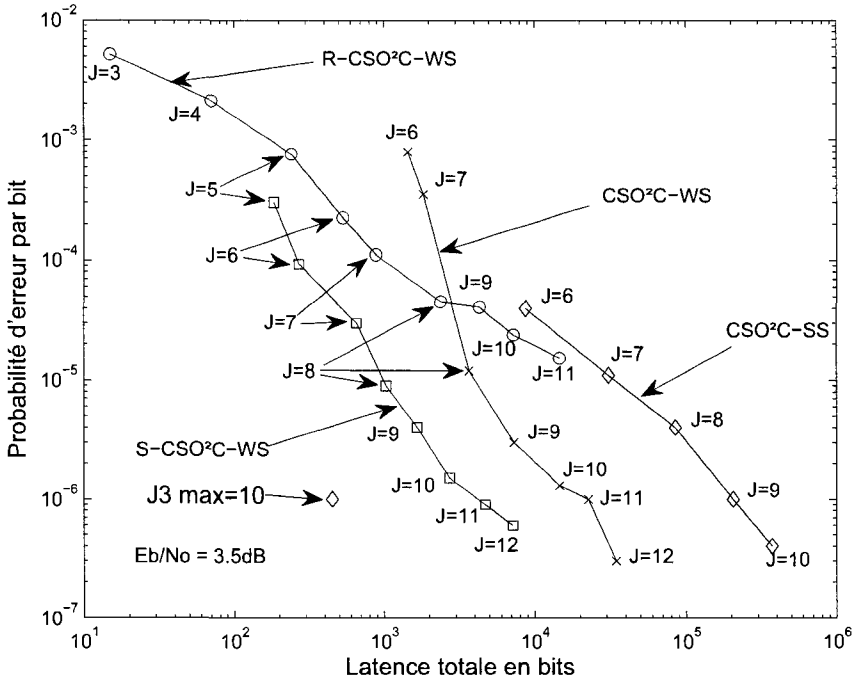

FIG. 4.10 Comparaison du code $R - CSO^2C - SS$ avec R=1/2, J=3 et m=10, avec les autres codes CSO^2C pour $E_b/N_0 = 3.5dB$

meilleures performances de correction d'erreurs pour une latence équivalente. En effet, le code affiché avec les performances les plus semblables, soit le code $S - CSO^2C - WS$ pour J=11, utilise une latence totale d'environ 4500 bits, soit 10 fois plus que la latence utilisée par le code $R - CSO^2C - SS$ pour J=3 avec un span de 10.

Ainsi, bien que ce code ne soit pas très performant pour de faibles rapports signal sur bruit, son petit span, ainsi que le faible nombre d'itérations nécessaires au décodage, le rendent très intéressant à de plus hautes valeurs de E_b/N_0.

4.3.2.2 Avec 2 connexions par ligne/colonne dans la sous matrice β

Le plus petit code obtenu avec deux connexions par ligne et colonne pour la sous matrice β est :

$$\boldsymbol{H6} = \left(\begin{array}{ccc} 33 & 26 & 4 \\ 7 & 15 & 0 \\ 20 & 0 & 21 \\ \hline 0 & 27 & 28 \\ 17 & 0 & 27 \\ 31 & 10 & 0 \end{array}\right)$$

Le span est donc de 33. Ses performances d'erreur sont indiquées à la figure 4.11.

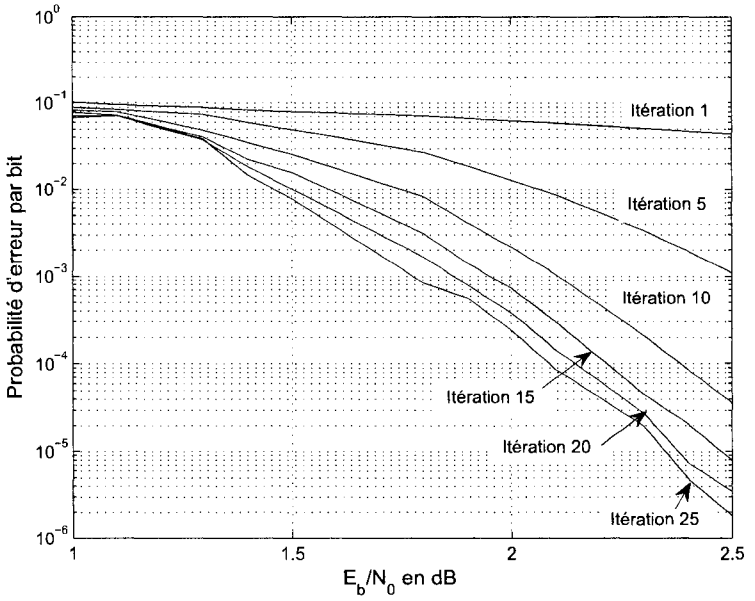

FIG. 4.11 Simulation du code $R - CSO^2C - SS$ avec R=1/2, J=3 et m=33 représenté par la matrice $\boldsymbol{H6}$

Les résultats sont meilleurs que ceux donnés par la matrice $\boldsymbol{H5}$ de span 10 avec un gain de codage d'environ 0.6dB pour une probabilité d'erreur par bit de 2.0×10^{-4}. De plus, le nombre d'itérations nécessaires reste assez faible (25). Comme précédemment, le faible span de ce code nous ammène à le comparer avec d'autres codes doublement orthogonaux non récursifs. Les résultats, pour $E_b/N_0 = 2.5dB$, sont donnés à la figure 4.12.

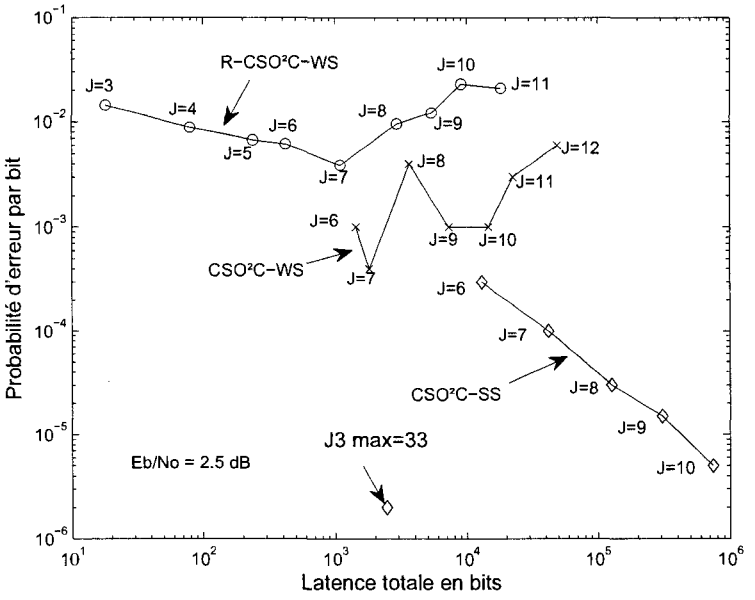

FIG. 4.12 Comparaison du code $R - CSO^2C - SS$ avec R=1/2, J=3 et m=33, avec les autres codes CSO^2C pour $E_b/N_0 = 2.5dB$

Á nouveau, le code $R - CSO^2C - SS$ est, avec une faible latence, beaucoup plus performant que les autres codes doublement orthogonaux. Les performances les plus proches sont obtenues par le code $CSO^2C - SS$ pour J=10 avec une probabilité d'erreur par bit de 5×10^{-6} contre 2×10^{-6} pour le code récursif. Mais le code non récursif a besoin de garder en mémoire 300 fois plus de bits que notre code avec le span de 33 (750000 contre 2500).

Il avait été observé dans [4] que les performances des codes $R - CSO^2C - SS$ dépendaient beaucoup du nombre de connexions dans la partie récursive du codeur, et que le fait de diminuer ce nombre permettait d'améliorer les performances. Il ne faut pas penser que les résultats présentés ici sont en contradiction avec ce qui avait été avancé. En effet, les deux codes qui ont été étudiés dans cette section ont de très faibles spans, ce qui explique pourquoi ils ne sont pas performants en dessous d'une valeur de E_b/N_0 de 2dB. Pour obtenir les mêmes observations que [4], il faut se placer avec un span constant. Ainsi, pour un J donné, on construit une matrice pleine à partir de laquelle on enlève des connexions. Cette méthode nous permet d'évaluer l'influence du nombre de connexions dans la partie récursive sur les performances d'erreur, sans que des variations de span perturbent les résultats.

4.3.3 Étude de différentes dispositions des connexions dans la sous matrice β

L'idée de cette section est d'essayer de déterminer la meilleure configuration possible des emplacements des connexions. Pour cela, nous avons fait varier leur nombre ainsi que leur répartition sur les lignes et colonnes de la partie récursive de la matrice.

4.3.3.1 Dispositions possibles pour 1 connexion par colonne dans la sous matrice β

Nous avons mentionné à la section 4.1.2, que le nombre d'équations qui servaient à décoder le symbole $p_{l,i}$, à l'instant $i = 0, 1, 2, \ldots$, avec $l \in \{1, \ldots, J\}$, était le nombre N_l de connexions à la ligne l de la sous matrice β. Pour savoir s'il était préférable de répartir les connexions sur les différentes lignes pour que les symboles $p_{l,i}$ aient tous un nombre identique d'équations pour être décodés, nous avons trouvé un ensemble de connexions qui vérifient l'ensemble des conditions des codes $R - CSO^2C - SS$ du tableau 4.6 pour plusieurs configurations de matrice. Les dispositions étudiées ont toutes la même partie α qui est la suivante :

$$\alpha = \begin{pmatrix} 142 & 133 & 123 & 138 \\ 33 & 114 & 66 & 116 \\ 90 & 67 & 91 & 136 \\ 72 & 1 & 118 & 0 \end{pmatrix}$$

En prenant une connexion par colonne dans la sous matrice β, et donc un nombre total de connexions dans la sous matrice β de 4, les 5 cas qui ont été étudiés sont résumés dans le tableau 4.8. Chaque ligne représente un des cas étudiés. Chaque colonne indique le nombre de connexions par ligne.

Par exemple, la matrice β_1 a 1 ligne avec 4 connexions et 3 lignes avec 0 connexion.

β_i \ Nb connex	4 connex	3 connex	2 connex	1 connex	0 connex
β_1	1	0	0	0	3
β_2	0	1	0	1	2
β_3	0	0	2	0	2
β_4	0	0	1	2	1
β_5	0	0	0	4	0

TAB. 4.8 Répartition des lignes en fonction du nombre de leurs connexions

Elle s'écrit :

$$\beta_1 = \begin{pmatrix} 394 & 491 & 290 & 295 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

De la même manière, les autre matrices s'écrivent :

$$\beta_2 = \begin{pmatrix} 0 & 491 & 290 & 295 \\ 394 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \beta_3 = \begin{pmatrix} 0 & 0 & 290 & 295 \\ 394 & 491 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\beta_4 = \begin{pmatrix} 0 & 0 & 290 & 295 \\ 0 & 491 & 0 & 0 \\ 394 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \beta_5 = \begin{pmatrix} 0 & 0 & 0 & 295 \\ 0 & 0 & 290 & 0 \\ 0 & 491 & 0 & 0 \\ 394 & 0 & 0 & 0 \end{pmatrix}$$

L'avantage d'avoir pu prendre les mêmes valeurs des connexions pour tous les cas est

que le seul paramètre qui change est la place des connexions, mais le span reste identique et n'influence donc pas les résultats des performances d'erreur. Les simulations, présentées sous forme de courbes donnant la probabilité d'erreur par bit en fonction du nombre d'itérations à $E_b/N_0 = 2dB$, sont à la figure 4.13.

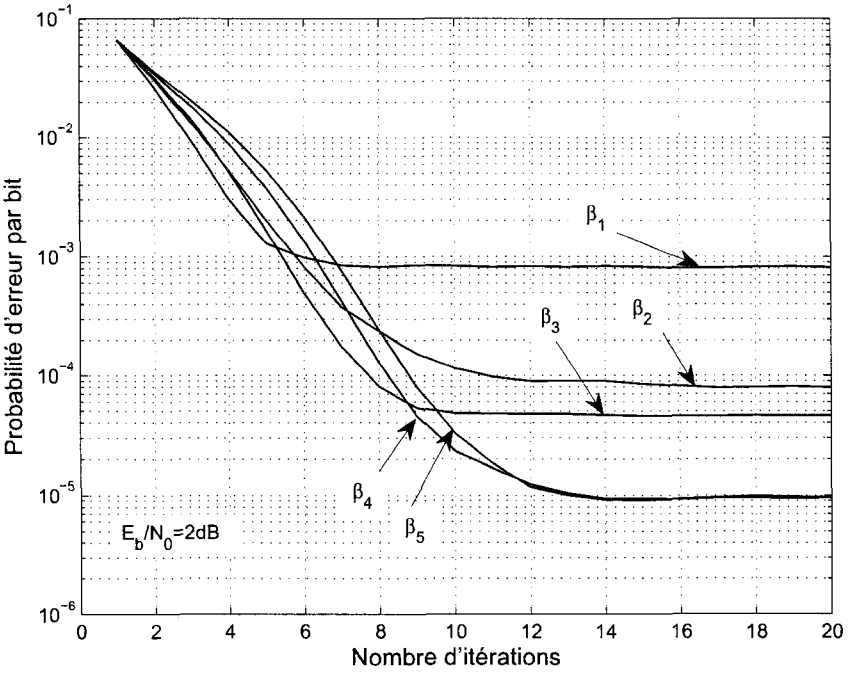

FIG. 4.13 Influence du nombre de connexions par ligne dans la sous matrice β

On peut observer qu'il n'est pas avantageux de privilégier une ligne sur les autres puisque la matrice β_1, qui contient toutes les connexions sur la même ligne, offre les moins bonnes performances d'erreur par bit avec 8×10^{-4}. Les meilleurs résultats sont obtenus avec les deux matrices qui répartissent le plus possible les connexions. En effet, les matrices β_4 et β_5 atteignent une probabilité d'erreur par bit de 9×10^{-6}.

4.3.3.2 Dispositions possibles pour 1 connexion par ligne dans la sous matrice β

Regardons maintenant l'influence de la répartition des connexions sur les colonnes de la partie récursive de la matrice. L'ensemble des connexions précédentes sont toujours valides pour les 4 matrices suivantes :

$$\beta_6 = \begin{pmatrix} 0 & 0 & 295 & 0 \\ 0 & 290 & 0 & 0 \\ 491 & 0 & 0 & 0 \\ 394 & 0 & 0 & 0 \end{pmatrix} \quad \beta_7 = \begin{pmatrix} 0 & 295 & 0 & 0 \\ 0 & 290 & 0 & 0 \\ 491 & 0 & 0 & 0 \\ 394 & 0 & 0 & 0 \end{pmatrix}$$

$$\beta_8 = \begin{pmatrix} 0 & 295 & 0 & 0 \\ 290 & 0 & 0 & 0 \\ 491 & 0 & 0 & 0 \\ 394 & 0 & 0 & 0 \end{pmatrix} \quad \beta_9 = \begin{pmatrix} 295 & 0 & 0 & 0 \\ 290 & 0 & 0 & 0 \\ 491 & 0 & 0 & 0 \\ 394 & 0 & 0 & 0 \end{pmatrix}$$

Avec β_5, ces quatres dispositions sont toutes telles que les matrices ont une connexion par ligne et la répartition sur les colonnes suit le tableau 4.9.

Nb connex / β_i	4 connex	3 connex	2 connex	1 connex	0 connex
β_9	1	0	0	0	3
β_8	0	1	0	1	2
β_7	0	0	2	0	2
β_6	0	0	1	2	1
β_5	0	0	0	4	0

TAB. 4.9 Répartition des colonnes en fonction du nombre de leurs connexions

Comme dans la partie précédente, nous simulons le comportement de ces différents codes à $E_b/N_0 = 2dB$. Les résultats sont donnés à la figure 4.14.

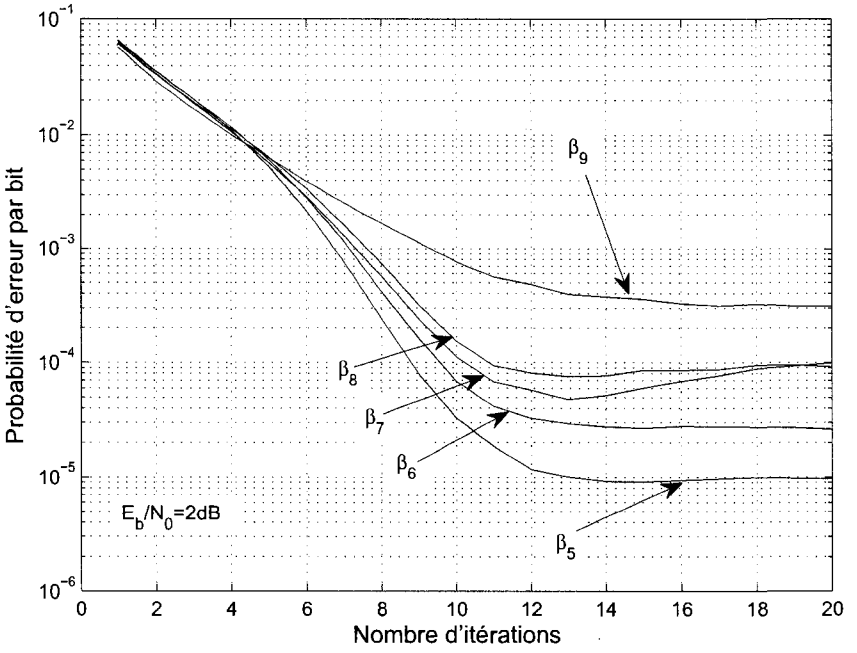

FIG. 4.14 Influence du nombre de connexions par colonne dans la sous matrice β

Les résultats obtenus indiquent à nouveau que le codeur représenté par la matrice $\boldsymbol{\beta 5}$ est le plus performant avec une probabilité d'erreur par bit de 9×10^{-6}. Cette observation est aussi plus importante que pour la répartition des lignes. En effet, les performances sont sensibles au nombre de connexions par colonne. La matrice $\boldsymbol{\beta_6}$, qui n'a qu'une colonne sans connexion, donne une probabilité d'erreur par bit de 2.5×10^{-5} contrairement à la matrice $\boldsymbol{\beta_4}$, qui, avec une seule ligne sans connexion, nous donnait aussi 9×10^{-6}.

Les simulations présentées dans cette partie nous poussent à croire qu'il est important d'avoir un nombre constant de connexions sur les colonnes et sur les lignes de la partie

récursive des matrices de représentation. Mais il paraît possible d'avoir une exception au niveau de la répartition des lignes. Cette propriété est importante car elle peut nous permettre de donner plus d'importance à d'autres conditions, comme avoir le plus grand span possible.

4.3.3.3 Nombre de connexions par colonne dans la sous matrice β

Suite aux observations précédentes, nous avons retiré certaines connexions aux matrices $\boldsymbol{H3}$ et $\boldsymbol{H4}$, en gardant un nombre constant de connexions par colonne. Dans le cas où on garde 1 connexion par colonne dans la sous matrice β, les matrices simulées sont :

$$\boldsymbol{H3_1} = \left(\begin{array}{ccc} 1 & 12 & 85 \\ 106 & 63 & 8 \\ 16 & 61 & 78 \\ \hline 0 & 0 & 103 \\ 0 & 119 & 0 \\ 111 & 0 & 0 \end{array}\right) \text{ et } \boldsymbol{H4_1} = \left(\begin{array}{ccc} 193 & 101 & 96 \\ 26 & 0 & 97 \\ 49 & 44 & 20 \\ \hline 0 & 195 & 0 \\ 135 & 0 & 0 \\ 0 & 0 & 151 \end{array}\right)$$

Les connexions qui ont été retirées sont les plus courtes. Les performances d'erreur sont données aux figures 4.15 et 4.16.

En comparaison avec les simulations des codes des figures 4.7 et 4.8, les résultats obtenus en enlevant deux connexions par colonne sont moins bons. En effet, le seul avantage de ces codes est le faible nombre d'itérations nécessaires au décodage. Le code de plus court

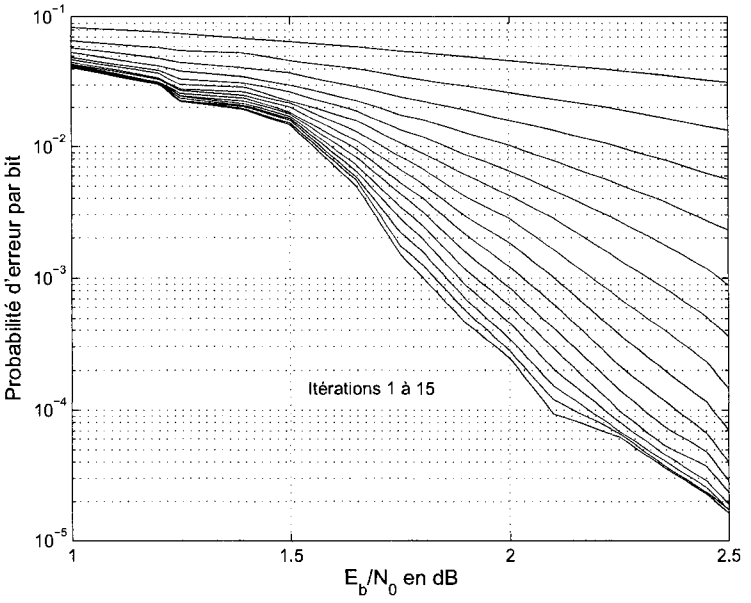

FIG. 4.15 Simulation du code $R - CSO^2C - SS$ avec R=1/2, J=3 et m=119 représenté par la matrice $\boldsymbol{H3}_1$

span a seulement besoin de 15 itérations pour converger, et l'autre code d'environ 20. Et si on se place à $E_b/N_0 = 1.6dB$, après 10 itérations, le code de longueur 119, avec une seule connexion par colonne dans la partie récursive, atteint une probabilité d'erreur par bit de 1×10^{-2}, alors que le code d'origine, avec toutes les connexions, atteignait une probabilité de seulement 2×10^{-2}. On en conclut qu'après 10 itérations, c'est le code avec le moins de connexions qui est le plus performant. Cependant, le code de la matrice $\boldsymbol{H3}_1$ n'améliore pas autant ses performances en effectuant plus d'itérations que le code de la matrice $\boldsymbol{H3}$ qui atteint, toujours à $E_b/N_0 = 1.6dB$, une probabilité d'erreur par bit de 7×10^{-5} après 35 itérations. On peut noter que ces observations sont valables dans les 2 cas étudiés.

Il est en revanche intéressant de comparer ces codes, pour J=3, avec une seule connexion

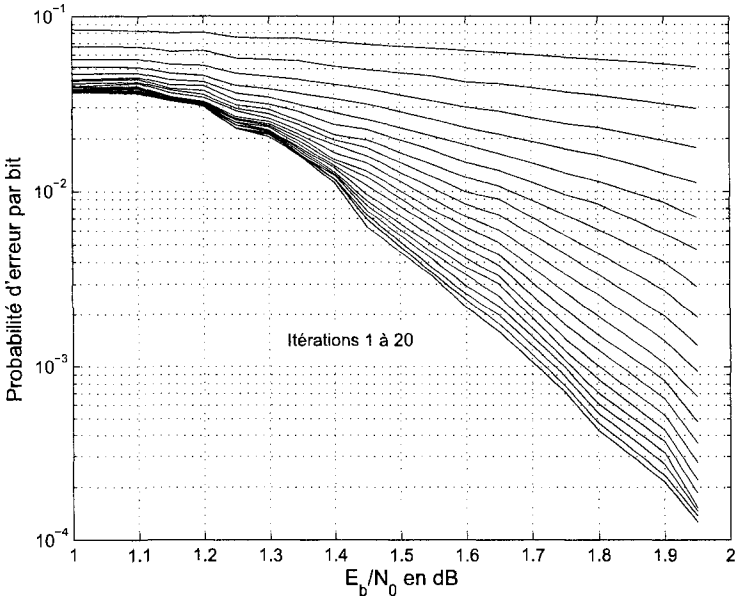

FIG. 4.16 Simulation du code $R - CSO^2C - SS$ avec R=1/2, J=3 et m=195 représenté par la matrice $\boldsymbol{H4_1}$

par ligne/colonne dans la sous matrice β, avec le code de span 10 représenté par la matrice $\boldsymbol{H5}$ qui a lui aussi une seule connexion par ligne/colonne dans la sous matrice β. Pour une valeur de E_b/N_0 de 2dB, le code de span 10 atteignait une probabilité d'erreur par bit de 5×10^{-3}. Les deux codes de cette partie, eux, donnent une probabilité d'erreur par bit de l'ordre de 1×10^{-4}. L'importance du span est ici encore plus prononcée que ce qu'il avait été observé à la section 4.3.1.

En remettant une connexion par colonne selon les matrices $\boldsymbol{H3_2}$ et $\boldsymbol{H4_2}$, nous obtenons

deux nouveaux codes dont les simulations donnent les figures 4.17 et 4.18.

$$\boldsymbol{H3_2} = \left(\begin{array}{ccc} 1 & 12 & 85 \\ 106 & 63 & 8 \\ 16 & 61 & 78 \\ \hline 0 & 115 & 103 \\ 112 & 119 & 40 \\ 111 & 0 & 0 \end{array}\right) \text{ et } \boldsymbol{H4_2} = \left(\begin{array}{ccc} 193 & 101 & 96 \\ 26 & 0 & 97 \\ 49 & 44 & 20 \\ \hline 182 & 195 & 155 \\ 135 & 172 & 0 \\ 0 & 0 & 151 \end{array}\right)$$

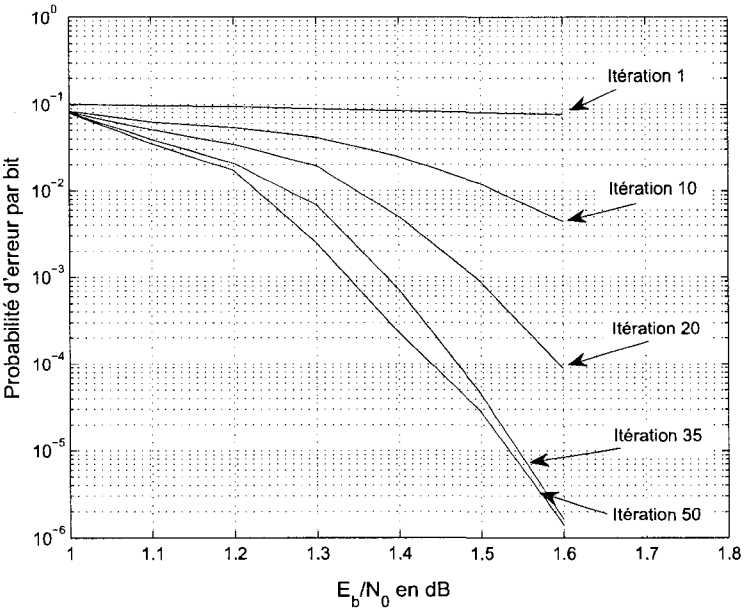

FIG. 4.17 Simulation du code $R - CSO^2C - SS$ avec R=1/2, J=3 et m=119 représenté par la matrice $\boldsymbol{H3_2}$

Cette fois, les codes avec deux connexions par colonne sont plus performants que les codes avec les matrices pleines. En effet, pour les codes de spans 119, avec $E_b/N_0 = 1.5dB$

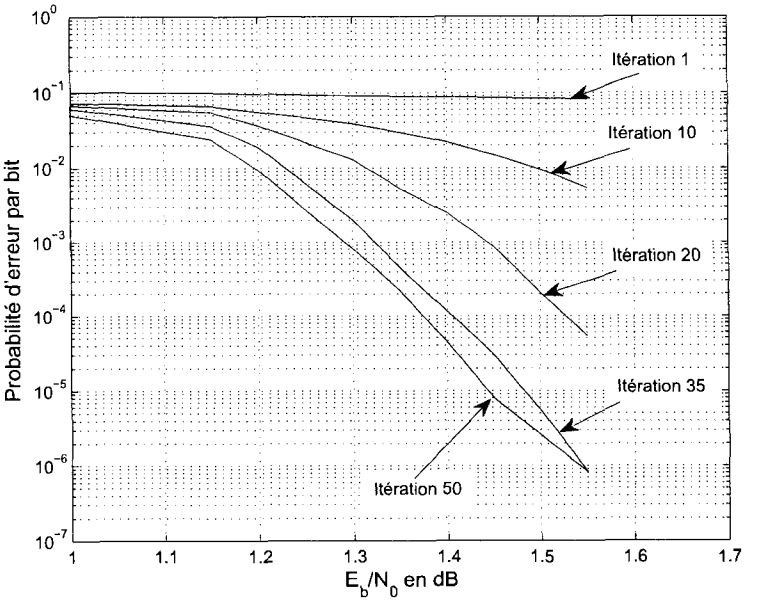

FIG. 4.18 Simulation du code $R - CSO^2C - SS$ avec R=1/2, J=3 et m=195 représenté par la matrice $\boldsymbol{H4_2}$

et après 50 itérations au décodage, le code avec la matrice pleine obtenait une probabilité d'erreur par bit de 2×10^{-4}, contre 3×10^{-5} pour le code avec 2 connexions par colonne dans la partie récursive. De la même manière, pour les codes avec un span de 195, on avait une probabilité de 6×10^{-5} avec la matrice pleine, contre 2×10^{-6} en enlevant une connexion par colonne. De plus, nous pouvons comparer les performances des deux codes représentés par les matrices $\boldsymbol{H3_2}$ et $\boldsymbol{H4_2}$ avec le code de span 33 représenté par la matrice $\boldsymbol{H6}$. En effet, ces trois codes ont deux connexions par colonne dans leurs sous matrices $\boldsymbol{\beta}$ respectives. Cependant, pour $E_b/N_0 = 1.4dB$, le code de span 195 atteint une probabilité d'erreur par bit d'environ 4×10^{-5}, contre 2×10^{-4} pour le code de span 119 et 1×10^{-2} pour le code de span 33. La relation entre les performances d'erreur et le span des codes

est donc une fois de plus mise en évidence.

Ces résultats confirment ce qui avait été observé dans [4] puisque les meilleures performances sont obtenues avec des matrices dont la partie récursive n'est pas pleine. Cependant, il est difficile de généraliser une propriété sur le nombre idéal de connexions à utiliser à partir de nos simulations.

4.4 Conclusion

Le travail de recherche résumé dans ce chapitre a permis de trouver des conditions spécifiques aux codes $R - CSO^2C - SS$. Grâce à elles, il a été possible de construire de nouveaux codes, en jouant sur certains paramètres, comme le span maximal ou le nombre total de connexions pour un J donné.

Les résultats de simulation de ces nouveaux codes sont extrêmement intéressants puisqu'ils montrent que les codes $R - CSO^2C - SS$ sont les codes doublement orthogonaux les plus performants, que ce soit à de hauts rapports signal sur bruit, en se servant de codes avec des petits spans, ou à de faibles rapports signal sur bruit, en prenant des codes plus grands.

Ces codes $R - CSO^2C - SS$ sont aussi les seuls codes doublement orthogonaux dont les performances d'erreur ne dépendent pas seulement du nombre de connexions qu'on utilise. En effet, les spans des codes $R - CSO^2C - SS$ jouent un rôle majeur dans les performances d'erreur. Cependant, il n'est pas intéressant d'augmenter le span de façon irréfléchie. En effet, il a été observé que plus le span augmente et plus le nombre d'itérations

nécessaires au décodage augmente aussi. Or, la latence est proportionnelle à la fois au span et au nombre d'itérations utilisées. Les deux cas extrêmes présentés dans ce mémoire concernent le code, pour J=5, de span 1135 et le code, pour J=3, de span 10. Pour arriver à la convergence des deux codes, nous avons utilisé 50 itérations pour le code le plus grand, soit un total de 283750 bits gardés en mémoire, contre 15 pour le plus petit, et donc 450 bits à garder en mémoire. Les performances d'erreur de ces deux codes sont aussi complètement différentes. En effet, le code de span 1135, avec J=5, atteint une probabilité d'erreur par bit de 1×10^{-4} pour $Eb/N_0 = 1.2dB$ alors que le code de span 10 atteint cette probabilité pour $Eb/N_0 = 3dB$ seulement. Des compromis entre la latence et les performances d'erreur seront donc à faire.

CHAPITRE 5

CONCLUSION

5.1 Bilan de la recherche effectuée

Ce travail de recherche, qui est la suite de tous les travaux sur les codes convolutionnels doublement orthogonaux déjà entrepris, a permis d'explorer la dernière grande classe des codes convolutionnels doublement orthogonaux, les codes convolutionnels doublement orthogonaux récursifs.

Notre étude a ainsi permis de définir de manière spécifique les codes $R - CSO^2C$, aux sens large et strict, en développant, dans chacun des cas, l'algorithme de décodage itératif à la deuxième itération.

Grâce à ces nouvelles conditions, il a été possible de construire de nouveaux codes selon différentes méthodes. En effet, tous les vecteurs générateurs utilisés jusqu'à ce jour pour les codes $R - CSO^2C$ étaient, pour les codes au sens large, issus de codes $CSO^2C - WS$ qu'on avait divisés en deux, et, pour les codes au sens strict, issus de codes $CSO^2C - SS$ de taux de codage $R = 2J/3J$. Ainsi, nous avons pu construire de nombreux codes qui se sont avérés, pour un J donné, de plus courts spans que les codes convolutionnels doublement orthogonaux non récursifs. Une recherche exhaustive des codes $R - CSO^2C - WS$ a notamment permis d'obtenir les codes de plus courts spans jusque J=7, sachant que les performances d'erreur de ces codes au sens large ne dépendent que du nombre J de

connexions.

Au niveau des performances d'erreur obtenues par simulations, les codes $R-CSO^2C-WS$ ne sont pas très intéressants en dessous de $E_b/N_0 = 3dB$. De manière générale, ils sont même surclassés par les codes non récursifs et en particulier par les codes simplifiés $S-CSO^2C-WS$.

En revanche, pour les codes $R-CSO^2C-SS$,les simulations nous indiquent que les performances d'erreur ne dépendent plus seulement du nombre de connexions utilisées, ce qui avait toujours été le cas pour tous les codes convolutionnels doublement orthogonaux. En effet, les probabilités d'erreur par bit obtenues sont aussi influencées par les spans des codeurs ainsi que par la répartition de leurs connexions dans leurs sous matrices β.

Cependant, les codes $R-CSO^2C-SS$ sont les codes convolutionnels doublement orthogonaux les plus performants, que ce soit pour des valeurs de E_b/N_0 supérieures à 2dB, avec des codes de petits spans, ou pour de plus petites valeurs de E_b/N_0, avec des codes de plus grandes longueurs. Mais pour obtenir les meilleures performances d'erreur possibles, il faut se servir de codes avec une importante mémoire et qui nécessitent un grand nombre d'itérations au décodage (>50). Ainsi, la latence engendrée par ces codes devient rapidement très grande puisqu'elle est proportionnelle à la fois au span du code simulé et au nombre d'itérations utilisées au décodage. Il est donc nécessaire de faire un compromis entre les performances d'erreur qu'on cherche à atteindre et la longueur des codes utilisés.

5.2 Améliorations envisageables

Tout d'abord, il serait possible d'améliorer les algorithmes de construction des codes convolutionnels récursifs doublement orthogonaux, et en particulier l'algorithme exhaustif de recherche des codes $R - CSO^2C - WS$, pour trouver les codes de plus courts spans pour $J > 7$. Il devrait aussi être possible d'améliorer l'algorithme de conception des codes $R - CSO^2C - SS$. Une idée serait de chercher des codeurs de taille J à partir de codeurs de taille J-1.

Ensuite, il serait intéressant de trouver la raison de la dépendance des performances des codes $R - CSO^2C - SS$ avec leurs spans respectifs. Une étude sur les cycles engendrés par les codes $R - CSO^2C - SS$ pourrait peut-être expliquer le fait que les codes de plus grands spans sont plus performants.

Enfin, pour les codes $R - CSO^2C - SS$, une étude plus approfondie sur la répartition des connexions dans la sous matrice β, pourrait permettre de généraliser une règle pour savoir le nombre idéal de connexions à utiliser.

5.3 Ouverture

Les codes doublement orthogonaux récursifs sont, en général, plus courts que les codes non récursifs et non simplifiés. Ainsi, il devrait être possible de relaxer certaines des conditions trouvées pour obtenir des codes convolutionnels doublement orthogonaux récursifs simplifiés. Ceci pourrait permettre, pour les codes récursifs au sens large, de réduire le span des codes, pour un J donné, sans trop dégrader les performances d'erreur. Pour les

codes récursifs au sens strict, la simplification des conditions, et donc la réduction du span, devrait aussi permettre de réduire le nombre d'itérations au décodage. Á nouveau, les performances d'erreur ne devraient pas trop s'en ressentir.

La dernière idée serait enfin de considérer des matrices creuses, en particulier en retirant des connexions dans la sous matrice α. Nous avons dans ce mémoire considéré uniquement des codes dont la partie supérieure des matrices était complète. Cependant, rien n'empêche d'y enlever des connexions. On pourrait ainsi, en gardant un faible nombre de connexions par ligne, augmenter la taille de la matrice.

BIBLIOGRAPHIE

[1] B. Baechler. *Analyse et Détermination de Codes Doublement Orthogonaux pour Décodage Itératif.* Mémoire de maîtrise, École Polytechnique de Montréal, Montréal, Juin 2000.

[2] B. Baechler, D. Haccoun, and F. Gagnon. *On the Search for Self-Doubly Orthogonal Codes.* Proceedings 2000 IEEE International Symposium on Information Theory, page 292, Sorrento Palace Hotel, Sorrento, Italie, Juin 2000.

[3] C. Berrou, A. Glavieux, and P. Thitimajshima. *Near Shannon Limit Error-Correcting Coding and Decoding : Turbo-Codes.* ICC, 1993 Symposium, vol. 2 : 1064-1070, Genève, Mai 1993.

[4] C. Cardinal. *Décodage à Seuil Itératif des Codes Convolutionnels Doublement Orthogonaux.* Thèse de doctorat, École Polytechnique de Montréal, Montréal, Juin 2001.

[5] C. Cardinal, D. Haccoun, and F. Gagnon. *Iterative Threshold Decoding Without Interleaving for Convolutional Self-Doubly Orthogonal Codes.* IEEE Transactions on Communications, IT-51 :1274-1282, Août 2003.

[6] C. Cardinal, D. Haccoun, F. Gagnon, and N. Batani. *Convolutional Self-Douby Orthogonal Codes for Iterative Decoding Without Interleaving.* Proceedings 1998 IEEE International Symposium on Information Theory, page 280, MIT, Cambridge, Mass. USA, Août 1998.

[7] C. Cardinal, D. Haccoun, F. Gagnon, and N. Batani. *Turbo Decoding Using Convolutional Self Doubly Orthogonal Codes*. International Conference on Communications, ICC'99, pages 113-117, Vancouver, Colombie Britannique, Canada, Juin 1999.

[8] P. Elias. *Coding for Noisy Channels*. IRE Conv. Rec., pt. 4 :37-46, 1955.

[9] M. Fossorier, M. Mihaljević, and H. Imai. *Reduced Complexity Iterative Decoding of Low-Density Parity Check Codes Based on Belief Propagation*. IEEE, Transactions on Communications, COM-47 :673-680, Mai 1999.

[10] F. Gagnon, N. Batani, and T. Q. Dam. *Simplified Designs for AAPP Soft Decision Threshold Decoders*. IEEE Transactions on Communications, COM-43, No 2/3/4 : 743-750, Février, Mars, Avril 1995.

[11] F. Gagnon, D. Haccoun, N. Batani, and C. Cardinal. *Apparatus for Convolutional Self-Doubly Orthogonal Encoding and Decoding*. US patent No 6167552, 26 Décembre 2000.

[12] R. G. Gallager. *Low-Density Parity-Check Codes*. IRE Transactions on Information Theory, vol. 8 :21-28, Janvier 1962.

[13] D. Haccoun. *ELE 6703 - Théorie des communications*. Presses internationales Polytechnique, École Polytechnique de Montréal, 2005.

[14] D. Haccoun, C. Cardinal, and F. Gagnon. *Search and Determination of Convolutional Self-Doubly Orthogonal Codes for Iterative Threshold Decoding*. IEEE Transactions on Communications, IT-53 :802-809, Mai 2005.

[15] J. L. Massey. *Threshold Decoding*. MIT Press, Cambridge, MA, 1963.

[16] E. Roy. *Recherche et analyse de codes convolutionnels doublement orthogonaux simplifié au sens large*. Mémoire de maîtrise, École Polytechnique de Montréal, Montréal, Août 2006.

[17] C. E. Shannon. *A Mathematical Theory of Communication*. Bell System Technical Journal, vol. 27 :379-423, 1948.

[18] J. M. Wozencraft and I. M. Jacobs. *Principles of Communication Engineering*. Wiley, New York, 1965.

[19] Y. Zerong. *Codes convolutionnels récursifs systématiques*. Mémoire de maîtrise, École Polytechnique de Montréal, Montréal, Avril 1998.

ANNEXE I

Calcul de l'estimation λ_i pour le décodage à seuil

I.1 Rappel du problème

Nous sommes ici dans le cas des codes convolutionnels simplement orthogonaux non récursifs. On considère un code $CSOC$ représenté par le vecteur des J connexions :$\{\alpha_1, \ldots, \alpha_J\}$. Nous cherchons à effectuer, au niveau du décodage, l'estimation du bit u_i à partir des symboles reçus du canal. Dans le cas du canal non quantifié, ces symboles sont $\{\tilde{u}_i, \ldots, \tilde{u}_{i+\alpha_J}\}$ et $\{\tilde{p}_i, \ldots, \tilde{p}_{i+\alpha_J}\}$. Le décodage se base sur le calcul de J symboles de syndrome définis ici bas, avec $k = 1, \ldots, J$:

$$s_{i+\alpha_k} = \tilde{p}_{i+\alpha_k} \oplus \sum_{j=1}^{J} \oplus \tilde{u}_{i+\alpha_k-\alpha_j} \tag{I.1}$$

A partir de l'équation I.1, nous avons défini dans le premier chapitre un système de $J+1$ équations orthogonles au bit d'information u_i valant, pour $k = 1, \ldots, J$:

$$\begin{aligned} B_{0,i} &= \tilde{u}_i \\ B_{k,i} &= \tilde{u}_i \oplus s_{i+\alpha_k} \end{aligned} \tag{I.2}$$

Enfin, le développement des équations présentées en I.2 donne le résultat donné à

l'équation 1.18 et rappelé ici :

$$\begin{aligned} B_{0,i} &= u_i \oplus e_i^u \\ B_{k,i} &= u_i \oplus e_{i+\alpha_k}^p \oplus \sum_{j=1}^{k-1} \oplus e_{i+\alpha_k-\alpha_j}^u \oplus \sum_{j=k+1}^{J} \oplus e_{i+\alpha_k-\alpha_j}^u \end{aligned} \tag{I.3}$$

Cependant, tous les termes de ces équations sont binaires alors que nous souhaitons effectuer un décodage à seuil sans quantification. Nous allons donc maintenant développer les étapes du raisonnement menant à l'estimation, qui sera réelle et non plus binaire, λ_i, du bit d'information u_i.

I.2 Calcul du logarithme du rapport de vraisemblance

Le décodeur se sert des $J+1$ équations $B_{k,i}$ pour effectuer une décision sur la valeur à donner au bit d'information u_i. Ainsi, il assigne au bit u_i les deux valeurs binaires $\xi = \{0,1\}$ et maximise la probabilité conditionnelle $Prob\{u_i = \xi | \{B_{k,i}\}\}$. Ainsi, il décidera $\hat{u}_i = 1$ si et seulement si :

$$Prob\{u_i = 1 | \{B_{k,i}\}\} \geq Prob\{u_i = 0 | \{B_{k,i}\}\} \tag{I.4}$$

sinon, il decidera $\hat{u}_i = 0$.

Le logarithme du rapport de vraisemblance (LRV) s'obtient en prenant le logarithme

du rapport des deux termes de l'éqaution I.4. Il est noté $L(u_i|\{B_{k,i}\})$ et vaut :

$$L(u_i|\{B_{k,i}\}) = ln\left(\frac{Prob\{u_i = 1|\{B_{k,i}\}\}}{Prob\{u_i = 0|\{B_{k,i}\}\}}\right) \tag{I.5}$$

La règle de décision utilisée par le décodeur algébrique à seuil devient donc : *Décider* $\hat{u}_i = 1$ *si et seulement si :*

$$L(u_i|\{B_{k,i}\}) \geq 0 \tag{I.6}$$

sinon, décider $\hat{u}_i = 0$. Les symboles d'erreur intervenant dans les équations I.3 sont indépendants les uns des autres du fait de l'orthogonalité des équations $B_{k,i}$. On peut donc appliquer le théorème de Bayes au LRV donné à la relation I.5 pour obtenir, en supposant que les probabilités *a priori* sont égales, soit $Prob\{u_i = 1\} = Prob\{u_i = 0\}$:

$$L(u_i|\{B_{k,i}\}) = \sum_{k=1}^{J} ln\left(\frac{Prob\{B_{k,i}|u_i = 1\}}{Prob\{B_{k,i}|u_i = 0\}}\right) + ln\left(\frac{Prob\{B_{0,i}|u_i = 1\}}{Prob\{B_{0,i}|u_i = 0\}}\right) \tag{I.7}$$

I.3 Calcul des poids de Massey

Nous introduisons maintenant la probabilité $\gamma_{k,i} = (1-\rho_{k,i})$, pour $k = 1, \ldots, J$, d'avoir un nombre impair de symboles d'erreur binaires ayant pour valeur "1" dans l'équation $B_{k,i}$. Il est clair que $\rho_{k,i}$ représente la probabilité d'en avoir un nombre pair.

Cependant, pour $k = 1, \ldots, J$, la probabilité que l'équation $B_{k,i}$ prenne la valeur "1" sachant que le bit d'information u_i vaut "0" est égale à la probabilité qu'il y ait un nombre impair de symboles d'erreur ayant la valeur "1", soit $\gamma_{k,i}$. De la même manière, la probabi-

lité que l'équation $B_{k,i}$ prenne la valeur "1" sachant que le bit d'information u_i vaut "1" est égale à $\rho_{k,i}$. On obtient ainsi, pour $k = 1, \ldots, J$:

$$Prob\{B_{k,i} = 0|u_i = 1\} = Prob\{B_{k,i} = 1|u_i = 0\} = \gamma_{k,i} \tag{I.8}$$

$$Prob\{B_{k,i} = 0|u_i = 0\} = Prob\{B_{k,i} = 1|u_i = 1\} = \rho_{k,i} \tag{I.9}$$

Les poids définis par Massey [15], notés $w_{k,i}$, sont donnés par $w_{k,i} = ln\left(\frac{\rho_{k,i}}{\gamma_{k,i}}\right)$. Il a alors été démontré, dans [15], qu'à partir des équations I.8 et I.9, la relation I.7 devient :

$$L(u_i|\{B_{k,i}\}) = \sum_{k=1}^{J} (2B_{k,i} - 1)w_{k,i} + (2B_{0,i} - 1)w_{0,i} \tag{I.10}$$

I.4 Calcul de l'estimation λ_i du LRV $L(u_i|\{B_{k,i}\})$

En observant les équations I.8 et I.9, il est possible de les reformuler selon :

$$\gamma_{k,i} = Prob\{B_{k,i} \oplus u_i = 1\} \tag{I.11}$$

$$\rho_{k,i} = Prob\{B_{k,i} \oplus u_i = 0\} \tag{I.12}$$

On a donc, pour $k = 1, \ldots, J$:

$$w_{k,i} = ln\left(\frac{\rho_{k,i}}{\gamma_{k,i}}\right) = -ln\left(\frac{Prob\{B_{k,i} \oplus u_i = 0\}}{Prob\{B_{k,i} \oplus u_i = 1\}}\right) = -L(B_{k,i} \oplus u_i) \tag{I.13}$$

Et pour $w_{0,i}$, on a

$$w_{0,i} = ln\left(\frac{\rho_{0,i}}{\gamma_{0,i}}\right) = -L(B_{0,i} \oplus u_i) \tag{I.14}$$

On peut en déduire, en réinjectant l'équation I.3 dans les équations I.13 et I.14, que pour $k = 1, \ldots, J$:

$$w_{k,i} = -L(e^p_{i+\alpha_k} \oplus \sum_{j=1}^{k-1} \oplus e^u_{i+\alpha_k-\alpha_j} \oplus \sum_{j=k+1}^{J} \oplus e^u_{i+\alpha_k-\alpha_j}) \tag{I.15}$$

et

$$w_{0,i} = -L(e^u_i) \tag{I.16}$$

Cependant, il a été prouvé dans [4] qu'une approximation du LRV de la somme modulo-2 de variables aléatoires indépendantes $\{\xi_1, \ldots, \xi_N\}$ s'obtient selon :

$$L\left(\sum_{n=1}^{N} \oplus \xi_n\right) \approx \sum_{n=1}^{N} \diamond L(\xi_n) \tag{I.17}$$

où l'opérateur *add-min*, représenté par le symbole $\diamond$, consiste en l'opération suivante :

$$L(\xi_1) \diamond L(\xi_2) = -sign(L(\xi_1))sign(L(\xi_2))min(|L(\xi_1)|, |L(\xi_2)|) \tag{I.18}$$

Ainsi, pour $k = 1, \ldots, J$, le poids $w_{k,i}$, dont la valeur vaut le LRV de l'équation $B_{k,i}$

excluant le bit d'information u_i, peut s'écrire d'après I.15 :

$$w_{k,i} \approx -L(e^p_{i+\alpha_k}) \diamond \sum_{j=1}^{k-1} \diamond L(e^u_{i+\alpha_k-\alpha_j}) \diamond \sum_{j=k+1}^{J} \diamond L(e^u_{i+\alpha_k-\alpha_j}) \tag{I.19}$$

Finalement, en reprenant les équations I.10, I.17 et I.19, il a été démontré dans [4] que l'estimation λ_i du LRV $L(u_i|\{B_{k,i}\})$ pouvait s'écrire :

$$L(u_i|\{B_{k,i}\}) \approx \lambda_i \quad = \quad y^u_i + \sum_{k=1}^{J} \left(y^p_{i+\alpha_k} \diamond \sum_{j=1}^{k-1} \diamond y^u_{i+\alpha_k-\alpha_j} \diamond \sum_{j=k+1}^{J} \diamond \lambda_{i+\alpha_k-\alpha_j} \right) \tag{I.20}$$

Enfin, la règle de décision suivie par le décodeur à seuil et définie à l'équation I.6 devient :

$$Dcider\ \hat{u}_i = \begin{cases} 1 & si\ \lambda_i \geq 0 \\ 0 & sinon \end{cases}$$

ANNEXE II

Equivalence de la représentation des codes $R - CSO^2C - WS$

Nous allons montrer ici qu'il revient au même de représenter les codes convolutionnels doublement orthogonaux récursifs au sens large définis par les vecteurs de connexions $\{\alpha_1\ \alpha_2 \cdots \alpha_{J_1}\}$ et $\{\beta_1\ \beta_2 \cdots \beta_{J_2}\}$, respectivement de J_1 et J_2 connexions, par les figures II.1 ou II.2 :

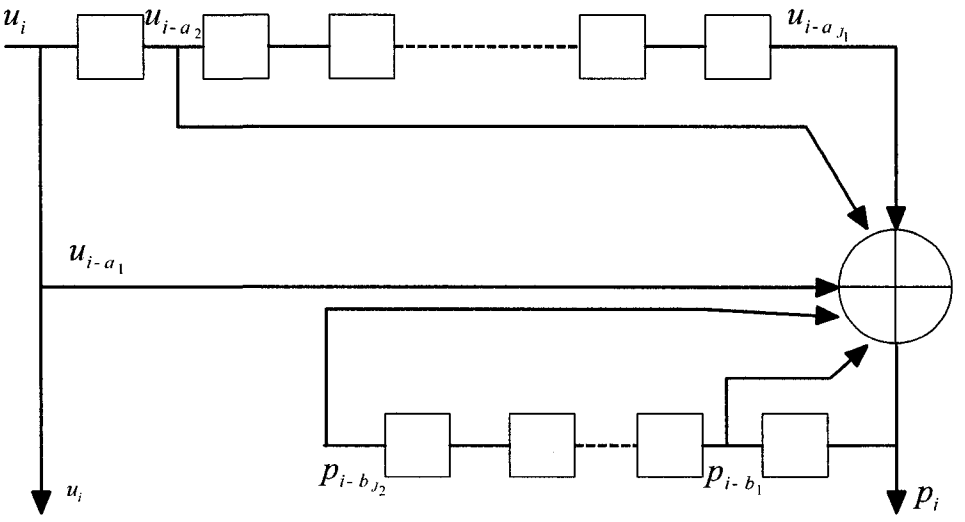

FIG. II.1 Codeur convolutionnel récursif systématique de taux $R = 1/2$

Il est immédiat de dire que le symbole de parité p_i, à l'instant i, obtenu par le codeur de la figure II.1 vaut :

$$p_i = \sum_{j=1}^{J_1} \oplus u_{i-\alpha_j} \oplus \sum_{k=1}^{J_2} \oplus p_{i-\beta_k} \tag{II.1}$$

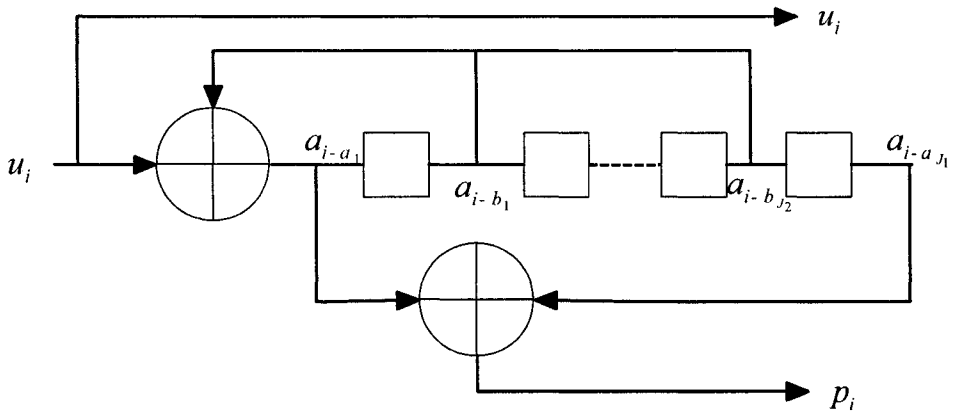

FIG. II.2 Codeur convolutionnel récursif systématique de taux $R = 1/2$

Pour le codeur représenté par la figure II.2, p_i se calcule selon :

$$p_i = \sum_{j=1}^{J_1} \oplus a_{i-\alpha_j} \tag{II.2}$$

Or, ces symboles a_i, représentés sur la figure II.2, sont calculés selon :

$$a_i = u_i \oplus \sum_{k=1}^{J_2} \oplus a_{i-\beta_k} \tag{II.3}$$

d'où en réinjectant II.3 dans II.2 :

$$\begin{aligned} p_i &= \sum_{j=1}^{J_1} \oplus \left(u_{i-\alpha_j} \oplus \sum_{k=1}^{J_2} \oplus a_{i-\alpha_j-\beta_k} \right) \\ &= \sum_{j=1}^{J_1} \oplus u_{i-\alpha_j} \oplus \sum_{k=1}^{J_2} \oplus \left(\sum_{j=1}^{J_1} \oplus a_{i-\beta_k-\alpha_j} \right) \end{aligned} \tag{II.4}$$

Or, nous avons d'après l'équation II.2 :

$$\sum_{j=1}^{J_1} \oplus a_{i-\beta_k-\alpha_j} = p_{i-\beta_k}$$

d'où le résultat :

$$p_i = \sum_{j=1}^{J_1} \oplus u_{i-\alpha_j} \oplus \sum_{k=1}^{J_2} \oplus p_{i-\beta_k} \tag{II.5}$$

Nous pouvons donc conclure que les deux représentations sont identiques.

ANNEXE III

Résultats de simulations des codes $R - CSO^2C - WS$

Cette annexe répertorie toutes les performances d'erreur des codes $R - CSO^2C - WS$ simulés. On rappelle les différents codes dans le tableau III.1.

J	J_1	J_2	gen_bit	gen_par
3	2	1	[0 2]	[3]
4	3	1	[0 2 10]	[7]
5	3	2	[0 1 24]	[4 17]
6	4	2	[0 1 49 53]	[13 42]
7	4	3	[0 1 78 110]	[4 66 91]
8	5	3	[0 3 30 104 296]	[7 35 214]
9	5	4	[0 2 12 103 385]	[3 40 226 539]
10	6	4	[0 3 14 92 455 905]	[10 47 216 595]
11	6	5	[0 3 21 116 417 1146]	[10 50 204 683 1831]

TAB. III.1 Codes $R - CSO^2C - WS$ les plus courts, simulés pour J variant de 3 à 11

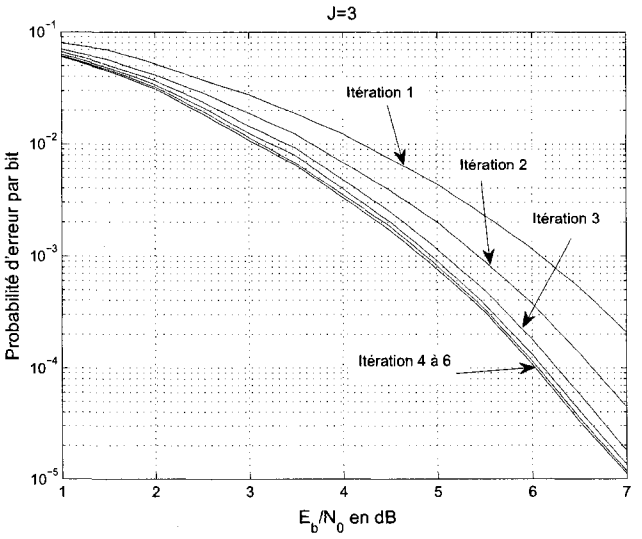

FIG. III.1 Simulation d'un code $R - CSO^2C - WS$ pour R=1/2, J=3 et m=3

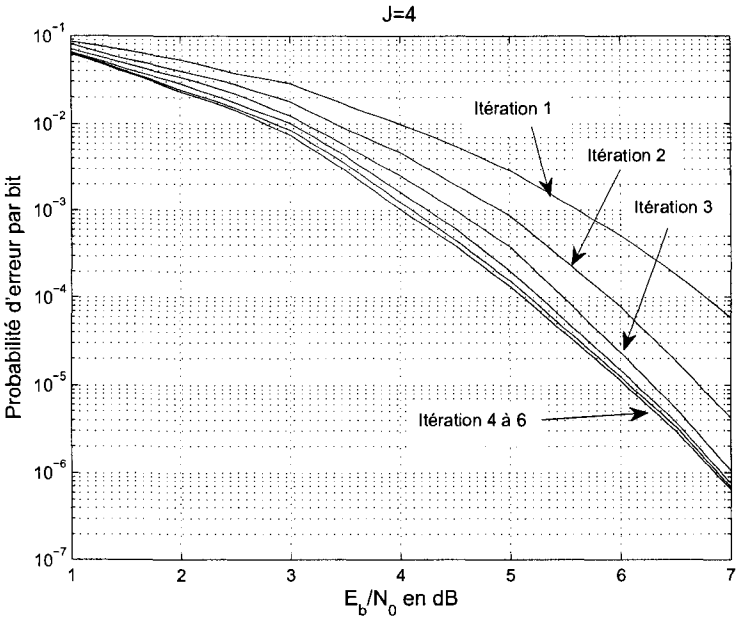

FIG. III.2 Simulation d'un code $R - CSO^2C - WS$ pour R=1/2, J=4 et m=10

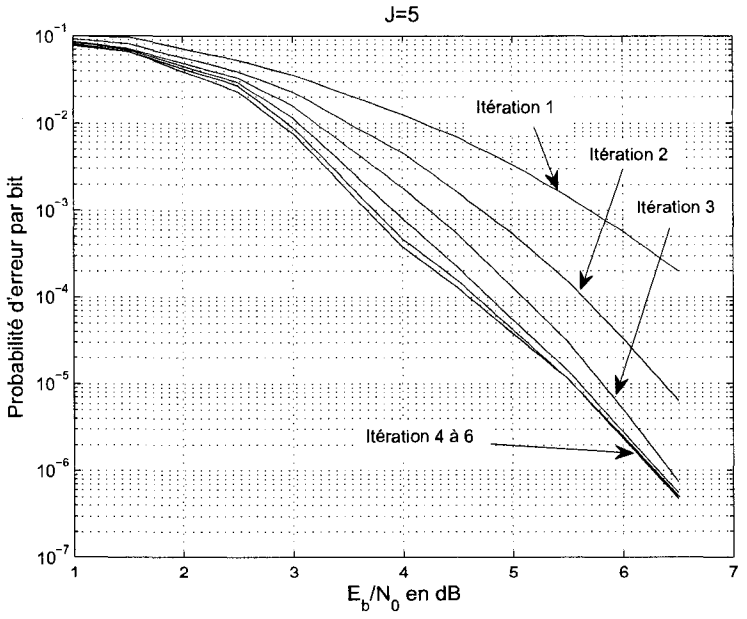

FIG. III.3 Simulation d'un code $R - CSO^2C - WS$ pour R=1/2, J=5 et m=24

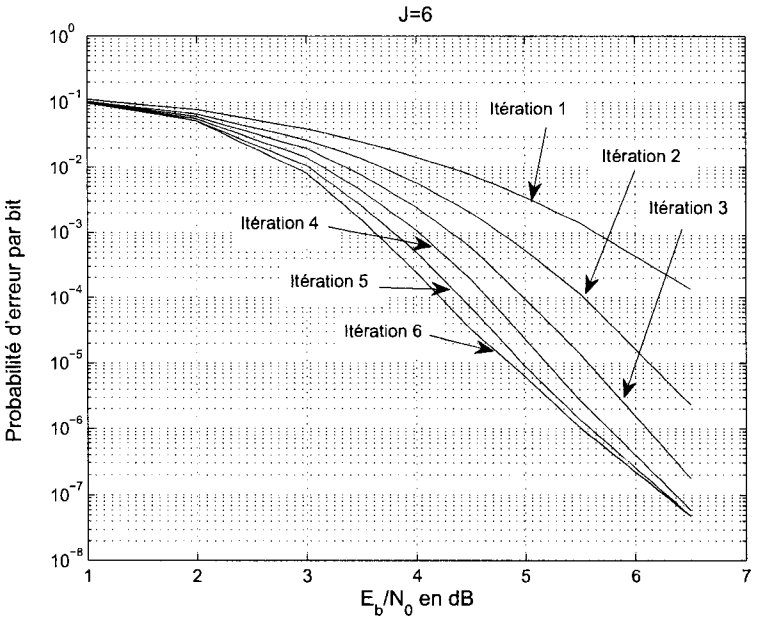

FIG. III.4 Simulation d'un code $R - CSO^2C - WS$ pour R=1/2, J=6 et m=53

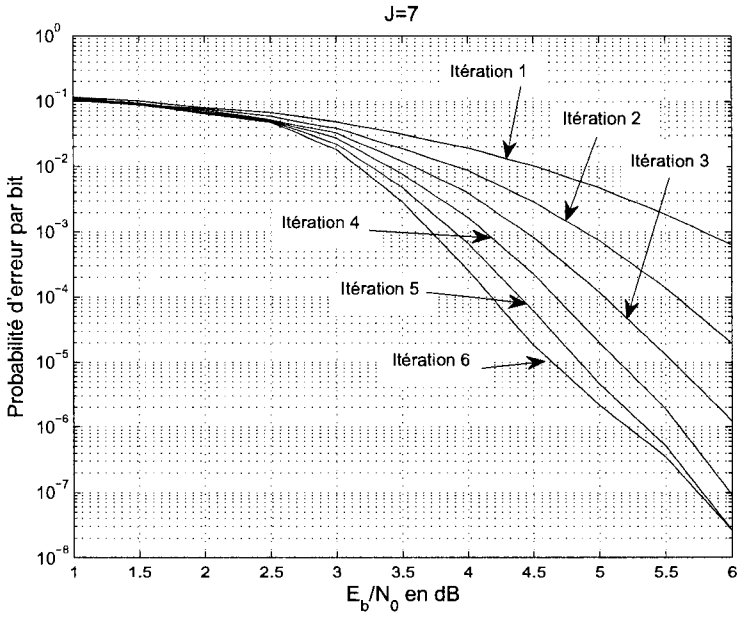

FIG. III.5 Simulation d'un code $R - CSO^2C - WS$ pour R=1/2, J=7 et m=110

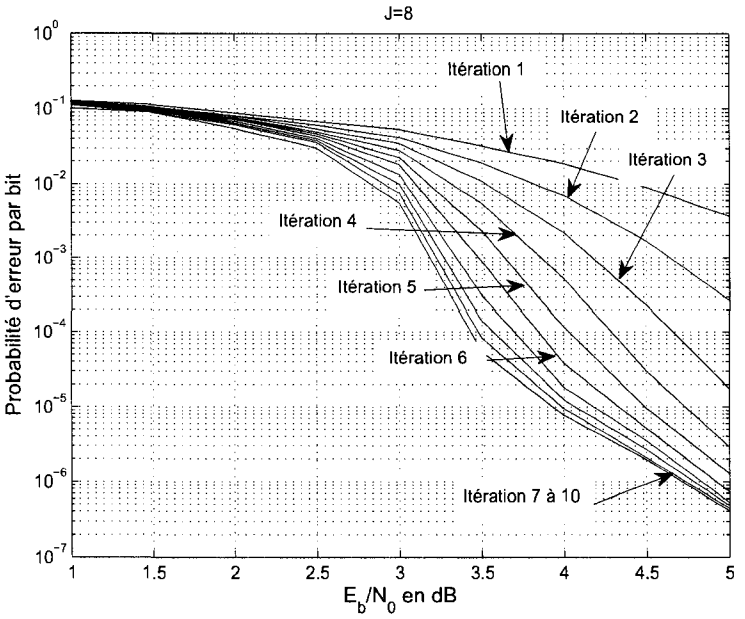

FIG. III.6 Simulation d'un code $R - CSO^2C - WS$ pour R=1/2, J=8 et m=296

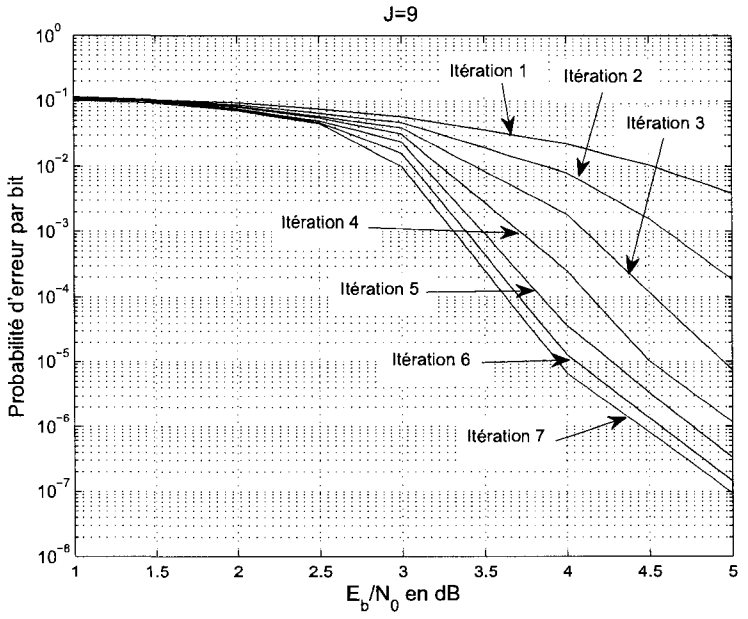

FIG. III.7 Simulation d'un code $R - CSO^2C - WS$ pour R=1/2, J=9 et m=539

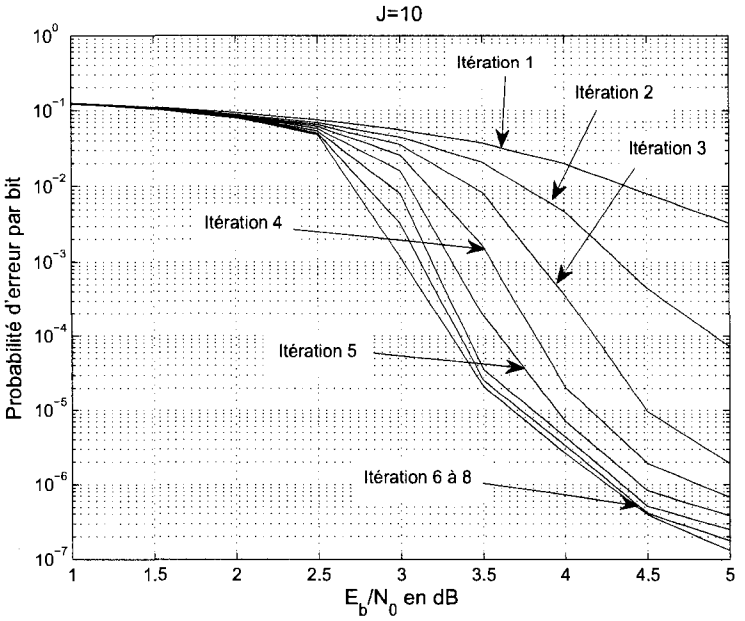

FIG. III.8 Simulation d'un code $R - CSO^2C - WS$ pour R=1/2, J=10 et m=905

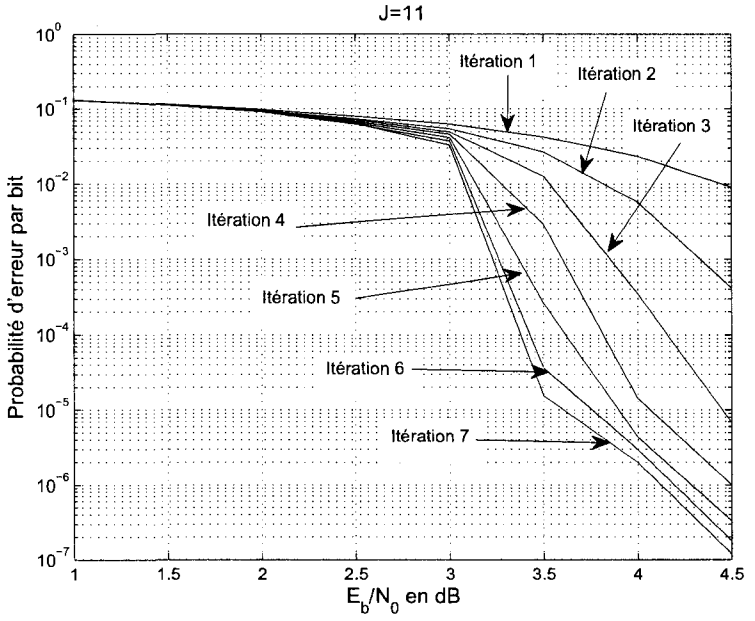

FIG. III.9 Simulation d'un code $R - CSO^2C - WS$ pour R=1/2, J=11 et m=1831