Titre: Title:	Sur le décodage des codes turbo
Auteur: Author:	Afif Osseiran
Date:	1999
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Osseiran, A. (1999). Sur le décodage des codes turbo [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/8623/

UNIVERSITÉ DE MONTRÉAL

SUR LE DÉCODAGE DES CODES TURBO

AFIF HANI OSSEIRAN

DÉPARTEMENT DE GÉNIE ÉLECTRIQUE ET DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE ÉLECTRIQUE)
OCTOBRE 1999

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

SUR LE DÉCODAGE DES CODES TURBO

présenté par: OSSEIRAN Afif Hani

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

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

Mde. JAUMARD Brigitte, Ph.D., président

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

M. GAGNON François, Ph.D., membre

À ma mère.....

Remerciements

Tout d'abord, je voudrais remercier mon directeur de recherche, le professeur David HACCOUN de l'Ecole Polytechnique de Montréal.

Je suis très reconnaissant au professeur Haccoun pour la confiance qu'il m'a accordée pendant mes deux ans en maîtrise, pour son encadrement, et pour son support financier.

Je tiens à remercier aussi respectivement, Guillaume, Pierre Fréderic et Grégory pour la lecture de mon mémoire.

Je n'oublirai pas les collègues de la section de communications, en particulier Mehdi, et Christian Cardinal.

Finalement, j'exprime ma profonde reconnaissance envers mes amis et les membres de ma famille.

Résumé

Ce mémoire porte sur le décodage itératif des Codes dits Turbo qui consistent en une concaténation parallèle de deux codeurs convolutionnels récursifs et systématiques séparés par un entrelaceur.

Le décodage des Codes Turbo souffre d'une très grande complexité. L'algorithme maximum a posteriori (MAP) est l'un des deux algorithmes qui permettent le décodage des Codes Turbo, une variante simplifiée du MAP, qui porte le nom du Log-MAP est utilisée dans cette recherche.

Le premier aspect de ce travail est d'analyser et d'évaluer, par simulation sur ordinateur, les performances d'erreur du décodeur Turbo utilisant l'algorithme Log-MAP pour les longueurs de contrainte $K = 3$ et 5, pour des taux de codage total $R_t = 1/2$ et $1/3$, et pour des blocs de petite à moyenne taille de longueurs $N = 196$ à 3600 bits. Deux types de canaux sont considérés : le canal à bruit blanc additif et gaussien et le canal à évanouissements de type Rayleigh. Dans nos simulations, seule la modulation cohérente BPSK est considérée.

L'inconvénient de l'utilisation du MAP est la necessité de l'estimation de la variance du bruit dans le canal. Ainsi diverses méthodes d'estimation ont été examinées, analysées et comparées. La méthode d'analyse utilisée est basée sur une approche probabiliste.

Le dernier volet de notre travail porte sur le problème de la terminaison des treillis des Codes Turbo qui diffère nettement de celle des codes convolutionnels classique. Plusieurs alternatives de terminaison sont présentées et comparées. Une des alterna-

tives nommée **MNoTail** permet d'éviter la transmission de queues. **MNoTail** est parmi les meilleures techniques de terminaison au niveau de la performance d'erreur tout en présentant l'avantage qu'aucune queue n'est ajoutée à la séquence d'information.

Abstract

This thesis investigates the iterative decoding of Turbo Code which consist in a parallel concatenation of two Recursive and Systematic convolutional Coders (RSC) separated by an interleaver.

The decoding of Turbo Codes suffers from great complexity. The maximum a posteriori (MAP) algorithm is among the two algorithms which allow the decoding of Turbo Codes, a simplified variant of the MAP algorithm, called the Log-MAP, is used in our research.

The first aspect of this work is to evaluate and analyze, throughout computations, the errors performance of the Turbo decoder using the Log-MAP algorithm for the constraint lengths $K = 3$ and 5, the total coding rates $R_t = 1/2$ and $1/3$, and for a medium bloc size of lengths $N = 196$ to 3600 bits. Two types of channel are taken into consideration : an additive white gaussien noise channel and a fading channel based on Rayleigh model. Only the coherent modulation BPSK is considered in our simulations.

The drawback of using the MAP is the necessity to estimate the channel noise. Therefore, different methods for estimating the noise variance were examined and analyzed. The applied analysis is based on a probabilistic approach.

Finally, this work highlights the problem of trellises termination in Turbo Codes which is absolutely different from the one used for classical convolutional codes. Several alternatives for trellis termination are presented and compared. An alternative named **MNoTail** suppresses the necessity of sending tails. **MNoTail** is among the

best techniques of termination regarding the error performance while its advantage is that no tail is added to the information sequence.

Table des matières

Table des figures

Liste des tableaux

Liste des annexes

Liste des sigles et abréviations

ACS	Add-Compare-Select
APP	A Posteriori Probability
AVT	Algorithme de Viterbi Tronqué
AWGN	Additive White Gaussien Noise
BCJR	Bahl, Cocke, Jelinek, and Raviv
BER	Bit Error Rate
BM	Branch Metric
BPSK	Binary Phase Shift Keying
BSM	Backward State Metrics
DM	Delta Modulation
DPCM	Differential Pulse Coded Modulation
FPB	Filtre Passe Bas
FSM	Forwad State Metrics
JP	Joint Probability
LLR	Logarithm Likelihood Ratio
MAP	Maximum A Posteriori
ML	Maximum Likelihood
NRNSC	Non-Recursive Non-Systemaric Coder
PCM	Pulse Coded Modulation
PDF	Fonction de densité de probabilité
RFI	Réponse Finie à une Impusion

RII	Réponse Infinie à une Impulsion
RSC	Recursive Systemaric Coder
SISO	Soft Input Soft Output
SOVA	Soft Outpout Viterbi Algorithm
TC	Turbo Code

Liste des notations

E_b/N_0	Rapport signal sur bruit
K	longueur de contrainte d'un codeur convolutionnel
R_t	Taux de codage global d'un codeur turbo
N	Taille d'un bloc d'information (bits)
L	Longueur d'une queue (bits)
W	largeur de bande (Hertz)
C	Capacité du canal
η	Efficacité Spectrale
$\alpha_k(m)$	Métrique d'état en avant de l'état m à l'instant k
$\beta_k(m)$	Métrique d'état en arrière de l'état m à l'instant k
$\delta_k^i(m)$	Métrique de branche de l'état m à l'instant k
$A_k(m)$	Logarithme de la Métrique d'état en avant de l'état m à l'instant k
$B_k(m)$	Logarithme de la Métrique d'état en arrière de l'état m à l'instant k
$D_k^i(m)$	Logarithme de la Métrique de branche de l'état m à l'instant k
d_k	Bit d'information à l'instant k
σ^2	Variance du bruit dans le canal
M	mémoire du codeur
D	Délai dans un codeur
G(D)	Matrice Génératrice
S_k	l'état du codeur à l'instant k.
P	matrice de perforation

λ Probabilité conjointe

L_a Information a priori

L_e Information extrinsèque

Chapitre 1

Introduction

1.1 Cinquante ans de Shannon aux Codes Turbo

Les premières utilisations des codes correcteurs d'erreur en communication furent dans le domaine spatial (deep space) où l'objectif était de surmonter la perte de propagation dans le vide entre le récepteur (ou transmetteur) de l'engin spatial et le récepteur de la station terrestre. Les vaisseaux ou engins spatiaux envoyés au-delà de l'orbite terrestre devraient être de petite taille ce qui implique une restriction au niveau de la taille des antennes, (aussi des piles, portes etc.), limitant ainsi la puissance de l'émetteur (de l'engin). Le gain de codage offert par les codes correcteurs d'erreur a permis des liaisons spatiales de plus en plus éloignées de la Terre. La largeur de bande était un facteur moins déterminant dans les liaisons spatiales que dans le domaine des communications terrestres (par exemple). L'efficacité d'un code se reflète dans le gain de codage. Le gain de codage est la différence entre un rapport signal à bruit E_b/N_0 requis pour atteindre une certaine performance d'erreur avec un système codé et le E_b/N_0 requis pour atteindre la même performance d'erreur par bit (BER) avec un système non codé.

Nous allons poursuivre avec l'exemple des engins spatiaux où la modulation à modulation binaire de phase BPSK semble être le choix logique pour ce type de

communications du fait de son efficacité au niveau de la puissance par rapport aux autres types de modulation. Par exemple, la modulation BPSK offre un gain de 3 dB par rapport à la modulation binaire de fréquence BFSK [Pro95] [Wic95]. Un système BPSK non codé nécessite un E_b/N_0 de 9.6 dB pour atteindre une BER égale à (10^{-5}). Le standard de codage utilisé par la NASA pour les missions spatiales consistait en une concaténation en série d'un code de Reed-Solomon avec un code convolutionnel [Wic95]. En supposant une modulation BPSK, le standard permet d'atteindre la BER $=10^{-5}$ pour E_b/N_0 égal à 2.2 dB. Ce standard permettant un gain de codage de (9.6-2.2)=7.6 dB.

En diminuant la valeur de E_b/N_0 requise à l'entrée du démodulateur, le code contrôle (ou correcteur) d'erreur qui est appelé aussi codage de canal permet d'assouplir les exigences vis à vis des autres paramètres de la liaison de communication. Par exemple, les communications fiables seront possibles pour des plus grandes distances ou la puissance de l'émetteur de l'engin spatial peut être réduite (ou une augmentation de sa longueur de vie et/ou réduction du poids de la batterie). Il est aussi possible de réduire la taille des antennes et en conséquence réduire le poids de l'engin. Tous ces facteurs ont contribué à ce que le code contrôle d'erreur joue un rôle incontournable dans la conception des systèmes de communication des engins spatiaux.

Du point de vue purement économique, l'impact est impressionnant. Dans la fin des années soixante, chaque dB de gain de codage est estimé à 1 million de dollars de coût qui inclue le développement et le lancement [Wic95]. Actuellement, la valeur d'un dB pour les communications spatiales est de 80 millions de dollars [Wic95].

La question est de savoir le gain de codage maximal qui pourrait être obtenu dans un système de communication. Shannon a répondu à cette question en considérant le cas d'un canal à bruit blanc additif et gaussien (AWGN). Il a ainsi défini la capacité du canal comme :

$$C = W \log_2(1 + \frac{S}{N_0}) \text{ bits par seconde} \tag{1.1}$$

où W est la largeur de bande du canal en Hertz, S est la puissance moyenne du signal et N_0 est la densité spectrale de puissance du bruit unilatérale dans le canal. En manipulant l'équation de la capacité, une limite du gain de codage pour la modulation BPSK peut être trouvée [Pie97]. Cette limite est exprimée en fonction de l'efficacité spectrale d'une modulation donnée η et du rapport E_b/N_0. L'efficacité spectrale est le nombre moyen de bits d'information transmis par un intervalle de signalisation de durée T. Si T est normalisée et reliée à l'inverse de la largeur de bande, η s'exprime en termes de bits par seconde par Hertz (b/s/Hz). La valeur de E_b/N_0 en fonction de η s'exprime sous la forme [Pie97] :

$$\frac{E_b}{N_0} > \frac{2^\eta - 1}{\eta} \tag{1.2}$$

Or l'efficacité spectrale pour une modulation BPSK est de 1 b/s/Hz donc la limite inférieure du E_b/N_0 est égale à 1 ou 0 dB. Donc le gain de codage maximum avec une BER de 10^{-5} est (9.6-0)=9.6 dB. Dans les années 80 et au début des années 90 plusieurs efforts ont été fournis de façon à obtenir un gain de codage supérieur à 7.4 dB. La majorité des efforts se sont concentrés sur les systèmes utilisant des concaténations en série, ce qui a conduit à des décodeurs de Viterbi très complexes et pas clairs car concaténés. La conséquence est l'obtention de plusieurs dixièmes de dB de gain mais au prix d'une grande complexité et de délai. La réalité est que 50 ans après la publication du théorème de Shannon, 2 dB de gain important séparaient les systèmes de contrôle d'erreur les plus avancés des résultats théoriques. Cet écart a presque disparu avec l'invention des Codes Turbo. Avant d'introduire les Codes Turbo, nous allons évoquer le fonctionnement des codes concaténés en série qui font partie du processus progressif de la découverte des Codes Turbo.

Forney dans son ouvrage sur les codes concatenés [For66], a examiné la concaténation en série d'un code Reed-Solomon avec un code convolutionnel. Le décodeur de Viterbi utilisé pour décoder les codes convolutionnels faisait une décision dure sur les symboles afin de les fournir au décodeur Reed-Solomon. Un entrelaceur est

inséré entre les codeurs afin de disperser les erreurs en salves ou rafales. Cette forme de concaténation permet d'avoir des très bonnes valeurs de BER et une faible complexité au niveau des décodeurs. Comme nous l'avons mentionné auparavant, ce type de codage a été utilisé par la NASA.

L'utilisation d'une décision dure à la sortie du premier décodeur limite les performances d'erreur. Alors que si le décodeur fournissait une information de fiabilité au deuxième décodeur au lieu de faire une décision sur le symbole codé, la capacité du système de codage concatené serait accrue. Mais malheureusement le décodeur de Reed-Salomon n'est pas adéquat pour fournir une décision douce. Par contre, il existe des algorithmes plus pratiques qui peuvent recevoir de l'information douce à l'entrée (Soft In) et qui produisent une décision douce à la sortie (Soft Out). L'algorithme "maximum a posteriori" (MAP) et le "Soft Output Viterbi Algorithm" (SOVA) sont parmi les meilleurs choix. Le développement de ces deux algorithmes a conduit tout droit à la concaténation de deux codes convolutionels.

L'événement marquant fut l'aboutissement des tentatives successives afin de s'approcher de la borne de Shannon. En 1993, Berrou, Glavieux et Thitimajshima ont montré avec l'introduction des Codes Turbo qu'on n'est plus qu'à 0.7 dB de la borne de Shannon soit un nouveau gain de codage de (2.2-0.7)=1.5 dB. Actuellement, les meilleures performances des Codes Turbo se situent à 0.2 dB de la borne de Shannon [Pie97]. Les Codes Turbo sont des codes convolutionnels concatenés en parallèle et décodés itérativement. Le premier codeur code l'information à transmettre alors qu'un entrelaceur sépare l'information du deuxième codeur.

Chaque code systématique est décodé en utilisant un décodeur Soft-In Soft-Out (SISO). La sortie du premier décodeur alimente le deuxième décodeur, ce qui correspond à une itération complète du décodeur turbo. Contrairement à l'algorithme de Viterbi qui fournit la séquence la plus vraisemblablement émise, l'algorithme MAP détermine le bit d'information le plus probable qui a été transmis dans une séquence codée. Pour des faibles valeurs de BER, la différence au niveau de la performance

entre le MAP et le Viterbi est quasi-négligeable. Étant donné que l'algorithme MAP est nettement plus complexe que l'algorithme de Viterbi, il a été longtemps ignoré. Cependant, pour des faibles valeurs de E_b/N_0 et des BER élevés, le MAP surpasse le Viterbi de 0.5 dB. Ceci est très important car dans le cas des Codes Turbo les performances BER sont élevées aux premières itérations. Par conséquent, la moindre amélioration obtenue pour des BER élevées implique une croissance au niveau des performances.

L'algorithme SOVA qui est aussi utilisé pour le décodeur turbo demeure le rival du MAP. Toutefois, les propriétés statistiques des sorties générées sont moins bonnes que celles qui sont fournies par l'algorithme MAP, ce qui résulte en une dégradation des performances de 0.8 dB.

1.2 Composition du mémoire

Ce mémoire se compose comme suit :

Le chapitre 1 introduit le principe du codage convolutionnel et décrit en particulier les codes convolutionnels récursifs et systématiques. Nous avons retracé le concept de base d'un système de communication. Le principe d'entrelacement qui est une partie intégrante des Codes Turbo est aussi abordé dans ce chapitre. Finalement, les différents types de canaux utilisés et la modulation BPSK sont aussi évoqués.

Dans le chapitre 2, nous nous sommes seulement intéressés à l'algorithme MAP. En premier lieu, nous avons introduit le principe du maximum a posteriori qui est la base de l'algorithme MAP. Ensuite nous avons présenté l'algorithme dans le cadre des codes convolutionnels. En second lieu, nous avons présenté la version de l'algorithme MAP que nous avons adoptée, et nous avons aussi analysé la complexité de l'algorithme.

Le chapitre 3 introduit le principe du décodage itératif. Une analyse de l'information extrinsèque est faite. Le décodeur utilisant l'algorithme MAP a été simulé, les performances des Codes Turbo sont ainsi mesurées, évaluées en fonction de plusieurs

paramètres (taux de codage, type et taille des entrelaceurs, longueur de contrainte) premièrement dans un canal AWGN et ensuite dans un canal de Rayleigh. Puisque le décodeur MAP nécessite l'estimation de la variance du bruit dans le canal, diverses méthodes d'estimation de la variance du bruit dans le canal sont présentées et simulées.

Au chapitre 4, nous avons traité le problème de la terminaison du treillis du codeur turbo qui est différent de celui de la treminaison d'un codeur convolutionnel ordinaire. Nous avons d'abord illustré la façon dont les deux codeurs convolutionnel peuvent être initialisés par l'envoi de deux queues. En effet, diverses façons de terminer le treillis sont possibles. Ensuite, nous avons présenté ces diverses solutions ou méthodes proposées dans la littérature et enfin nous avons évalué une nouvelle alternative qui consiste à ne pas transmettre de queue.

1.3 Contributions

Cette recherche a permis d'apporter les contributions suivantes :

1. Dresser une comparaison entre différentes versions de l'algorithme MAP et expliciter l'expression de la métrique de branche dans le cas d'un canal AWGN et d'un canal de Rayleigh.
2. Introduire les diverses méthodes qui permettent l'estimation de la variance du bruit dans le canal et faire une comparaison de trois différentes méthodes par le biais des simulations. Nous nous sommes basés sur le comportement de la fonction de densité de probabilité (PDF) afin de comparer les avantages et les inconvénients de chacune des ces méthodes.
3. Comparer les différentes alternatives de terminaison de treillis des Codes Turbo tout en proposant une méthode simple de terminaison pour les blocs de tailles moyennes. Étudier l'influence du choix l'entrelacement, de la longueur du bloc d'information et du type de canal (AWGN ou Rayleigh) sur le choix de la

terminaison.

Chapitre 2

Concaténation Parallèle

2.1 Introduction

La description des Codes Turbo nécessite l'introduction des éléments de base et des structures sur lesquelles le système décrit dans le mémoire est fondé. C'est dans cette optique que nous allons parcourir brièvement les fondements des Codes Turbo. Nous allons voir que les Codes Turbo sont avant tout des codes convolutionnels avec une certaine spécificité. Avant de décrire les codes convolutionnels en général et les codes récursifs en particulier, nous allons retracer le concept de base d'un système de communication. Le principe d'entrelacement qui est une partie intégrante des Codes Turbo sera abordé dans ce chapitre. Finalement, les différents types de canaux utilisés et la modulation BPSK sont aussi évoqués.

2.2 Système de Communication

Un système de communication numérique est un moyen de transport de l'information d'un usager à un autre en général éloigné. Le système est appelé numérique ce qui signifie que l'information est représentée par une séquence de symboles choisis parmi un alphabet fini (binaire par exemple). Nous avons présenté les éléments de

base d'un système de communication numérique à la figure (2.1). Un système de communications consiste à faire subir à l'information émise plusieurs opérations comme le codage de source, le cryptage et le codage de canal avant de la transmettre.

Le codage de source [Skl88] est utilisé afin de réduire la redondance dans les informations issues de la source. La source de données peut être analogique ou numérique. Dans le premier cas le signal représentant la source doit d'abord être échantillonné, ensuite quantifié et enfin transformé en une suite de 0 et 1 par les techniques de conversion analogique numérique comme la modulation à impulsion codée (PCM) par exemple. Citons, à titre d'exemple d'autre technique de conversion analogique-numérique tels que la DPCM et la DM. Dans le cas où la source est numérique, le codage de source est directment appliqué à la source. Nous pouvons citer le codage de Huffman à titre d'exemple. Le cryptage empêche un usager non autorisé de comprendre le message envoyé ou d'injecter des fausses informations. Le codage de canal encore appelé codage pour le contrôle d'erreur est utilisé afin d'ajouter de la redondance à l'information transmise dans le but de la protéger contre le bruit et les perturbations introduites dans le canal. Notons que l'ordre de ces trois premiers blocs (codeur de source, encrypteur et codeur de canal) ne peut être modifié. Mentionnons par contre que plusieurs chercheurs [Hag95] se sont intéressés à trouver une alternative qui puisse regrouper ces trois blocs en un seul ou au moins faire les opérations de codage de source et de codage correcteur d'erreur en une seule opération. Le modulateur adapte la séquence codée au canal physique qui peut être modélisé sous plusieurs formes qui dépendent du type d'application (liaison satellite, ou terrestre par câble ou sans fil).

À la figure (2.1), nous avons illustré trois différents types de canal : le premier est un canal spatial c.à.d le signal émis est transmis dans un canal satellite où le signal est acheminé directement ou par l'intermédiaire d'un station terrestre au récepteur. Le deuxième type de canal est typique des liaisons radiomobile sans fil où le signal est transmis par des relais d'antennes avant d'atteindre le récepteur. Enfin le dernier type

de canal, qui est le plus traditionnel, est celui d'une liaison câblée où les usagers sont reliés par un câble (coaxial, fibre optique, etc.) comme dans le réseau téléphonique. Les modèles de canaux utilisés dans notre étude sont présentés au paragraphe (2.7).

À la réception, les étapes inverses de l'émission sont appliquées à l'information reçue : le signal est démodulé, ensuite il est décodé par le décodeur de canal. Le décodage est effectué dans le but de reproduire exactement la séquence émise et de minimiser la probabilité d'erreur. Le décodage peut être décrit comme un algorithme qui exécute des opérations sur les symboles bruités reçus du canal afin de les corriger. Enfin le signal est décrypté avant de subir le décodage de source de façon à retrouver l'information originale.

Après avoir décrit un sytème de communication, nous allons présenter les codes convlutionnels.

2.3 Codage Convolutionnel Récursif

Un codeur convolutionnel binaire (n, k) est un dispositif qui accepte des k-tuples binaires en entrée et produit n-tuples binaires en sortie. Il existe deux types de codeurs convolutionnels, le premier est appelé codeur convolutionnel RFI (Réponse Finie à une Impulsion et connu sous le nom de codeur convolutionnel non récursif). Le deuxième type est appelé codeur convolutionnel RII (Réponse Infinie à une Impulsion ou mieux connu sous le nom de codeur convolutionnel récursif). Un codeur convolutionnel est systématique si l'information présente à l'entrée est produite à la sortie. Dans le cas contraire le codeur est dit non-systématique.

Un codeur est dit RFI si sa sortie peut s'exprimer comme une combinaison linéaire de l'entrée courante et d'un nombre fini des entrées précédentes. La combinaison linéaire est exprimée en fonction des bits d'entrée et des séquences génératrices du codeur. Soit la séquence génératrice $\{g_{i,o,l}\}$ qui relie une séquence d'entrée particulière $\{d_{t-l}\}$ à une séquence de sortie particulière $\{X_t^o\}$. Une valeur particulière de $g_{i,o,l}$

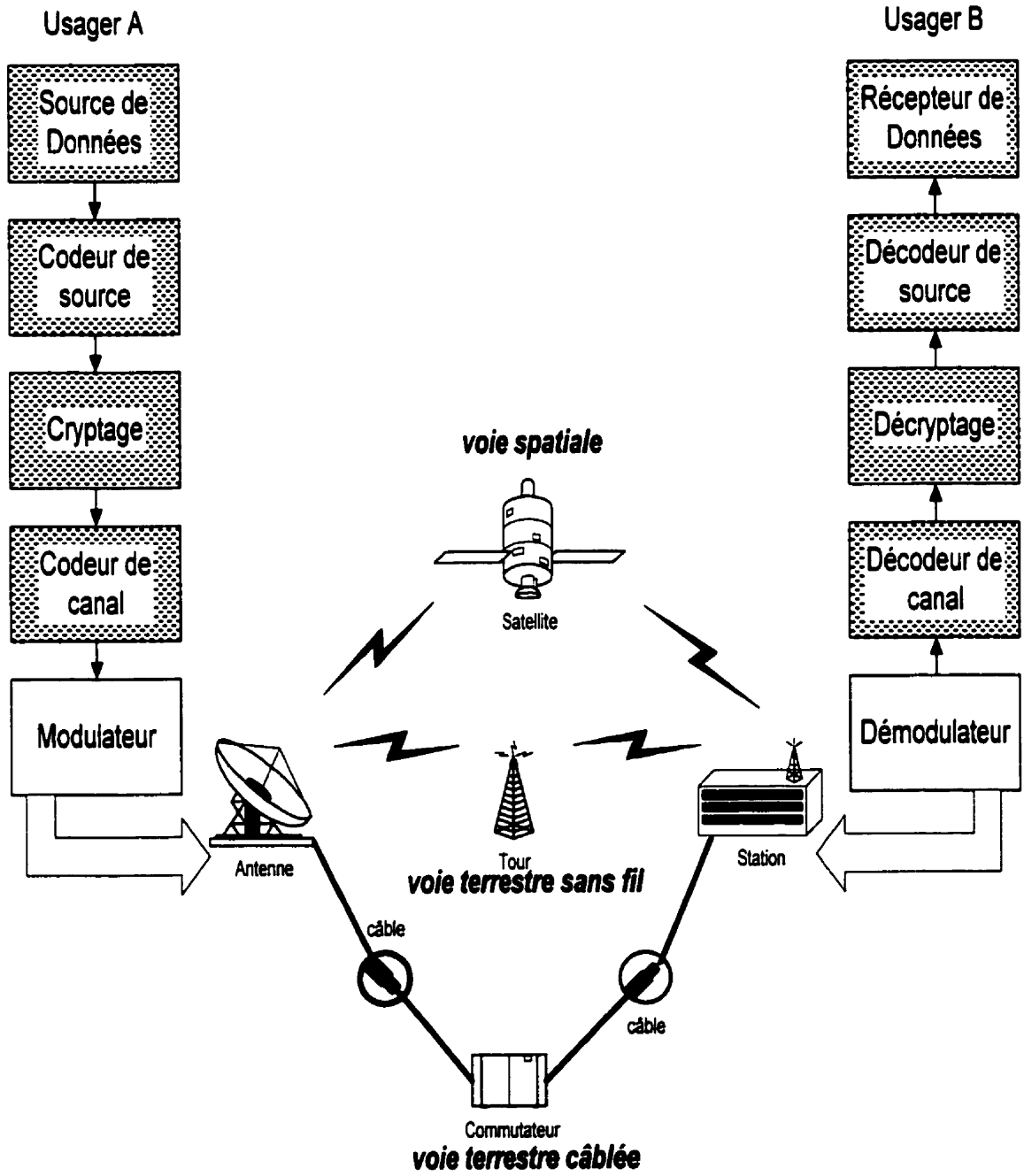

Figure 2.1: Système de Communication

signale la présence ou bien l'absence d'une connexion entre le $l$ieme élément de mémoire $l \in [1, v_i]$ du $i$ieme entrée $i \in [1, k]$ à la $o$ieme sortie $o \in [1, n]$. Les n équations de sortie ont la forme :

$$X_t^o = \sum_{i=1}^{k} \sum_{l=1}^{v_i} g_{i,o,l} d_{t-l}, \tag{2.1}$$

Les mémoires de chacune des k entrées sont énumérées par le vecteur mémoire $(v_1, v_2, \cdots, v_k)$. Aux figures (2.2) et (2.3), les codeurs possèdent une seule entrée donc $k = 1$, nous remarquons que $n = 2$ (deux sorties) et enfin $v_1 = 2$ (deux délais) est la valeur de la mémoire.

En définissant D le délai séparant deux instants successifs, les sorties des codeurs RFI peuvent s'exprimer en fonction des entrées par l'équation matricielle suivante :

$$\begin{aligned} \vec{\mathbf{X}}(D) &= [X_1(D), X_2(D), ..., X_n(D)] \\ &= [d_1(D), d_2(D), ..., d_n(D)] \begin{bmatrix} g_{1,1}(D) & g_{1,2}(D) & \cdots & g_{1,n}(D) \\ g_{2,1}(D) & g_{2,2}(D) & \cdots & g_{2,n}(D) \\ \vdots & \vdots & \ddots & \vdots \\ g_{k,1}(D) & g_{k,2}(D) & \cdots & g_{k,n}(D) \end{bmatrix} \\ &= \vec{d(D)} G(D) \end{aligned} \tag{2.2}$$

où $d_u(D) = \sum_t d_t^u D^t$. La matrice polynomiale $G(D)$ est appelée la matrice génératrice du codeur. Dans le cas d'un codeur RFI, les termes de la matrice génératrice sont des polynômes de la forme $g_{i,o}(D) = \sum_{l=1}^{v_i} g_{i,o,l} D^l$ dont le degré est au plus v_k. En appliquant les définitions précédentes au codeur de la figure (2.2), nous obtenons la matrice génératrice suivante :

$$G(D) = \begin{bmatrix} 1 + D^2 & 1 + D + D^2 \end{bmatrix} \tag{2.3}$$

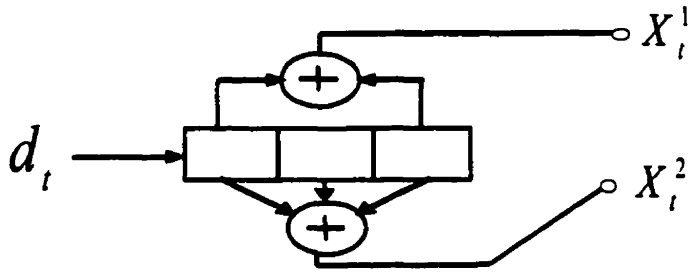

Figure 2.2: Codeur Convolutionnel non récursif et non systématique de matrice génératrice $G(D) = [1 + D^2 \quad 1 + D + D^2]$

Un codeur convolutionnel est dit RII si les n équations de sortie sont en fonction des précédentes entrées et sorties, ce qui implique que la sortie dépend d'une infinité d'entrées précédentes . Un exemple de codeur RII est présenté à la figure (2.3). Ce dernier est aussi systématique du fait que l'entrée du codeur se retrouve à sa sortie. Dans le cas des Codes Turbo, l'intérêt se porte sur les codeurs convolutionnels RII et systématiques. D'où l'intérêt de transformer un codeur RFI spécifique en un codeur RII systématique. La condition à satisfaire est que le codeur RFI soit non catastrophique [Hee99]. Un codeur est dit non catastrophique si et seulement si le seul chemin qui produit exclusivement des zéros à la sortie est la boucle unique qui part de l'état initial du codeur (en général l'état zéro c.à.d les registres du codeur sont supposés remplis par des zéros) et y retourne instantanément [Wic95]. Le codeur systématique récursif (RSC) de la figure (2.3) est construit à partir du codeur non systématique non récursif (NRNSC) de la figure (2.2). De la même façon qu'un codeur RFI, un codeur convolutionnel RII peut être décrit par une matrice génératrice. A la différence du cas non récursif, les éléments de la matrice génératrice sont des fonctions rationnelles de la variable D. Ajoutons, que dans le cas où le codeur est aussi systématique, au moins le premier terme de $G(D)$ doit être égal à 1. La matrice génératrice du codeur

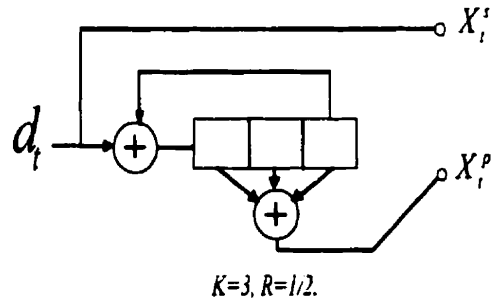

Figure 2.3: Codeur Systématique et Récursif (RSC) de matrice génératrice $G(D) = \left[1 \quad \frac{1+D+D^2}{1+D^2}\right]$

systématique récursif à la figure (2.3) s'écrit sous la forme :

$$G(D) = \left[1 \quad \frac{1+D+D^2}{1+D^2}\right] \tag{2.4}$$

Il y a trois alternatives pour représenter un code convolutionnel : l'arbre, le treillis et le diagramme d'état. Notons que le treillis peut être compacté sous une autre forme qu'on nomme le diagramme d'état. Ce diagramme est tout simplement un graphe représentant les états possibles du codeur et les transitions d'un état à un autre. Les diagrammes d'états des codeurs précédents (figure (2.2) et (2.3)) sont présentés aux figure (2.4) et (2.5). S_0 , S_1, S_2 et S_3 correspondent respectivement aux états 00, 01, 10 et 11. Les lignes en gras indiquent que le bit d'information est un 1 et celles en pointillés indiquent que c'est un 0. La structure des diagrammes d'états est identique pour les codeurs RFI de la figure (2.3) et RII de la figure (2.2). Cependant, les séquences de sorties des codeurs RFI et RII sont différentes pour la même séquence d'entrée. Nous pouvons remarquer ceci en comparant les deux diagrammes d'états précédents : par exemple le passage de l'état S_1 à l'état S_0 a été provoqué par le bit d'entrée 0 pour le codeur de la figure (2.2) tandis que ce même passage est provoqué par le bit 1 pour le codeur de la figure (2.3).

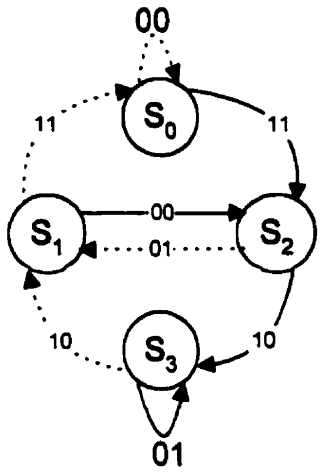

Figure 2.4: Diagramme d'état du codeur RIF de la figure (2.2).

Ayant décrit les codeurs convolutionnels. nous allons introduire l'entrelacement qui est une technique de protection contre les salves de bruit très utilisée en communications numériques en général et dans le codage Turbo en particulier.

2.4 Entrelacement

Les codes utilisés pour des canaux satellites sont conçus pour combattre des erreurs indépendantes et surtout espacées dans le temps. D'autre types de canaux (radiomobile, téléphonique...) sont soumis à des erreurs qui arrivent en salves ("burst error"). L'utilisation d'une technique qui est appelée "entrelacement" permet d'améliorer la capacité de correction du code et de lutter contre les erreurs en salves. Le but de la technique est d'espacer les symboles de bruit consécutifs. Nous pouvons définir l'entrelaceur comme une technique qui prend les symboles d'information d'un alphabet fixe à l'entrée et qui reproduit les même symboles mais dans un ordre temporel différent.

Il existe deux types ou familles d'entrelaceurs : l'entrelacement "bloc" et l'entrelacement "convolutionnel"[Wic95]. La famille d'entrelacement bloc regroupe plusieurs

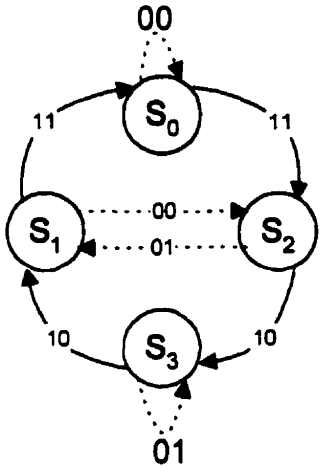

Figure 2.5: Diagramme d'état du codeur RII de la figure (2.3).

types d'entrelacement comme l'entrelacement bloc classique, pseudo-aléatoire classique, aléatoire, hélicoïdal etc. Nous nous sommes surtout intéressés à l'entrelacement bloc classique et au pseudo-aléatoire. Pour avoir plus de détails, on pourra se référer à [Hee99].

Un entrelaceur bloc classique est un entrelaceur de période $P = N.M'$ décrit par une matrice de taille $N \times M'$. Ce type d'entrelaceur se caractérise par un processus dans lequel les données sont écrites par ligne dans une matrice et ensuite ces données sont lues par colonne. Il existe quatre différentes façons de lire les données. Ces façons dépendent de l'ordre dans lesquels les colonnes (GD : de gauche à droite ou DG : de droite à gauche) et les lignes (HB : du haut vers le bas ou BH : du bas vers le haut) sont écrites. Ces modes d'opérations sont illustrés à la figure (2.6). Nous allons essayer à l'aide d'un exemple d'illustrer le fonctionnement de l'entrelaceur bloc classique. Supposons qu'une séquence de 9 symboles ($N = 3, M' = 3, P = 9$) soit reçue à l'entrée de l'entrelaceur. Nous allons d'abord placer ces 9 symboles dans une matrice (voir figure (2.1))dans un ordre croissant des nombres indiqués dans les cases (c.à.d le premier symbole reçu est placé dans la case 0, le deuxième dans la case 1 et ainsi de suite). En appliquant les quatre différents modes d'opérations à la séquence

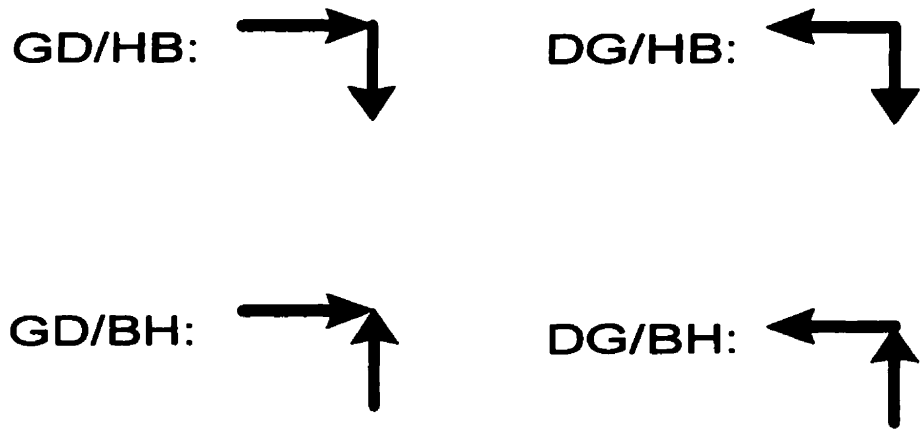

Figure 2.6: Les quatres modes d'opérations d'un entrelaceur bloc classique.

de la figure (2.1) les symboles sont réécrits à la sortie de l'entrelaceur bloc classique dans l'ordre suivant :

pour GD/HB : 0 3 6 1 4 7 2 5 8
pour GD/BH : 6 3 0 7 4 1 8 5 2
pour DG/HB : 2 5 8 1 4 7 0 3 6
pour DG/BH : 8 5 2 7 4 1 6 3 0

0	1	2
3	4	6
6	7	8

Tableau 2.1: Entrelaceur bloc classique de taille 3×3

Un entrelaceur aléatoire est un entrelaceur bloc qui est généré par une permutation aléatoire basée sur une séquence aléatoire. Par exemple, un vecteur aléatoire de taille P est généré , ensuite la permutation qui ordonne le vecteur alétoire dans un certain ordre est utilisée afin de générer l'entrelaceur aléatoire. En pratique, le vecteur aléatoire peut être généré par un générateur de bruit pseudo-aléatoire.

Les entrelaceurs pseudo-aléatoires sont les plus performants pour les Codes Turbo

dans le cas des blocs d'information de tailles moyenne et grande comme nous allons le voir dans les chapitres suivants. Leurs inconvénients par rapport aux entrelaceurs blocs classiques résident dans leur complexité. Notons au passage qu'il existe plusieurs façons de décrire des entrelaceurs, ce qui est utile pour les étudier et les concevoir. En particulier, les questions de délai, d'espace mémoire requis et la capacité de disperser ou d'espacer les symboles bruités [Hee99]. La conception des entrelaceurs pour les Codes Turbo est loin d'être une science exacte d'où la nécessité de valider la performance des entrelaceurs par simulation. Cependant, chaque simulation démontre seulement les performances de l'entrelaceur pour un choix particulier de codeur et de décodeur. Donc, la valeur générale d'une méthode de conception d'un entrelaceur ne fait pas l'unanimité parmi les chercheurs.

2.5 Concaténation parallèle

La concaténation de plusieurs codeurs convolutionnels en parallèle a favorisé la découverte des Codes Turbo. En effet l'idée de relier ou concaténer plusieurs codeurs est ancienne. Forney en 1965 a été le premier à introduire les codes concaténés en série [For66]. La concaténation en série consiste à relier deux codeurs de type différent dont le plus classique consiste à relier un codeur de Reed Solomon à un codeur convolutionnel, séparés par un entrelaceur. La concaténation en série a permis une décroissance exponentielle dans les probabilités d'erreur au prix d'une croissance de la complexité de décodage. Presque 30 ans plus tard, la concaténation parallèle a été introduite par Berrou, Glavieux et Thitimajshima [BGT93]. C'était l'un des deux concepts fondamentaux des Codes Turbo, le deuxième concept étant le décodage itératif. La concaténation parallèle comme la concaténation en série, utilise deux ou plusieurs codeurs reliés par un entrelaceur. Dans le cas des Codes Turbo, les deux codeurs sont de préférence identiques et de type convolutionnel récursifs systématiques (RSC). À la figure (2.7), un codeur Turbo qui consiste en la concaténation parallèle

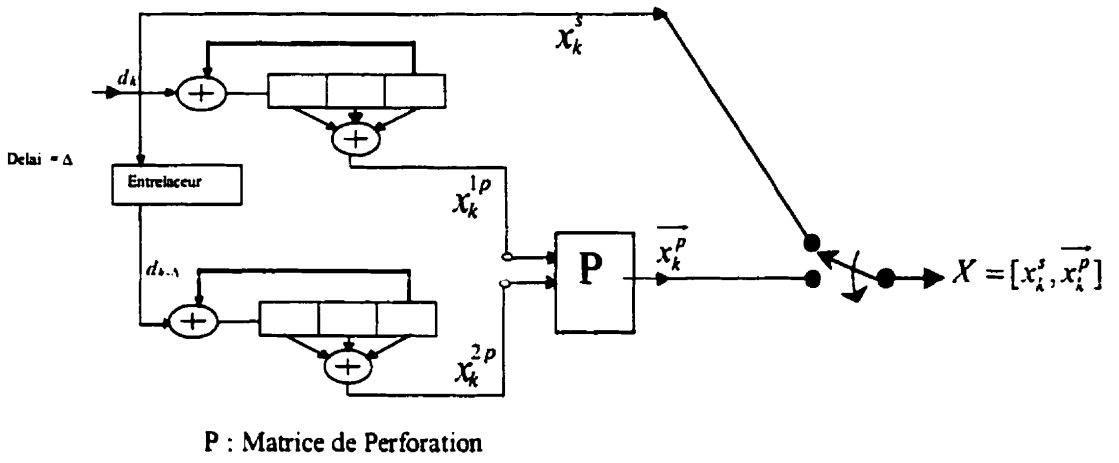

Figure 2.7: Codeur Turbo

de deux codeurs identiques de type RSC séparés par un entrelaceur est présenté. À l'instant k le bit d_k est codé par le codeur supérieur lequel produit le bit de parité x_k^{1p}. L'ensemble des bits d_k sont ensuite entrelacés et codés par le codeur inférieur qui produit le second bit de parité x_k^{2p}. Les deux bits de parité peuvent être perforés ou non afin de produire le vecteur de parité $\overrightarrow{x_k^p}$ qui sera multiplexé avec le bit d'entrée x_k^s. Si $\overrightarrow{x_k^p} = (x_k^{1p}, x_k^{2p})$ alors le taux de codage du codeur Turbo est égale à 1/3. Les séquences codées générées par un codeur concaténé en parallèle ont permis d'atteindre des performances d'erreur surprenantes qui sont de l'ordre de quelques dixièmes de dB de la limite de Shannon [BGT93] [BGl96] [Hag96][Rob94]. Mentionnons que l'utilisation de codeurs RSC se justifie par le fait que pour des petits rapport signal sur bruit les codeurs RSC sont plus performants que les codeurs NRNSC pour la même longueur de contrainte [Ryan]. Ajoutons que ceci est aussi valable lorsque le taux de codage est élevé [Skl97]. Par contre pour des rapports signal à bruit élevés, les codeurs NRNSC donnent une performance d'erreur meilleure que les codeurs RSC.

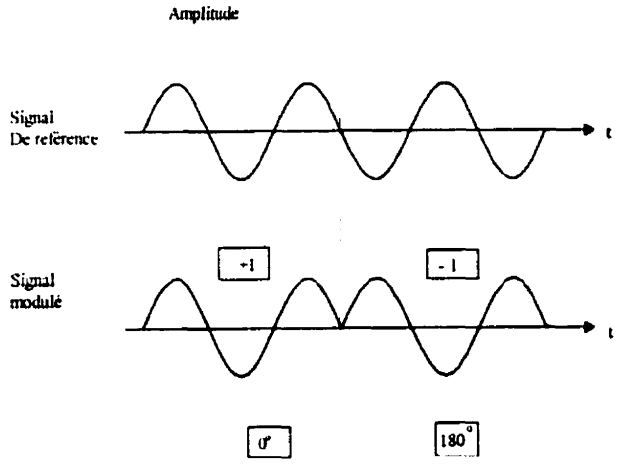

Figure 2.8: Forme d'un signal BPSK

2.6 Modulation BPSK

Dans notre travail, nous avons considéré le cas où la modulation est BPSK et le canal est un canal binaire symétrique.

La modulation BPSK (Binary Phase Shift Keying) est parmi les modulations numériques les plus utilisées [Meh94]. En effet, elle est la plus performante au niveau de la probabilité d'erreur et donc la plus adéquate pour les communications spatiales (un gain de 3 dB par rapport au BFSK). En BPSK, les phases opposées de la porteuse (0 et π) sont transmises toutes les T secondes (si on considére que la durée d'un bit est T) et sont basées sur les symboles d'information +1 ou -1. BPSK peut être considéré comme un type de modulation d'amplitude où la porteuse est multipliée par -1 ou +1 suivant le bit de parité comme le montre la figure (2.8) où les données +1 et -1 sont représentées respectivement par $Cos(\omega_c t)$ et $Cos(\omega_c t + \pi)$. Nous pouvons représenter ceci sous forme de constellation dans un plan x/y. BPSK se limite à deux signaux sur l'axe des x tel qu'il est montré à la figure (2.9) où E_b est l'énergie d'un bit. Contrairement à la structure du modulateur BPSK [Lab87] montrée à la figure (2.10), le démodulateur est beaucoup plus complexe. En effet le récepteur BPSK doit fournir une phase de référence cohérente afin de démoduler le signal reçu.

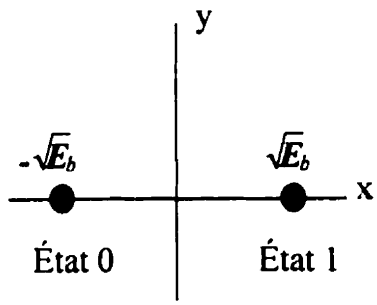

Figure 2.9: Constellation du signal BPSK

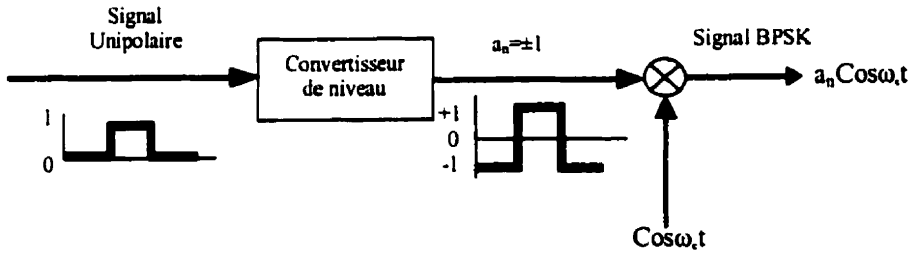

Figure 2.10: Modulateur BPSK

Ou encore, un bit de synchronisation permet de retrouver la phase de référence. Le problème de synchronisation s'élimine dans le cas de l'utilisation de la modulation DPSK [Pro95][Wic95] mais ceci s'accompagne par une chute dans le taux d'erreur par bit (**BER**) de 0.3 dB dans un canal Gaussien (l'écart s'agrandit dans le cas d'un canal radiomobile). Mentionnons finalement que le **BER** de la modulation **BPSK** cohérente dans un canal **AWGN** est donné par la formule suivante :

$$P_b = Q(\sqrt{\frac{2E_b}{N_o}}), \qquad \text{où} \quad Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-y^2/2} dy \tag{2.5}$$

2.7 Les Types de Canaux

Le bruit introduit par un canal peut prendre différentes formes. Le modèle de bruit le plus commun est celui du bruit additif blanc et Gaussien (AWGN). Ce dernier modèle est le plus simple et en plus il fournit un modèle presque parfait dans certains systèmes de communication. Par exemple, pour les canaux satellite où les communications sont en vue directe, le modèle AWGN est exact. Mentionnons que l'adjectif "additif" dans AWGN signifie que l'impact du bruit sur le signal transmis peut être modélisé comme une variable aléatoire n qui s'additionne au signal modulé. La variable n est supposée Gaussienne de moyenne nulle et de variance σ^2 et dont la densité de probabilité est définie par :

$$p(n) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-n^2/2\sigma^2} \tag{2.6}$$

Le terme "blanc" indique que la densité spectrale de puissance du bruit est considérée unilatérale (c.à.d de la fréquence 0 à ∞) et constante de valeur N_0. Si x_k est la variable qui représente le signal modulé correspondant à un bit d'information à l'instant k, alors à la sortie du canal nous recueillons :

$$r_k = x_k + n \tag{2.7}$$

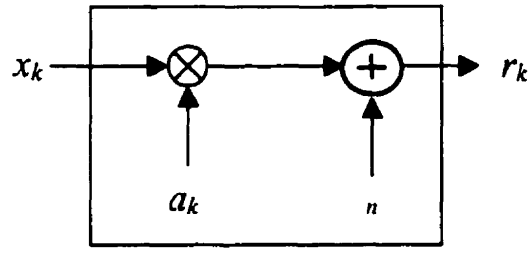

Figure 2.11: Modèle d'un canal Rayleigh

Dans les communications radiomobiles, l'atténuation du signal varie avec la vitesse du véhicule (qui est le récepteur) et la nature des obstacles. Ce type de canal est appelé "multipath fading channel", l'amplitude a_k du signal est souvent modélisée suivant la loi de Rayleigh ou de Rice. Nous allons seulement nous intéresser au cas de Rayleigh. Dans ce cas, le signal reçu est sous la forme :

$$r_k = a_k x_k + n \tag{2.8}$$

où a_k porte aussi le nom d'enveloppe du signal. Sa fonction de densité de probabilité est donnée par :

$$p(a_k) = a_k e^{-\frac{(a_k)^2}{2}}, \qquad a_k \geqslant 0 \tag{2.9}$$

Le modèle d'un canal de Rayleigh est schématisé à la figure (2.11) où r_k est la version bruitée du signal x_k à l'entrée du canal . La méthode la plus simple afin d'obtenir l'enveloppe dont la puissance moyenne est unitaire est de générer deux variables Gaussiennes b_k et c_k de variance 1/2 et de moyenne nulle. En considérant que le processus d'évanouissement ou fading est décorrélé (c.à.d il n'y a pas de corrélation temporelle entre les symboles qui subissent l'évanouissement), l'enveloppe du signal

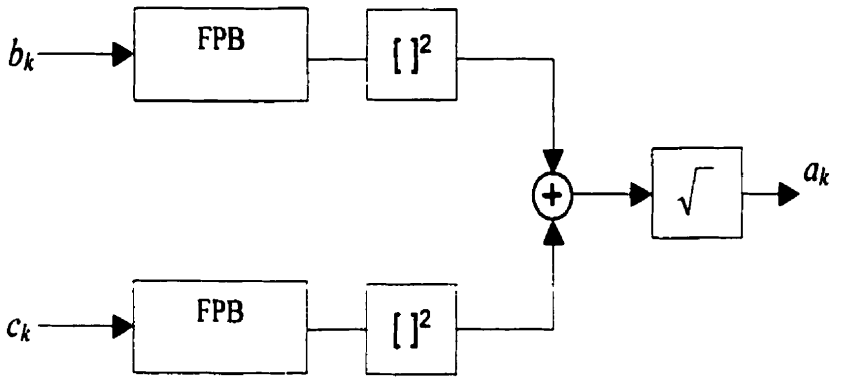

Figure 2.12: Enveloppe du signal

sera donnée par :

$$a_k = \sqrt{b_k^2 + c_k^2} \tag{2.10}$$

La figure (2.12) suggère la façon dont l'enveloppe a été générée. Les blocs "FPB" représentent des filtres passe-bas mais, comme nous avons fait la supposition de décorrélation, alors la largeur de bande des "FPB" est supposée égale à l'infinie [Hal96][Bar96].

2.8 Conclusion

Dans ce chapitre, nous avons mis la lumière sur le codeur Turbo qui est constitué de codeurs convolutionnels récursifs et systématiques. Nous avons aussi décrit les conditions ou les hypothèses qui nous serviront comme base de départ de nos simulations. Le principe d'entrelacement bloc classique et aléatoire a été décrit. Comme nous l'avons signalé dans la description du système de communications : un système de codage ou un codeur ne peut pas fonctionner seul. Nous avons donc besoin d'un décodeur d'où la nécessité d'un algorithme de décodage qui pourrait remplir cette tâche. L'algorithme MAP est l'algorithme par excellence qui est désigné ou choisi par la majorité des chercheurs pour le décodage des Codes Turbo. Le chapitre suivant aura donc la lourde tâche de décrire (tout en essayant d'être simple et non pas

simpliste) le principe et l'algorithme MAP.

Chapitre 3

L'Algorithme M.A.P

3.1 Introduction

Le choix d'un algorithme de décodage est un problème majeur. En effet, deux algorithmes de décodage ont été proposés pour les Codes Turbo. Le premier est le SOVA (Soft Output Viterbi Algorithm) qui est une variante de l'algorithme de Viterbi [Bha85]. Le deuxième est celui que nous allons traiter dans ce chapitre et qui porte le nom de l'algorithme MAP ou BJRC (Bahl, Jelenik, Raviv, Cocke) [BCJ74]. La caractéristique principale de ces deux algorithmes est qu'ils fournissent une mesure de fiabilité sous forme probabiliste (appelée aussi métrique) sur chaque symbole décodé [Hee99]. Dans ce chapitre, nous allons seulement nous intéresser à l'algorithme MAP. Dans un premier lieu, nous allons introduire le principe du maximum a posteriori qui est la base de l'algorithme MAP. Ensuite nous allons présenter l'algorithme dans le cadre des codes convolutionnels. Dans un deuxième lieu, nous allons présenter une version de l'algorithme MAP que nous avons adoptée. Finalement, après avoir décrit brièvement différentes versions optimales ou sous optimales de l'algorithme, nous allons analyser la complexité de l'algorithme, au niveau des ressources informatiques requises pour son implantation.

3.2 Le principe du Maximum A Posteriori

Contrairement à l'algorithme de Viterbi (VA), où le but est de minimiser la probabilité d'erreur d'un mot ou d'une séquence de symboles, l'algorithme MAP consiste à minimiser la probabilité d'erreur par symbole ou par bit. Dans le cas de VA la règle de décision est appelée le principe de maximum de vraisemblance (ML). Dans le cas du MAP on parlera de la règle du maximum a posteriori c.à.d de prendre le maximum des probabilités a posteriori (APP) du symbole ou bit envoyé dans le canal. Le calcul de l'APP se base sur l'observation de la séquence reçue, d'où la nomination "a posteriori" qui signifie en latin "en partant de ce qui vient après". Afin d'éviter la confusion des termes qui sont utilisés dans la littérature, nous rappelons que la notion Algorithme MAP signifie que la règle maximum a posteriori associée à un symbole a été appliquée. Nous pouvons alors parler de l'algorithme MAP symbole-par-symbole. La règle de maximum a posteriori peut être appliquée à un symbole ou une séquence. Dans le cas où cette règle est appliquée à une séquence (ou un mot de code par exemple) et que la distribution des mots de source est uniforme, ceci revient exactement à appliquer le principe ML. En effet, il suffit d'appliquer la règle de Bayes afin de prouver l'identité entre les deux règles. L'algorithme MAP est aussi désigné par les termes SISO ou BJRC.

Donc à la sortie du décodeur, les bits décodés ne sont pas comparées à un seuil donné, les valeurs (des bits) obtenues ne sont pas modifiées (par large approximation ou quantification). Dans ce cas, nous pouvons parler d'une décision douce (Soft) en opposition à une décision dure où la valeur du bit à la sortie du décodeur est affectée soit à la valeur 0 soit à 1.

Soient les événements $A_i, i = 1, 2, ..., M$, l'ensemble des messages transmis dans un intervalle de temps donné, $P(A_i)$ représente la probabilité a priori de ces événements. Soit B le signal reçu qui n'est qu'un des A_i corrompu par du bruit. B est une variable aléatoire continue de densité de probabilité $p(B)$. L'expression $P(A_i|B)$ est l'APP de l'événement A_i sachant que le signal reçu B a été observé. Le détecteur MAP cherche

à maximiser l'APP $P(A_i|B)$. Cette dernière s'exprime sous la forme [Pro95] :

$$P(A_i|B) = \frac{p(B|A_i)P(A_i)}{p(B)} \tag{3.1}$$

où $p(B|A_i)$ est la fonction de densité de probabilité (pdf) conditionnelle du vecteur observé A_i, et $P(A_i)$ est la probabilité a priori du ième signal transmis. Le dénominateur de l'équation (3.1) est donné par :

$$p(B) = \sum_{i=1}^{M} p(B|A_i)P(A_i) \tag{3.2}$$

Notons que $p(B|A_i)$ est appelé aussi la fonction de vraisemblance. Le critère de décision qui consiste à prendre le maximum de $p(B|A_i)$ parmi les M signaux est le principe du ML si les signaux sont équiprobables.

Si la distribution des signaux A_i est uniforme (c.à.d les signaux A_i sont équiprobables), or $p(B)$ est indépendant du signal A_i transmis, donc maximiser $P(A_i|B)$ revient à maximiser l'expression $p(B|A_i)$. Donc le détecteur basé sur le principe MAP fera la même décision que celui basé sur le principe ML.

Dans notre étude les événements A_i appartiennent au champ de Galois $GF(2)$ dont les éléments sont $\{0, 1\}$ où 0 est l'élément neutre sous l'addition $\oplus$, donc $M = 2$. Soit R la séquence reçue pour toutes les classes de signaux. Alors l'équation (3.1) devient :

$$P(d_k = i|R) = \frac{p(R|d_k = i)P(d_k = i)}{p(R)}, \qquad i = 0, 1 \tag{3.3}$$

La règle de décision MAP consiste à comparer les probabilités APP $P(d_k = 0|R)$ et $P(d_k = 1|R)$, ensuite à en prendre le maximum, ceci peut se résumer dans l'équation

suivante :

$$P(d_k = 1|R) \underset{H_1}{\overset{H_0}{\lessgtr}} P(d_k = 0|R) \tag{3.4}$$

Dans l'équation (3.4) l'hypothèse H_1 signifie que le détecteur MAP assigne d_k à la valeur 1 dans le cas où $P(d_k = 1|R)$ est supérieur à $P(d_k = 0|R)$. Dans le cas contraire le détecteur MAP choisit $(d_k = 0)$ ce qui correspond à l'hypothèse H_0.

Afin de faire le lien entre l'équation (3.4) et le rapport de vraisemblance, il suffit de remplacer l'équation (3.3) dans l'équation (3.4) :

$$\frac{p(R|d_k = 1)P(d_k = 1)}{p(R)} \underset{H_1}{\overset{H_0}{\lessgtr}} \frac{p(R|d_k = 0)P(d_k = 0)}{p(R)} \tag{3.5a}$$

$$p(R|d_k = 1)P(d_k = 1) \underset{H_1}{\overset{H_0}{\lessgtr}} p(R|d_k = 0)P(d_k = 0) \tag{3.5b}$$

$$\frac{p(R|d_k = 1)P(d_k = 1)}{p(R|d_k = 0)P(d_k = 0)} \underset{H_1}{\overset{H_0}{\lessgtr}} 1 \tag{3.5c}$$

En supposant que les bits de données d_k sont équiprobables, l'équation (3.5a) se réduit au rapport des fonctions de vraisemblance. En prenant le logarithme de l'équation (3.5a), on obtient le logarithme du rapport de vraisemblance LLR :

$$LLR(d_k) = \log_e \left[\frac{p(R|d_k = 1)}{p(R|d_k = 0)}\right] = \log_e \left[\frac{P(d_k = 1|R)}{P(d_k = 0|R)}\right] \tag{3.6}$$

Le LLR est la charnière de l'algorithme MAP. Nous pouvons déduire deux pro-

priétés de l'équation (3.6), la première est que la valeur du LLR n'est pas entière donc l'algorithme MAP délivre une réponse quantifiée **douce** et non plus dure. La deuxième propriété est que le LLR obtenu est valable pour un seul bit ou symbole et non plus pour une séquence entière, comme c'est le cas pour le VA.

3.3 Notations et règle de decision

Afin d'appliquer le principe MAP aux Codes Turbo, considérons un (RSC) de taux de codage $R = 1/2$ (Voir Figure (2.3)). Nous allons adopter les notations suivantes :

- $d_k = i, \quad i = 0, 1$ le bit d'information présent à l'entrée du codeur à l'instant k.
- $M = (K - 1)$ bits, est la mémoire du codeur.
- $S_k = m, \quad m = 0, 1, \cdots, 2^M$ est l'état du codeur à l'instant k.
- N est la taille de la séquence à coder (ou encore la longueur du bloc exprimée en bits).
- $(d_1, \cdots, d_k, \cdots, d_N) = (X_1^s, \cdots, X_k^s, \cdots, X_N^s)$ est la séquence à coder.
- $(X_1^s, \cdots, X_k^s, \cdots, X_N^s)$ est appelée la séquence systématique.
- $(X_1^p, \cdots, X_k^p, \cdots, X_N^p)$ est la séquence de parité à la sortie du codeur RSC.
- $\mathbf{R}_k = (r_k^s, r_k^p)$ est la version bruitée de (X_k^s, X_k^p) à l'instant k.
- $\mathbf{R}_1^N = (\mathbf{R}_1, \cdots, \mathbf{R}_k, \cdots, \mathbf{R}_N)$ est la séquence reçue après passage dans un canal (AWGN ou Rayleigh dans notre cas).

En supposant que la séquence reçue est $\mathbf{R}_1^N$, l'expression du LLR devient :

$$LLR(d_k) = \log_e \left[\frac{P(d_k = 1|\mathbf{R}_1^N)}{P(d_k = 0|\mathbf{R}_1^N)} \right] \tag{3.7}$$

Le calcul de l'APP $P(d_k = i|\mathbf{R}_1^N)$, $i = 1, 0$ s'avère complexe d'où l'idée d'introduire la probabilité conjointe (JD) définie par :

$$\lambda_k^i(m) = P(d_k = i, S_k = m|\mathbf{R}_1^N) \tag{3.8}$$

donc l'APP d'un bit décodé d_k, peut s'exprimer en fonction de la JD et est égale à :

$$P(d_k = i|\mathbf{R}_1^N) = \sum_{m=0}^{2^M-1} \lambda_k^i(m), \tag{3.9}$$

où $i = 1, 0$ et la sommation est sur les 2^Métats du codeur.

En remplaçant l'expression de la probabilité conjointe dans l'équation (3.7), nous obtenons :

$$LLR(d_k) = \log_e \left[\frac{\sum_{m=0}^{2^M-1} \lambda_k^1(m)}{\sum_{m=0}^{2^M-1} \lambda_k^0(m)} \right] \tag{3.10}$$

Le décodeur MAP peut faire la décision sur le bit décodé en comparant le $LLR(d_k)$ à un seuil égal à zéro :

$$\text{si } LLR(d_k) \geq 0, \text{le bit décodé est 1} \tag{3.11}$$

$$\text{si } LLR(d_k) < 0, \text{le bit décodé est 0} \tag{3.12}$$

Notons que l'expression de la probabilité conjointe (JD) diffère d'un auteur à un autre. Dans l'algorithme d'origine BJRC [BCJ74], JD a été définie sous forme de deux équations :

$$\delta_k(m) = P(S_k = m|\mathbf{R}_1^N) \tag{3.13}$$

$$\sigma_k(m', m) = P(S_{k-1} = m', S_k = m, \mathbf{R}_1^N) \tag{3.14}$$

En analysant, la deuxième équation (3.14), nous remarquons que ceci revient à calculer la JD en la divisant par une constante. En effet, nous travaillons dans un espace binaire donc le passage d'un état à un autre (nous supposons que le codeur est représenté sous forme de treillis) est provoqué par un bit d'entrée. En supposons que le passage de l'état $S_{k-1} = m'$ à l'état $S_k = m$ est provoqué par le bit $(d_{k-1} = i)$ et en appliquant ceci à σ_k nous obtenons :

$$\begin{aligned} \sigma_k(m', m) &= P(S_{k-1} = m', S_k = m, \mathbf{R}_1^N) \\ &= \frac{P(S_{k-1} = m', S_k = m | \mathbf{R}_1^N)}{P(\mathbf{R}_1^N)} \\ &= \frac{P(d_{k-1} = i, S_k = m | \mathbf{R}_1^N)}{P(\mathbf{R}_1^N)} \\ &= \frac{\lambda_{k-1}^i(m)}{P(\mathbf{R}_1^N)} \end{aligned} \tag{3.15}$$

La majorité des auteurs ont adopté la définition de la probabilité conjointe que nous avons présenté en premier (équation (3.8)). En revanche le développement de la JD afin de simplifier le LLR est beaucoup plus diversifié. Pour notre part, nous avons adopté le dernier développement utilisé par [Pie97]. Évidemment, la façon dont l'expression est développée affectera le nombre d'opérations ou plus c.à.d la complexité de l'algorithme MAP. Nous allons essayer d'analyser les versions de l'algorithme MAP les plus utilisées. A l'origine du MAP est l'article phare paru en 1993 [BGT93]. Les auteurs de ce dernier article ont adopté une version modifiée de l'algorithme original BJRC. Cette version a été appelée BJRC modifiée du fait qu'elle a été adoptée en particulier pour le codes récursifs convolutionnels (RSC).

Dans un premier temps, nous allons décrire la dernière version de l'algorithme MAP que nous avons utilisée. Ensuite, nous allons présenter brièvement les quelques versions les plus utilisées de l'algorithme tout en essayant de faire un parallèle avec la version que nous avons adopté.

3.4 L'Algorithme MAP Utilisé

3.4.1 Expression de la probabilité conjointe

En notant que les événements $\mathbf{R}_k$ sont indépendants et en utilisant des relations les plus élémentaires en probabilité, nous pouvons développer l'expression de la probabilité conjointe sous la forme suivante :

$$\begin{aligned}
\lambda_k^i(m) &= P(d_k = i, S_k = m|\mathbf{R}_1^N) \\
&= \frac{P(d_k = i, S_k = m, \mathbf{R}_1^{k-1}, \mathbf{R}_k^N)}{P(\mathbf{R}_1^N)} \\
&= P(\mathbf{R}_1^{k-1}|d_k = i, S_k = m, \mathbf{R}_k^N) \times \frac{P(d_k = i, S_k = m, \mathbf{R}_k^N)}{P(\mathbf{R}_1^N)} \\
&= P(\mathbf{R}_1^{k-1}|d_k = i, S_k = m, \mathbf{R}_k^N) \times \frac{P(d_k = i, S_k = m, \mathbf{R}_k, \mathbf{R}_{k+1}^N)}{P(\mathbf{R}_1^N)} \\
&= P(\mathbf{R}_1^{k-1}|d_k = i, S_k = m, \mathbf{R}_k^N) \times P(\mathbf{R}_{k+1}^N|d_k = i, S_k = m, \mathbf{R}_k) \\
&\quad \times \frac{P(d_k = i, S_k = m, \mathbf{R}_k)}{P(\mathbf{R}_1^N)}
\end{aligned} \tag{3.16}$$

nous allons définir la métrique d'état en avant (FSM) par

$$\alpha_k(m) = P(\mathbf{R}_1^{k-1}|d_k = i, S_k = m, \mathbf{R}_k^N) \tag{3.17}$$

or les événements produits après l'instant k n'influencent pas les événements qui les ont précédés, donc l'équation précédente se réduit à

$$\alpha_k(m) = P(\mathbf{R}_1^{k-1}|S_k = m) \tag{3.18}$$

De même nous définissons aussi la métrique d'état en arrière BSM comme :

$$\beta_k(m) = P(\mathbf{R}_k^N|S_k = m) \tag{3.19}$$

Nous définissons la métrique de branche (BM) par :

$$\delta_k^i(m) = P(d_k = i, S_k = m, \mathbf{R}_k) \tag{3.20}$$

En remplaçant les expressions des FSM, BSM et BM dans l'équation (3.16) :

$$\begin{aligned} \lambda_k^i(m) &= \frac{\alpha_k(m) \times P(\mathbf{R}_{k+1}^N | d_k = i, S_k = m, \mathbf{R}_k) \times \delta_k^i(m)}{P(\mathbf{R}_1^N)} \\ &= \frac{\alpha_k(m) \times P(\mathbf{R}_{k+1}^N | S_{k+1} = f(i,m)) \times \delta_k^i(m)}{P(\mathbf{R}_1^N)} \\ &= \frac{\alpha_k(m) \times \beta_{k+1}(f(i,m)) \times \delta_k^i(m)}{P(\mathbf{R}_1^N)} \end{aligned} \tag{3.21}$$

En connaissant l'état du codeur $S_k = m$ et le bit d'entrée $d_k = i$ à l'instant k, l'état successeur S_{k+1} peut se noter sous la forme $S_{k+1} = f(i,m)$. L'égalité précédente signifie que l'état précédent de S_{k+1} est l'état m et que le bit d'entrée à l'instant k est $d_k = i$.

Finalement, en remplaçant l'expression de la probabilité conjointe dans l'équation (3.10), nous obtenons :

$$LLR(d_k) = \log \left[\frac{\sum_{m=0}^{2^M-1} \alpha_k(m) \times \beta_{k+1}(f(1,m)) \times \delta_k^1(m)}{\sum_{m=0}^{2^M-1} \alpha_k(m) \times \beta_{k+1}(f(0,m)) \times \delta_k^0(m)} \right] \tag{3.22}$$

D'après l'equation (3.22), le LLR est en fonction des trois expressions définies précédemment : FSM, BSM et BM. Donc, il suffit de calculer ces trois expressions afin de fournir une valeur pour le LLR. Nous allons d'abord simplifier l'expression de la FSM ensuite celle de la BSM et finalement celle de la métrique de branche (BM).

3.4.2 Expression de la métrique d'état en avant (FSM)

Nous pouvons remarquer que la FSM à l'instant k dépend de la séquence reçue $\mathbf{R}_1^k$ qui précède l'instant k (ou au plus jusqu'à l'instant k). En utilisant la règle de Bayes, nous pouvons exprimer la métrique d'état en avant sous la forme :

$$\begin{aligned}
\alpha_k(m) &= P(\mathbf{R}_1^{k-1}|S_k = m) \\
&= \sum_{m'=0}^{2^M-1}\sum_{j=0}^{1} P(d_{k-1} = j, S_{k-1} = m', \mathbf{R}_1^{k-1}|S_k = m) \\
&= \sum_{m'=0}^{2^M-1}\sum_{j=0}^{1} P(\mathbf{R}_1^{k-2}|S_k = m, d_{k-1} = j, S_{k-1} = m', \mathbf{R}_{k-1}) \\
&\quad \times P(d_{k-1} = j, S_{k-1} = m', \mathbf{R}_{k-1}|S_k = m)
\end{aligned}$$

Sachant que le passage de l'état $S_{k-1} = m$ a été provoqué par le bit $d_{k-1} = j$, nous pouvons compacter ces informations sous la forme $S_{k-1} = b(j, m)$. Ceci signifie que l'état S_{k-1} est l'état qui précède l'état $S_k = m$ sachant que le bit d'entrée au codeur à l'instant $(k-1)$ est égale à j.

$$\begin{aligned}
\alpha_k(m) &= \sum_{j=0}^{1} P(\mathbf{R}_1^{k-2}|S_{k-1} = b(j,m)) P(d_{k-1} = j, S_{k-1} = b(j,m), \mathbf{R}_{k-1}) \\
&= \sum_{j=0}^{1} \alpha_{k-1}(b(j,m)) \delta_{k-1}^{j}(b(j,m)) \qquad (3.23)
\end{aligned}$$

Nous pouvons déduire que la métrique d'état à l'instant k s'exprime en fonction de la FSM à l'instant $k-1$ d'où la nomination "en avant". En effet, il faut connaître la valeur initiale à l'instant $k = 0$ afin d'avancer aux instants suivants.

La métrique d'état en avant peut être schématisée sous forme graphique comme à la figure (3.1).

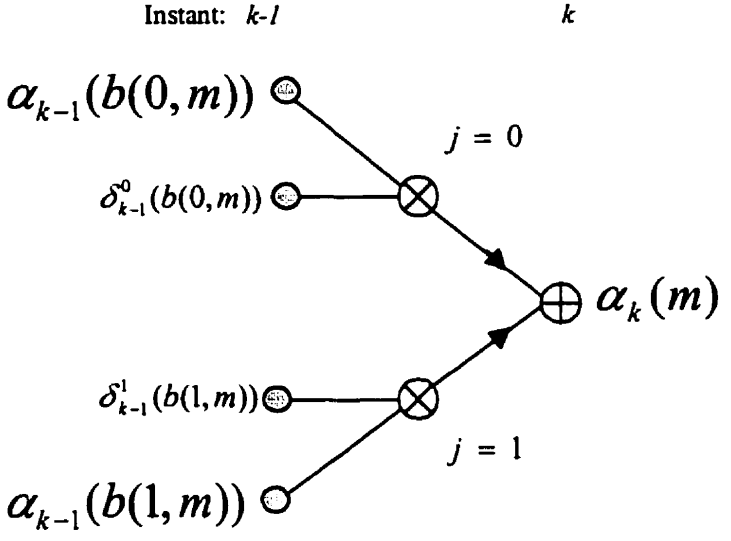

Figure 3.1: Représentation graphique de la métrique d'état en avant.

3.4.3 Expression de la métrique d'état en arrière (BSM)

Contrairement à la FSM, la métrique d'état en arrière à l'instant k dépend de la séquence reçue après cet instant k (ou à partir de l'instant k) et jusqu'à recevoir la totalité de la séquence (instant N). En utilisant encore la règle de Bayes, nous pouvons exprimer la métrique d'état en arrière (BSM) sous la forme :

$$\begin{aligned}
\beta_k(m) &= P(\mathbf{R}_k^N | S_k = m) \\
&= \sum_{m'=0}^{2^M-1} \sum_{j=0}^{1} P(d_k = j, S_{k+1} = m', \mathbf{R}_k^N | S_k = m) \\
&= \sum_{m'=0}^{2^M-1} \sum_{j=0}^{1} P(\mathbf{R}_{k+1}^N | S_k = m, d_k = j, S_{k+1} = m', \mathbf{R}_k) \\
&\quad \times P(d_k = j, S_{k+1} = m', \mathbf{R}_k | S_k = m)
\end{aligned}$$

$$\begin{aligned}\beta_k(m) &= \sum_{j=0}^{1} P(\mathbf{R}_k^N | S_{k+1} = f(j,m)) P(d_k = j, S_k = m, \mathbf{R}_k) \\ &= \sum_{j=0}^{1} \delta_k^j(m) \beta_{k+1}(f(j,m)) \end{aligned} \tag{3.24}$$

Nous pouvons déduire que la BSM à l'instant k s'exprime en fonction de la BSM à l'instant $k+1$ d'où la nomination "en arrière". En effet, il faut connaître la valeur initiale à l'instant $k = N$ afin de reculer dans le treillis et de calculer les BSM précédentes.

La métrique d'état en arrière peut illustrée être aussi schématisée sous forme graphique (à la figure (3.2)). Nous remarquons une similitude entre l'architecture de l'algorithme de Viterbi et celle du MAP. En effet, en réexaminant les figures (3.1) et (3.2), nous déduisons que la métrique de branche est multipliée par la métrique d'état (en avant ou en arrière) au lieu d'être additionnée comme dans le cas de l'VA. Nous pouvons aussi remarquer qu'au lieu de minimiser les métriques du chemin dans le cas de l'VA. Les métriques sont additionnées dans le cas de l'algorithme MAP. Par conséquent, l'opération Add-Compare-Select (ACS) dans l'VA devient l'opération Multiply-Add (MA) dans le cas de l'algorithme MAP.

3.4.4 Calcul de la métrique de branche (BM) :

Comme nous l'avons indiqué précédemment le calcul de la métrique de branche est nécessaire afin de calculer le logarithme du rapport de vraisemblance. D'après l'expression de δ_k^i (equation(3.20)), la BM à l'instant k dépend seulement de l'instant même et non plus des instants précédents ou suivants d'où la nomination métrique

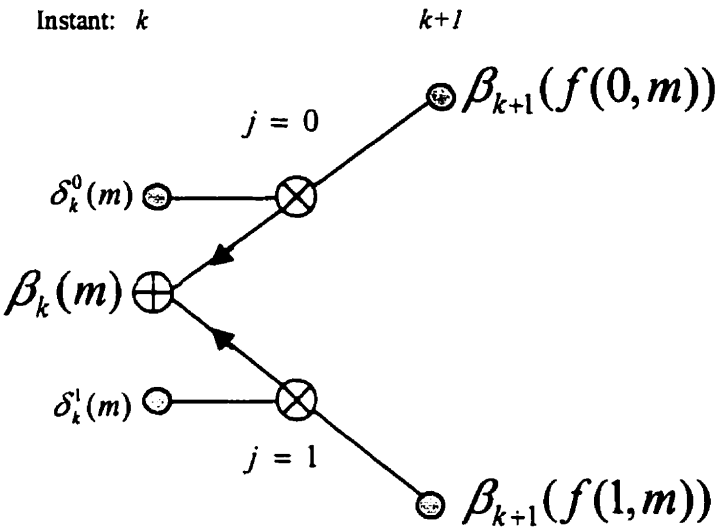

Figure 3.2: Représentation graphique de la métrique d'état en arrière.

de branche.

$$\begin{aligned}\delta_k^i(m) &= P(d_k = i, S_k = m, \mathbf{R}_k) \\ &= P(\mathbf{R}_k | d_k = i, S_k = m) \times P(d_k = i, S_k = m) \\ &= P(\mathbf{R}_k | d_k = i, S_k = m) \times P(S_k = m | d_k = i) \times P(d_k = i) \end{aligned} \quad (3.25)$$

en remplaçant $\mathbf{R}_k = (r_k^s, r_k^p)$ dans l'équation précédente nous obtenons :

$$\begin{aligned}\delta_k^i(m) &= p(r_k^s | d_k = i, S_k = m) \times p(r_k^p | d_k = i, S_k = m) \\ &\quad \times P(S_k = m | d_k = i) \times P(d_k = i) \end{aligned} \quad (3.26)$$

les deux premiers termes de la métrique de branche dépendent des symboles recueillis à la sortie du canal. Donc le type canal utilisé affecte l'expression de la BM. Tout d'abord, nous allons exprimer la BM dans un canal à bruit blanc additif Gaussien (AWGN). Ensuite nous allons présenter les modifications à apporter dans le cas d'un canal de Rayleigh.

3.4.4.1 Canal AWGN

Le modèle de ce canal a été défini au chapitre précédent (équation (2.7)). Nous pouvons exprimer respectivement la densité de probabilité (pdf) de l'information reçue et la pdf de l'information de parité reçue comme suit :

$$p(r_k^s|d_k = i, S_k = m) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(r_k^s-(2i-1))^2} \tag{3.27}$$

et

$$p(r_k^p|d_k = i, S_k = m) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(r_k^p-(2c_m^i-1))^2} \tag{3.28}$$

où c_m^i est le bit codé sachant $d_k = i$ et $S_k = m$. En remplaçant les deux précédentes expressions dans l'équation (3.26), la métrique de branche devient :

$$\begin{aligned}
\delta_k^i(m) &= \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(r_k^s-(2i-1))^2} \times \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(r_k^p-(2c_m^i-1))^2} \\
&\quad \times \tfrac{1}{2^M} \times P(d_k = i) \\
&= \frac{1}{2^M \times 2\pi\sigma^2} \times P(d_k = i) \\
&\quad \times e^{-\frac{1}{2\sigma^2}\left[(r_k^s-(2i-1))^2+(r_k^p-(2c_m^i-1))^2\right]} \\
&= \frac{1}{2^{M+1}\pi\sigma^2} \times P(d_k = i) \\
&\quad \times e^{-\frac{1}{2\sigma^2}\left[(r_k^s)^2-2r_k^s(2i-1)+\underbrace{(2i-1)^2}_{1}+(r_k^p)^2-2r_k^p(2c_m^i-1)+\underbrace{(2c_m^i-1)^2}_{1}\right]} \\
&= \frac{1}{2^{M+1}\pi\sigma^2} \times P(d_k = i) \\
&\quad \times e^{-\frac{1}{2\sigma^2}\left[-4r_k^s i-4r_k^p c_m^i+(r_k^s)^2+(r_k^p)^2+2(r_k^s+r_k^p)+2\right]} \\
&= \frac{1}{2^{M+1}\pi\sigma^2} \times e^{-\frac{1\left[(r_k^s)^2+(r_k^p)^2+2(r_k^s+r_k^p)+2\right]}{2\sigma^2}} \\
&\quad \times P(d_k = i) \\
&\quad \times e^{\frac{2}{\sigma^2}\left[r_k^s i+r_k^p c_m^i\right]} \\
&= C(M, \sigma, r_k^s, r_k^p) \times P(d_k = i) \times e^{\frac{2}{\sigma^2}\left[r_k^s i+r_k^p c_m^i\right]}
\end{aligned} \tag{3.29}$$

où la variable $C()$ ne dépend pas de la variable i et de l'état m. $C()$ peut être supprimée car en faisant le LLR, la constante $C()$ disparaîtra.

3.4.4.2 Canal de Rayleigh

Dans le cas où le canal est modélisé suivant la loi de Rayleigh (voir équation (2.8)), nous pouvons distinguer deux cas. Dans le premier nous considérons que le gain du canal à chaque instant est connu. Par contre, dans le deuxième cas, la valeur moyenne du gain est connue et non pas la valeur instantanée du gain.

Avec connaissance du gain du canal (Side Information) La fonction de densité de probabilité peut se calculer à partir de la fonction de répartition F_R, encore appelé fonction cumulative (cdf).

$$p(r_k^s|d_k = i, S_k = m) = \frac{d}{dr_k^s} F_R(r_k^s|d_k = i, S_k = m) \tag{3.30}$$

or

$$\begin{aligned} F_R(r_k^s|d_k = i, S_k = m) &= \Pr(R < r_k^s|d_k = i, S_k = m) \\ &= \Pr([a_k(2d_k - 1) + n_k] < r_k^s|d_k = i, S_k = m) \\ &= \Pr([a_k(2i - 1) + n_k] < r_k^s) \\ &= \Pr(n_k < [r_k^s - a_k(2i - 1)]) \\ &= F_{n_k}(r_k^s - a_k(2i - 1)) \end{aligned} \tag{3.31}$$

$$\Longrightarrow p(r_k^s|d_k = i, S_k = m) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(r_k^s - a_k(2i-1))^2} \tag{3.32}$$

de la même façon nous pouvons démontrer que

$$p(r_k^p|d_k = i, S_k = m) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(r_k^p - a_k(2c_m^i - 1))^2} \tag{3.33}$$

En raisonnant de la même façon que dans la section (3.4.4.1), l'expression de la probabilité conjointe sera de la forme :

$$\delta_k^i(m) = C'(M, \sigma, a_k, r_k^s, r_k^p) \times P(d_k = i) \times e^{\frac{2}{\sigma^2} a_k \left[r_k^s i + r_k^p c_m^i\right]} \tag{3.34}$$

où $C'()$ ne dépend pas de la variable i et de l'état m, donc nous pouvons l'enlever car comme nous l'avons indiqué précédement $C'()$ disparaîtra en calculant le LLR.

Sans Connaissance du gain (without Side Information) La fonction pdf de l'information systématique est donnée par [Hal96][HWi98] :

$$\begin{aligned} p(r_k^s | d_k = i, S_k = m) &= \int_0^\infty p_A(a_k) p(r_k^s | d_k = i, S_k = m, a_k) da_k \\ &\approx N(E_A(a_k) \times (2i-1)\sqrt{E_s}, N_0/2) \end{aligned} \tag{3.35}$$

où $N(moy, var)$ est une loi normale de moyenne moy et de variance var. $E_A(a_k)$ est égale à 0.8862. Dans notre étude, nous avons considéré $E_s = 1$ et $N_0 = \frac{\sigma^2}{2}$ donc l'expression de $p(r_k^s | d_k = i, S_k = m)$ est

$$p(r_k^s | d_k = i, S_k = m) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2} m_A (r_k^s - (2i-1))^2} \tag{3.36}$$

où $m_A = 0.8862$. Finalement l'expression de la probabilité conjointe sera :

$$\delta_k^i(m) = C''(M, \sigma, r_k^s, r_k^p) \times P(d_k = i) \times e^{\frac{2}{\sigma^2} m_A \left[r_k^s i + r_k^p c_m^i\right]} \tag{3.37}$$

où $C''()$ est une constante qui ne dépend pas de m ni de i donc nous pouvons la supprimer.

3.4.5 Algorithme Log-MAP Utilisé

L'algorithme Log-MAP comme son nom l'indique consiste à prendre le logarithme Néperien des trois métriques (FSM, BSM et BM) tel que :

$$A_k(m) = -\log \alpha_k(m) \tag{3.38}$$

$$B_k(m) = -\log \beta_k(m) \tag{3.39}$$

$$D_k^i(m) = -\log \delta_k^i(m) \tag{3.40}$$

En remplaçant les nouvelles expressions dans l'expression du LLR :

$$\begin{aligned} LLR(d_k) &= \log\Big(\sum_{m=0}^{2^M-1} \alpha_k(m)\delta_k^1(m)\beta_{k+1}(f(1,m))\Big) \\ &\quad - \log\Big(\sum_{m=0}^{2^M-1} \alpha_k(m)\delta_k^0(m)\beta_{k+1}(f(0,m))\Big) \\ &= \log\Big(\sum_{m=0}^{2^M-1} e^{-A_k(m)}e^{-D_k^1(m)}e^{-B_k(f(1,m))}\Big) \\ &\quad - \log\Big(\sum_{m=0}^{2^M-1} e^{-A_k(m)}e^{-D_k^0(m)}e^{-B_k(f(0,m))}\Big) \\ &= \log\Big(\sum_{m=0}^{2^M-1} e^{-(A_k(m)+D_k^1(m)+B_k(f(1,m))}\Big) \\ &\quad - \log\Big(\sum_{m=0}^{2^M-1} e^{-A_k(m)+D_k^0(m)+B_k(f(0,m))}\Big) \end{aligned} \tag{3.41}$$

Afin de simplifier l'expression du LLR, nous allons introduire l'opérande $\mathbf{E}$ entre deux variables a_1 et a_2 défini comme suit [Bar96][Hag96][RHV97] :

$$\begin{aligned} a_1 \,\mathbf{E}\, a_2 &\equiv -\log\left(e^{-a_1} + e^{-a_2}\right) \\ &= \min\left(a_1, a_2\right) \text{-} \log(1 + e^{-|a_1 - a_2|}) \end{aligned} \tag{3.42}$$

L'équation précédente peut se généraliser sous la forme suivante :

$$\begin{aligned} \underset{j=0}{\overset{l}{\mathbf{E}}} a_j &= a_0 \mathbf{E}\, a_1 \mathbf{E} \cdots \mathbf{E}\, a_{l-1} \mathbf{E}\, a_l \\ &= -\log\left(\sum_{j=0}^{l} e^{a_j}\right) \end{aligned} \tag{3.43}$$

En appliquant la relation (3.43) à l'équation (3.41) :

$$\begin{aligned} LLR(d_k) &= \underset{m=0}{\overset{2^M-1}{\mathbf{E}}} - [A_k(m)+D_k^1(m)+B_k(f(1,m)] \\ &\quad - \underset{m=0}{\overset{2^M-1}{\mathbf{E}}} - [A_k(m)+D_k^0(m)+B_k(f(0,m)] \\ &= \underset{m=0}{\overset{2^M-1}{\mathbf{E}}} [A_k(m)+D_k^0(m)+B_k(f(0,m)] \\ &\quad - \underset{m=0}{\overset{2^M-1}{\mathbf{E}}} [A_k(m)+D_k^1(m)+B_k(f(1,m)] \end{aligned} \tag{3.44}$$

Reste à exprimer les variables A_k et B_k ; il suffit de prendre le logarithme des équations (3.19) et (3.18) :

$$\begin{aligned} A_k(m) &= -\log \alpha_k(m) \\ &= -\log \sum_{j=0}^{1} \alpha_{k-1}(b(j,m)) \delta_{k-1}^j(b(j,m)) \\ &= -\log \sum_{j=0}^{1} e^{A_{k-1}(b(j,m))} e^{D_{k-1}^j(b(j,m))} \\ &= -\log \sum_{j=0}^{1} e^{A_{k-1}(b(j,m))+D_{k-1}^j(b(j,m))} \\ &= \underset{j=0}{\overset{1}{\mathbf{E}}} A_{k-1}(b(j,m))+D_{k-1}^j(b(j,m)) \end{aligned} \tag{3.45}$$

$$
\begin{aligned}
B_k(m) &= -\log \beta_k(m) \\
&= -\log \sum_{j=0}^{1} \delta_k^j(m) \beta_{k+1}(f(j,m)) \\
&= -\log \sum_{j=0}^{1} e^{D_k^j(m)} e^{B_{k+1}(f(j,m))} \\
&= -\log \sum_{j=0}^{1} e^{D_k^j(m)+B_{k+1}(f(j,m))} \\
&= \mathop{\mathrm{E}}_{j=0}^{1} D_k^j(m)+B_{k+1}(f(j,m)) \qquad (3.46)
\end{aligned}
$$

3.4.5.1 Les Etapes de l'implémentation l'algorithme MAP

Voici les principales étapes de l'algorithme de décodage :

- Étape 1 : A partir de l'instant $k = 0$, pour $i = 0, 1$ et $m = 0, ..., M-1$ calculer la métrique de branche $D_k^j(m)$ suivant l'expression $\frac{2C_a}{\sigma^2}[r_k^s i + r_k^p c_m^i]$ (où C_a est une constante qui dépend du type de canal utilisé; elle est égale à 1 lorsque le canal est AWGN) pour tous les symboles reçus et les sauvegarder dans un tableau.
- Étape 2 : Initialisation de la variable $A_k(m)$ telle que $A_0(0) = 0$ et $A_0(m) = -\infty$, pour tout $m \neq 0$; et calcul de la métrique d'état en avant à partir de l'instant $k = 1$ et pour tout les états $(m = 0, ..., M-1)$ en utilisant l'équation (3.45).
- Étape 3 : Initialisation de la variable $B_k(m)$ telle que $B_N(0) = 0$ et $B_N(m) = -\infty$, pour tout $m \neq 0$; et calcul de la métrique d'état en arrière à partir de l'instant $k = N-1$ et pour tout les états $(m = 0, ..., M-1)$ en utilisant l'équation (3.46).

3.5 Quelques Versions de l'algorithme MAP

Après avoir présenté la version de l'algorithme MAP que nous avons utilisé dans notre étude, nous allons décrire brièvement différentes versions de l'algorithme MAP.

3.5.1 Première version Log-MAP

Nous avons mentionné que la probabilité conjointe peut se développer de façons différentes et donc ceci engendre des légères modifications dans la définition des entités BSM et FSM. Dans la version développée par [PBa96] et adoptée par Naoufel [Bou97], les FSM et BSM sont définies respectivement par :

$$\alpha_k^i(m) = P(d_k = i, S_k = m, R_1^{k-1}) \tag{3.47}$$

et

$$\beta_k^i(m) = P(R_{k+1}^N | d_k = i, S_k = m) \tag{3.48}$$

Nous pouvons remarquer que la différence avec la version utilisée ici est l'introduction d'un bit systématique dans les expressions des FSM et BSM. En suivant un raisonnement similaire à celui de la section précédente [PBa96] et en introduisant les termes suivants :

$$\delta_k^i(m) = P(R_k | d_k = i, S_k = m)/2, \tag{3.49}$$

$$A_k^i(m) = -\log \alpha_k^i(m) \tag{3.50}$$

$$B_k^i(m) = -\log \beta_k^i(m) \tag{3.51}$$

$$D_k^i(m) = -\log \delta_k^i(m) \tag{3.52}$$

Le LLR s'exprime sous la forme :

$$LLR(d_k) = \mathop{\mathrm{E}}_{m=0}^{2^M-1} - [A_k^1(m) + B_k^1(m)] - \mathop{\mathrm{E}}_{m=0}^{2^M-1} - [A_k^0(m) + B_k^0(m)] \tag{3.53}$$

où

$$A_k^i(m) = D_{k-1}^i(m) + \overset{1}{\underset{j=0}{\mathbf{E}}} A_{k-1}^j(b(j,m)) \tag{3.54}$$

$$B_k^i(m) = \overset{1}{\underset{j=0}{\mathbf{E}}} D_{k+1}^j(f(i,m)) + B_{k+1}^j(f(i,m)) \tag{3.55}$$

3.5.2 Log-MAP Sous Optimal

En examinant de nouveau la version Log-MAP, nous pouvons remarquer que nous avons besoin de calculer l'opérande **E** défini dans les équations (3.42) et (3.43). Or d'après l'équation (3.42) l'opération **E** n'est pas linéaire. En effet, elle consiste en une comparaison (calcul du minimum) et deux autres calculs plus complexes (calcul de la valeur absolue et du logarithme). L'algorithme Log-MAP sous optimal consiste à négliger le deuxième terme de l'équation (3.42) comme suit :

$$a_1 \,\mathbf{E}\, a_2 \approx \min(a_1, a_2) \tag{3.56}$$

Cette approximation est surtout vraie lorsque l'écart entre a_1 et a_2 est grand. Donc, au niveau de l'implémentation matérielle l'algorithme MAP est très simplifié, mais malheureusement, ceci s'accompagne d'une dégradation substantielle des performances [RHV97].

3.5.3 Sliding Windows BJRC

Cette version s'inspire de l'algorithme de Viterbi tronqué (AVT). En effet, l'obligation d'attendre la réception de toute la séquence avant de commencer à décoder, complique l'opération de décodage surtout au niveau de la latence. Dans la but de réduire cette latence, Benedetto [BMD96] a proposé en s'inspirant de l'AVT de commencer à décoder à partir d'un certain délai fixé à l'avance (ce délai est un multiple de la longueur de contrainte du codeur K). Le décodeur fera la décision sur le bit reçu

après ce délai D [BMD96] ne précise pas le rapport entre D et K, mais Viterbi [Vit98] a utilisé une approche similaire et affirme qu'un délai égal à sept fois la longueur de contrainte est suffisant pour obtenir les même performances d'erreur ou au pire, des pertes négligeables. Cette version est très pratique pour l'implémentation matérielle. En outre la réduction au niveau de la latence, elle requiert moins d'espace mémoire et aussi évite le problème de terminaison du treillis.

3.6 Complexité

La complexité est parmi les facteurs determinants dans le choix de la version de l'algorithme Log-MAP utilisée. Donc, il nous sera utile de comparer rapidement la complexité des divers versions proposées. Nous allons d'abord dresser les nombres et types d'opérations nécessaires pour la version originale [BGT93] dont nous rappelons les équations de l'algorithme à titre d'informations :

$$\alpha_k^i(m) = \frac{\sum_{m'}\sum_{j=0}^{1}\delta_k^i(m',m)\alpha_{k-1}^j(m')}{\sum_{m}\sum_{m'}\sum_{i=0}^{1}\sum_{j=0}^{1}\delta_k^i(m',m)\alpha_{k-1}^j(m')} \tag{3.57}$$

$$\beta_k(m) = \frac{\sum_{m'}\sum_{j=0}^{1}\delta_{k+1}^i(m,m')\beta_{k+1}(m')}{\sum_{m}\sum_{m'}\sum_{i=0}^{1}\sum_{j=0}^{1}\delta_{k+1}^i(m',m)\alpha_k^j(m')} \tag{3.58}$$

où la métrique de branche est égale à :

$$\delta_k^i(m',m) = P(R_k|d_k = i, S_k = m, S_{k-1} = m') \tag{3.59}$$

Ensuite nous allons comparer la première version Log-MAP que nous avons utilisée au départ avec celle à laquelle nous nous sommes intéressés plus tard et que nous

avons désignés par l'*Algorithme Log-MAP* utilisé (voir section (3.4)). Revenons à la version proposée par [BGT93][BGl96] et présentons le type et nombre d'opérations informatiques nécessaire pour décoder une séquence reçue R_k [Mad96] :

- Pour α_k^i :

  Calcul : 2^Madditions et 2×2^Mmultiplications

  Normalisation : 2^M divisions

- Pour β_k : idem que α_k^i

- Pour δ_k^i : 1 exponentiel et 3 multiplications

- Pour le calcul du rapport de vraisemblance : 1 logarithme, 1 division, $2\times(2^M-1)$ additions et (2×2^M) multiplications.

Le total des opérations est :

- 3 exponentielles

- $(4\times2^M-2)$ additions

- $(6\times2^M+3)$ multiplications

- $(2\times2^M+1)$ divisions

Nous pouvons remarquer que le décodage d'une seule séquence reçue R_k nécessite 10×2^M multiplications et divisions . Or c'est précisément, ce type d'opérations qui ralentit le temps de décodage au niveau des simulations informatiques mais surtout pose d'énormes problèmes au niveau de l'implémentation [RHV97].

Contrairement à la version de Berrou, la version Log-MAP supprime les opérations de division et de multiplication et les remplacent par des additions, soustractions et le calcul de l'opérande **E**. Donc, il reste à connaître la complexité de l'opérande **E** afin de dresser une idée plus précise sur la nature de la simplification. Révisant l'équation (3.42) on pose :

$$f_c(|z|) = -log(1+e^{-|z|}) \tag{3.60}$$

où la fonction f_c est appelée la fonction de correction. La fonction $f_c(|z|)$ décroît rapidement vers 0 en fonction de z. Donc, il suffit de la calculer à l'avance pour un nombre fini de valeurs afin de couvrir toute la plage sans erreur. Ainsi, le calcul de la fonction f_c au niveau de l'implémentation se réduit à une lecture d'une mémoire. Donc l'évaluation de l'opérande E se réduit à une comparaison de deux valeurs et une lecture dans la mémoire. Ceci est évidemment beaucoup plus simple à réaliser qu'une division.

Nous avons résumé le type et le nombre d'opérations des deux versions Log-MAP que nous avons utilisées dans les tableaux (3.1) et (3.2). La dernière ligne horizontale des tableaux représente le nombre total d'opérations. En comparant les deux tableaux, nous pouvons déduire que la version du Log-MAP utilisée (Table (3.2)) requiert 20% de moins de calculs concernant l'opérande **E**. Mais, en pratique, l'écart se réduit en remarquant que dans le calcul de $B_k^i(m)$ (équation(3.55)) il suffit de calculer la moitié des ces paramètres et ensuite déduire l'autre moitié de la relation :

$$B_k^1(m) = B_k^0(m') \qquad \text{où} \qquad m' = b(0, f(1, m))$$

Donc, au niveau des simulations, l'effort de calcul est presque identique mais avec un léger avantage pour la version utilisée. Par contre, au niveau de l'implémentation, la version utilisée ici est largement plus efficace [Pie97].

3.7 Conclusion

Dans ce chapitre, nous avons décrit l'algorithme MAP et plusieurs de ses versions. Mentionnons que d'autres versions existent, mais malheureusement, nous ne pouvons pas toutes les décrire. Néanmoins, la version proposée par Benedetto [BDM96] et nommée SISO (Soft In Soft Out) donne une autre approche originale à l'algorithme MAP. Lorsque le principe itératif est appliqué au décodeur MAP, ceci nous conduit tout droit au décodage Turbo que nous allons présenter au chapitre suivant.

Version1	type d'opérations		
Paramétres	Addition	Soustraction	fonction E
$A_k^i(m)$	$2x2^M$	0	2^M
Normalisation de $A_k^i(m)$	0	$2x2^M$	0
$B_k^i(m)$	$2x2^M$	0	$2x2^M$
Normalisation de $B_k^i(m)$	0	$2x2^M$	0
$D_k^i(m)$	1	0	0
$LLR(d_k)$	$2x2^M$	1	$2x(2^M-1)$
Total	$(6x2^M+1)$	$(4x2^M+1)$	$(5x2^M-2)$

Tableau 3.1: Première Version Log-MAP

Version2	type d'opérations		
Paramétres	Addition	Soustraction	fonction E
$A_k(m)$	$2x2^M$	0	2^M
Normalisation de $A_k(m)$	0	2^M	0
$B_k(m)$	$2x2^M$	0	2^M
Normalisation de $B_k(m)$	0	2^M	0
$D_k(m)$	1	0	0
$LLR(d_k)$	$4x2^M$	1	$2x(2^M-1)$
Total	$(8x2^M+1)$	$(2x2^M+1)$	$(4x2^M-2)$

Tableau 3.2: Version Log-MAP utilisée

Chapitre 4

Décodage Itératif (Turbo)

4.1 Introduction

Nous avons présenté les diverses versions de l'algorithme MAP qui pourront être utilisées pour le décodage des Codes Turbo sans toutefois traiter l'architecture d'un décodeur Turbo. Or, en examinant soigneusement le codeur turbo (voir figure(2.7)), nous pourrions nous attendre à avoir autant de décodeurs MAP que de codeurs RSC. Les deux décodeurs sont en série et sont séparés par des entrelaceurs du même type que ceux utilisés au codage. Chaque décodeur va se concentrer surtout sur les symboles provenant du codeur qui lui est associé. Les deux décodeurs vont s'échanger de l'information douce qui inclue une estimation de la vraisemblance sur l'information commune (symbole systématique) au codage. Ce va et vient sous forme d'échange de l'information qui va se répéter jusqu'à l'obtention des performances requises nous amène tout droit vers le principe itératif.

Dans les pages qui suivent, nous allons d'abord développer et préciser le type d'information qui est échangée entre les décodeurs composant le décodeur Turbo. Ensuite une description détaillée du décodeur Turbo sera présentée. L'information échangée est considérée comme une donnée ou variable supplémentaire que le décodeur reçoit. Donc ceci modifie l'expression des paramètres de décodage (BSM, FSM et BM)

mais c'est surtout la métrique de branche (BM) qui va inclure dans son expression cette information échangée entre les décodeurs.

Ce décodeur utilisant l'algorithme MAP a été simulé, les performances des Codes Turbo sont analysées en fonction de plusieurs paramètres (taux de codage, type et taille des entrelaceurs, longueur de contrainte) d'abord dans un canal AWGN et ensuite dans un canal de Rayleigh.

Finalement, comme nous l'avons mentionné dans le chapitre précédent, le décodeur MAP nécessite l'estimation de la variance du bruit dans le canal. Donc, diverses méthodes d'estimation de la variance du bruit dans le canal sont présentées et simulées.

4.2 Information extrinsèque

Comme nous l'avons mentionné dans le chapitre précédent, le décodeur MAP fournit une décision douce sur chaque bit décodé. Or dans le cas des Codes Turbo, le codeur fournit un bit systématique qui représente le bit de donnée et deux bits de parités issus des deux codeurs. Donc, nous nous attendons à avoir un accès direct de l'état du canal dû à l'information systématique et une information supplémentaire indirecte qui est le résultat de l'envoi des bits de parités. Nous allons examiner ceci de plus près en reprenant l'expression du rapport de vraisemblance et en replaçant la métrique de branche dans l'équation (3.22) :

$$LLR(d_k) = \log \left[\frac{\sum_{m=0}^{2^M-1} \alpha_k(m) \times \beta_{k+1}(f(1,m)) \times C''() \times P(d_k = 1) \times e^{a_k L_c \left[r_k^s + r_k^p c_m^1 \right]}}{\sum_{m=0}^{2^M-1} \alpha_k(m) \times \beta_{k+1}(f(0,m)) \times C''() \times P(d_k = 0) \times e^{a_k L_c \left[r_k^p c_m^0 \right]}} \right]$$

$$
\begin{aligned}
&= \log\left[\frac{P(d_k=1)\times e^{a_k L_c r_k^s}\sum_{m=0}^{2^M-1}\alpha_k(m)\times\beta_{k+1}(f(1,m))\times e^{a_k L_c\left[r_k^p c_m^1\right]}}{P(d_k=0)\times\sum_{m=0}^{2^M-1}\alpha_k(m)\times\beta_{k+1}(f(0,m))\times e^{a_k L_c\left[r_k^p c_m^0\right]}}\right] \\
&= \log\frac{P(d_k=1)}{P(d_k=0)}+\log e^{a_k L_c r_k^s} \\
&\quad +\log\frac{\sum_{m=0}^{2^M-1}\alpha_k(m)\times\beta_{k+1}(f(1,m))\times e^{a_k L_c\left[r_k^p c_m^1\right]}}{\sum_{m=0}^{2^M-1}\alpha_k(m)\times\beta_{k+1}(f(0,m))\times e^{a_k L_c\left[r_k^p c_m^0\right]}} \\
&= L_a(d_k)+a_k L_c r_k^s+L_e(d_k) \qquad (4.1)
\end{aligned}
$$

Dans l'équation (4.1) $L_a(d_k)$ est défini comme le rapport des probabilités a priori et qui est égal à zéro dans le cas où $d_k = 0$ ou 1 sont équiprobables. $L_e(d_k)$ est défini comme l'information extrinsèque du décodeur et représente une information supplémentaire qui n'a pas été exploitée par le décodeur MAP. $L_e(d_k)$ peut être vu comme un terme de correction qui affirme l'information à l'entrée de façon à minimiser la probabilité d'erreur au décodage. L'information extrinsèque est très importante dans le décodage Turbo car elle permet de faire circuler le terme de correction d'un décodeur à un autre. $L_e(d_k)$ est une combinaison d'influence de toutes les entrées douces sauf celle de r_k^s. Le deuxième terme représente la réponse du canal.

4.2.1 Redéfinition de la Métrique de branche

En réexaminant le paragraphe (3.4.4), la métrique de branche se met sous la forme :

$$\delta_k^i(m) = C_k \times P(d_k = i) \times e^{C_a L_c\left[r_k^s i + r_k^p c_m^i\right]} \qquad (4.2)$$

où C_k ne dépend ni de l'état m, ni du bit i; $L_c = \frac{2}{\sigma^2}$

$$C_a = \left\{ \begin{array}{c} \mathbf{1} \text{ pour un canal AWGN} \\ \left\{ \begin{array}{c} \mathbf{a}_k \text{ si le gain du canal est connu} \\ \text{sinon } C_a = \mathbf{m}_A = \mathbf{0.8862} \end{array} \right\} \text{pour un canal de Rayleigh} \end{array} \right\} \quad (4.3)$$

Reprenons l'expression de la métrique de branche et appliquons la version *Log-MAP*

$$\begin{aligned} D_k^i(m) &= -\log \delta_k^i(m) \\ &= -\log(C_k \times P(d_k = i) \times e^{C_a L_c \left[r_k^s i + r_k^p c_m^i\right]}) \\ &= -\log(C_k) - \log(P(d_k = i)) - C_a L_c(r_k^s i + r_k^p c_m^i) \end{aligned} \quad (4.4)$$

$$\text{Posons } \varphi_k^i = P(d_k = i) \Longrightarrow \varphi_k^i = (\varphi_k^0)^{1-i} \times (\varphi_k^1)^i = (\varphi_k^0) \left(\frac{\varphi_k^1}{\varphi_k^0}\right)^i \quad (4.5)$$

En remplaçant $P(d_k = i)$ par φ_k^i dans l'expression de la métrique de branche :

$$\begin{aligned} D_k^i(m) &= -\log(C_k) - \log(\varphi_k^0) - i \log\left(\frac{\varphi_k^1}{\varphi_k^0}\right) - C_a L_c(r_k^s i + r_k^p c_m^i) \\ &= -K_k - (L_a(d_k) + C_a L_c r_k^s) i - C_a L_c r_k^p c_m^i \end{aligned} \quad (4.6)$$

où K_k est une constante et $L_a(d_k)$ est l'*information a priori* ou le logarithme de la probabilité a priori. Dans le cas où le décodage n'est pas itératif c'est à.dire le décodeur ne reçoit pas une information supplémentaire sur le bit qui va être décodé, l'information a priori est nulle comme nous l'avons indiqué précédemment. Or, le principe itératif est justement le fondement des Codes Turbo. Donc le décodeur MAP va recevoir une information a priori qui va lui servir à corriger un nombre supplémentaire d'erreurs que le décodeur n'a pas pu faire au tour précédent. Notons que cette information a priori est obtenue à partir des autres bits décodés (du même bloc) par

le deuxième décodeur MAP.

4.3 Décodeur Turbo

La figure (4.1) illustre le schéma de principe du décodage itératif dans un décodeur Turbo. À la première itération, nous n'avons pas besoin d'ajouter l'*information a priori* $L_a^1(d_k)$ à l'expression $[a_k L_c r_k^s]$. En effet, comme nous l'avons mentionné, les bits sont équiprobables et donc l'information de la probabilité a priori est nulle à la première itération. Après décodage des symboles par le premier décodeur MAP, la sortie est égale à $[a_j L_c r_j^s + L_e^1(d_j)]$ et $[j = k - D]$ où D est le délai du décodeur MAP. Cette donnée est ensuite entrelacée afin de correspondre aux symboles entrelacés par le deuxième codeur (voir figure (2.7)).

Le terme $[a_j L_c r_j^s + L_e^1(d_j)]$ fourni par le premier décodeur alimente le deuxième décodeur MAP. Dans ce cas, l'information extrinsèque $L_e^1()$ fournie par le premier décodeur devient l'*information a priori* L_a^2 pour le deuxième décodeur. Le but de l'entrelacement est de disperser les erreurs arrivées en rafale qui sont difficilement détectable par les décodeurs MAP et Viterbi. L'entrelaceur permet aussi de fournir des observables non correlés aux décodeurs d'une itération à l'autre. Plus la taille de l'entrelaceur est grande plus les erreurs en bloc sont dispersées.

Après avoir acquis une information sur le bit qui va être décodé, le deuxième décodeur MAP va pouvoir corriger un nombre d'erreurs que le premier décodeur n'a pas pu faire. A la sortie du deuxième décodeur et à la première itération nous avons :

$$LLR_2(d_i) = [L_e^1(d_i) + a_i L_c r_i^s + L_e^2(d_i)] \tag{4.7}$$

où $L_e^2(d_i)$ est l'*information extrinsèque* du deuxième décodeur et l'indice i est utilisé afin d'indiquer que le bit a été entrelacé et retardé. Afin d'accroître l'efficacité du premier décodeur, il faut lui fournir une information supplémentaire sur le bit qui sera décodé. Cette information supplémentaire qui est connue sous le nom d'*information a*

priori doit être différente de l'information que le décodeur possède déjà. Donc il faut extraire du deuxième décodeur l'information utilisée précédemment. Or l'information utile pour le premier décodeur est $L_e^2(d_i)$, donc il faut soustraire de $LLR_2(d_i)$ le terme $[L_e^1(d_i)+a_iL_cr_i^s]$ afin d'obtenir l'*information extrinsèque* du deuxième décodeur $L_e^2(d_i)$ qui sera délacée et passée à la prochaine itération sous le nom d'information a priori pour le premier décodeur $[L_a^1(d_{k-\Delta})]$ (où Δ est le délai total de l'itération). Notons que le délacement sert à disperser les erreurs en bloc reçues du deuxième décodeur. Mentionnons que si ces erreurs en bloc sont injectées dans le premier décodeur, la performance d'erreur chutera au lieu de s'améliorer. Après la première itération, le logarithme du rapport de vraisemblance à la sortie du premier décodeur $LLR_1()$ est donné par :

$$LLR_1(d_j) = [L_e^2(d_j) + a_jL_cr_j^s + L_e^1(d_j)] \tag{4.8}$$

Pour une raison similaire appliquée au premier décodeur, l'information fournie au deuxième décodeur est le troisième terme de l'équation précédente $L_e^1(d_j)$ lequel sera entrelacé avant d'être injecté au deuxième décodeur MAP.

Rappelons que Berrou [BGT93] a considéré l'*information extrinsèque* comme une variable Gaussienne mais cette hypothèse s'avére fausse si le nombre d'itérations n'est pas élevée (supérieur à 10). Or l'un des principaux objectifs est de réduire le nombre d'itérations de façon à réduire la durée de décodage. Dans le cas de l'hypothèse de Berrou la densité de probabilité de l'*information extrinsèque* sera donnée par :

$$P(L_e(d_k)|d_k = i) = e^{\frac{1}{\sigma_L^2}L_e(d_k)\times(2d_k-1)} \tag{4.9}$$

où

$$\sigma_L^2 = \frac{1}{N}\sum_{k=1}^{N}(|L_e(d_k)| - m_L)^2 \qquad \text{où} \quad m_L = \frac{1}{N}\sum_{k=1}^{N}|L_e(d_k)| \tag{4.10}$$

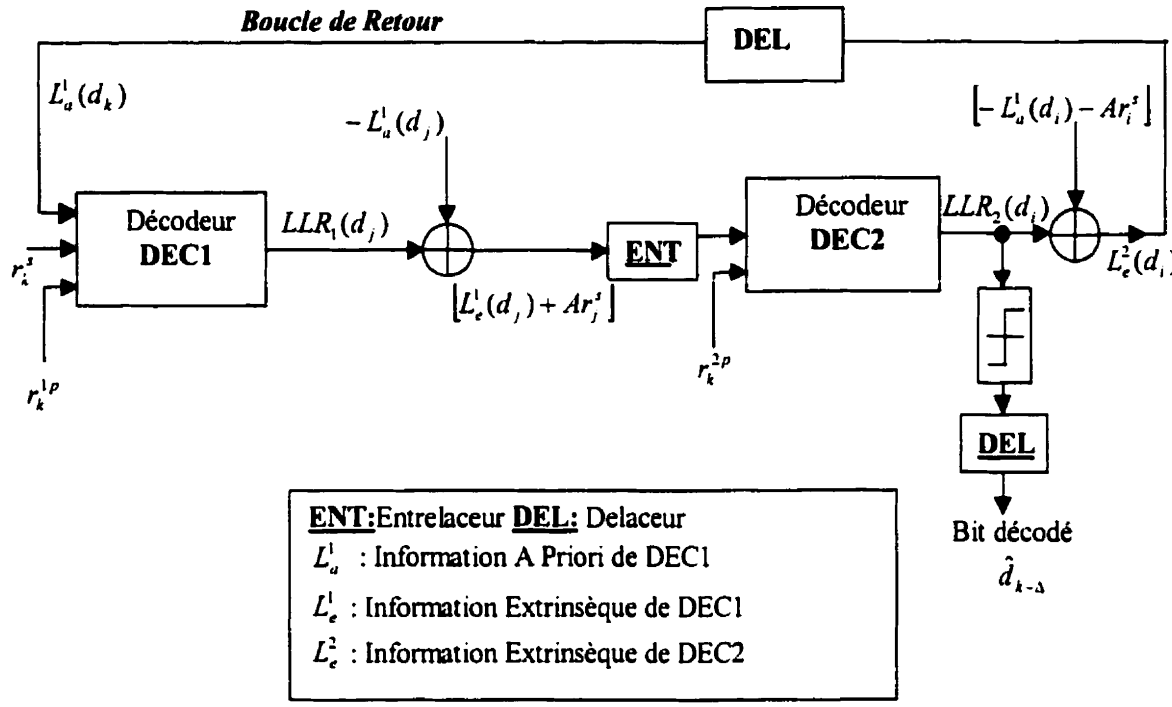

Figure 4.1: Schéma de principe d'un décodeur Turbo

Mais heureusement, cette hypothèse fut écartée plus tard par l'auteur lui même et donc l'*information extrinsèque* est considérée, comme Hagenauer l'a proposé au départ [Hag96], comme un rapport de vraisemblance.

4.4 Résultats

Nous allons décrire brièvement le comportement des Codes Turbo dans un canal AWGN. En guise de références, on pourra consulter le mémoire de Naoufel [Bou97]qui a commenté longuement les performances des Codes Turbo dans un canal AWGN. Ensuite, nous étudierons les performances d'erreurs des Codes Turbo dans un canal de Rayleigh, d'autant plus que les Codes Turbo sont en train de s'imposer dans le futur standard de la troisième génération des systèmes de communications sans fil et précisément pour les applications multimédia. C'est pourquoi il est intéressant d'étudier le comportement des Codes Turbo dans les canaux radiomobiles.

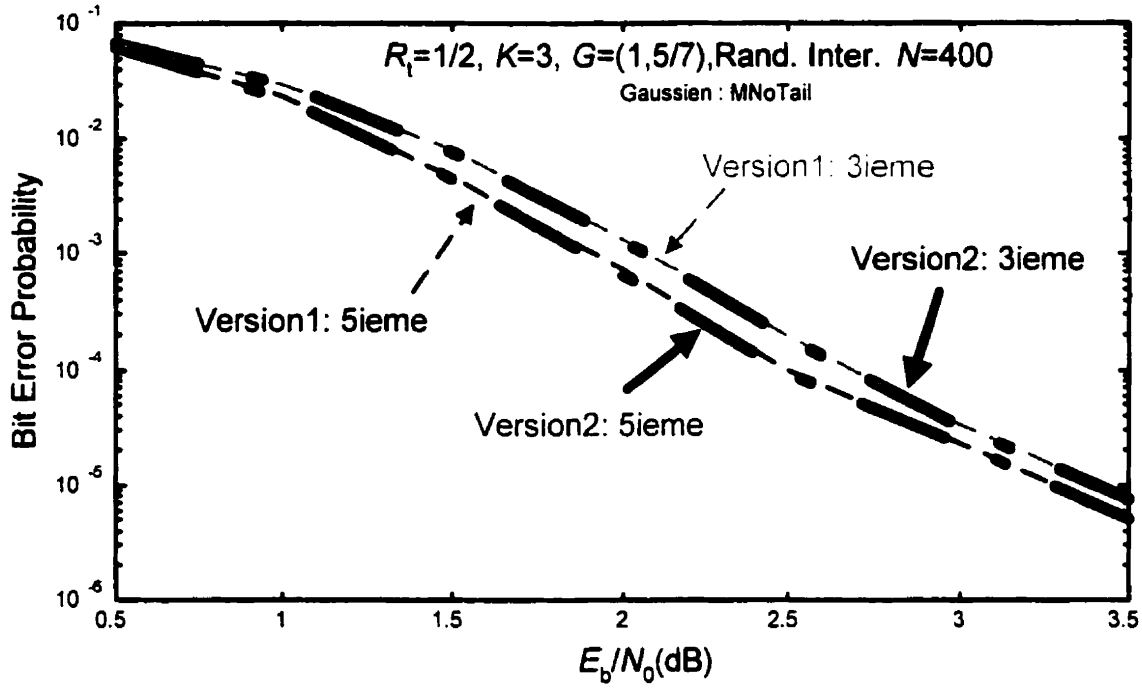

Figure 4.2: Comparaison de la BER de la version1 à la version2 dans un canal AWGN

Avant d'entamer l'analyse des performances, il est nécessaire de mentionner que, dans la majorité des cas, nous n'avons pas envoyé de queues pour terminer les treillis mais tout simplement, nous avons adopté l'alternative que nous avons dénotée **MNo-Tail.** Mentionnons que la notation **2Tails** dans les figures des performances d'erreur signifie tout simplement que deux queues ont été envoyées (voir le chapitre suivant). D'autre part, la *Version Log-MAP utilisée* de l'algorithme MAP a été utilisée. Néanmoins, les deux versions comparées aux chapitres précèdent fournissent exactement des performances d'erreurs identiques. En effet, la figure (4.2) montre bien que la *Version Log-MAP utilisée* (Version2) et la *première version Log-MAP (*Version1) performent identiquement. À la figure (4.2), nous avons seulement présenté les performances d'erreurs à la troisième (3ieme) et cinquième (5ieme) itération.

Dans l'ensemble des simulations, voici les divers intervalles de variations des paramètres utilisés :

$3 \leqslant K \leqslant 5$, $1/3 \leqslant R_t \leqslant 1/2$, $0 \leqslant E_b/N_0 \leqslant 5$ dB, $1 \leqslant$ (nombre d'itérations maximun)$\leqslant 5$.

4.4.1 Performance des Codes Turbo dans un canal AWGN

4.4.1.1 Turbo Code de taux $R_t = 1/3$ et $K = 3$

Nous présentons la probabilité d'erreur par bit appelée "Bit Error Probability" (BER) en fonction du rapport E_b/N_0. Des résultats des simulations, nous pouvons déduire les comportements suivants :

- Pour des faibles rapports signal sur bruit E_b/N_0 , nous constatons une amélioration de la BER lorsque la taille du bloc croît (donc la taille de l'entrelaceur car nous avons pris la taille de l'entrelaceur identique à celle de la longueur du bloc d'information). Cette amélioration est de l'ordre de $1/N$. Par exemple à $E_b/N_0 = 1.5$ dB, en comparant $N = 400$ (figure (4.4)) et $N = 3600$ (figure (4.6)) la BER s'améliore d'un facteur de 9, ce qui n'est que le rapport entre 3600 et 400 (3600/400=9). Notons que ceci a été démontré théoriquement par *Benedetto* [BMo96] dans le cas de l'entrelacement bloc classique)
- Nous avons adopté l'entrelacement bloc de type pseudo-aléatoire dans presque la totalité de nos simulations et ceci pour une raison simple : ce type d'entrelacement donne les meilleurs BER. Dans la figure (4.3) nous avons utilisé un entrelaceur bloc classique qui fournit des performances d'erreurs équivalentes à celui de l'entrelaceur aléatoire dans le cas où la taille du bloc est petite (inférieur à 400).
- Le nombre d'itération requis décroît lorsque E_b/N_0 croît. Par exemple, à $E_b/N_0 =$ 1 dB et pour $N = 1024$ (figure (4.5))ou $N = 3600$ (4.6), nous avons besoin d'au moins cinq itérations afin de se rapprocher de la limite de la BER; par contre à $E_b/N_0 = 3$ dB , (figure (4.6)) trois itérations sont largement suffisantes.

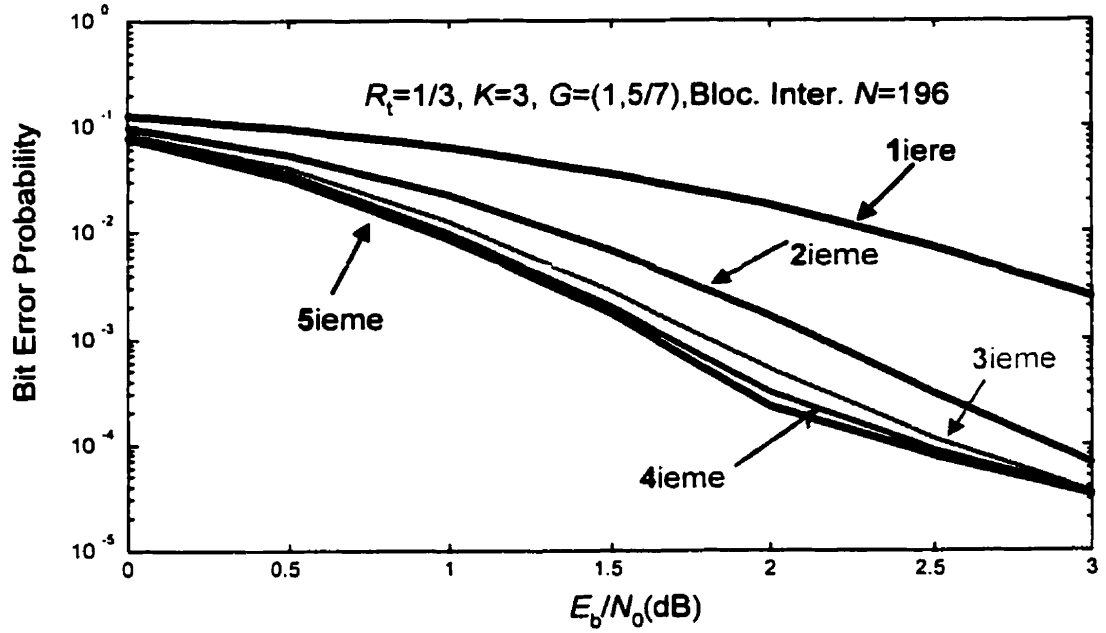

Figure 4.3: BER, $K = 3$, $R_t = 1/3$, $N = 196$, Entrel. Bloc, Canal AWGN, Terminaison : 2Tails.

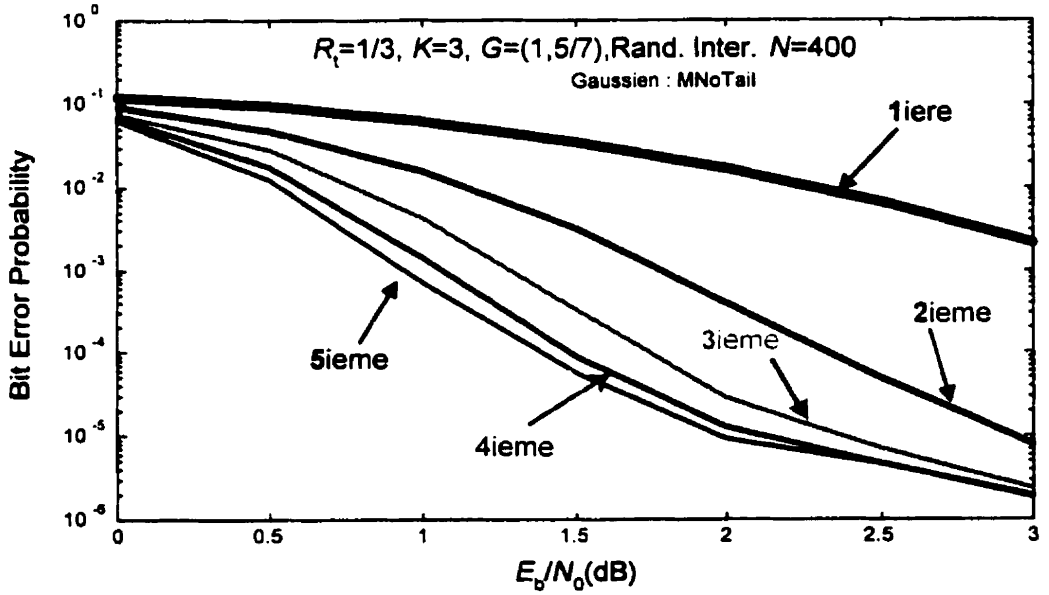

Figure 4.4: BER, $K = 3$, $R_t = 1/3$, $N = 400$, Entrel. Aléatoire, Canal AWGN, Terminaison : MNoTail.

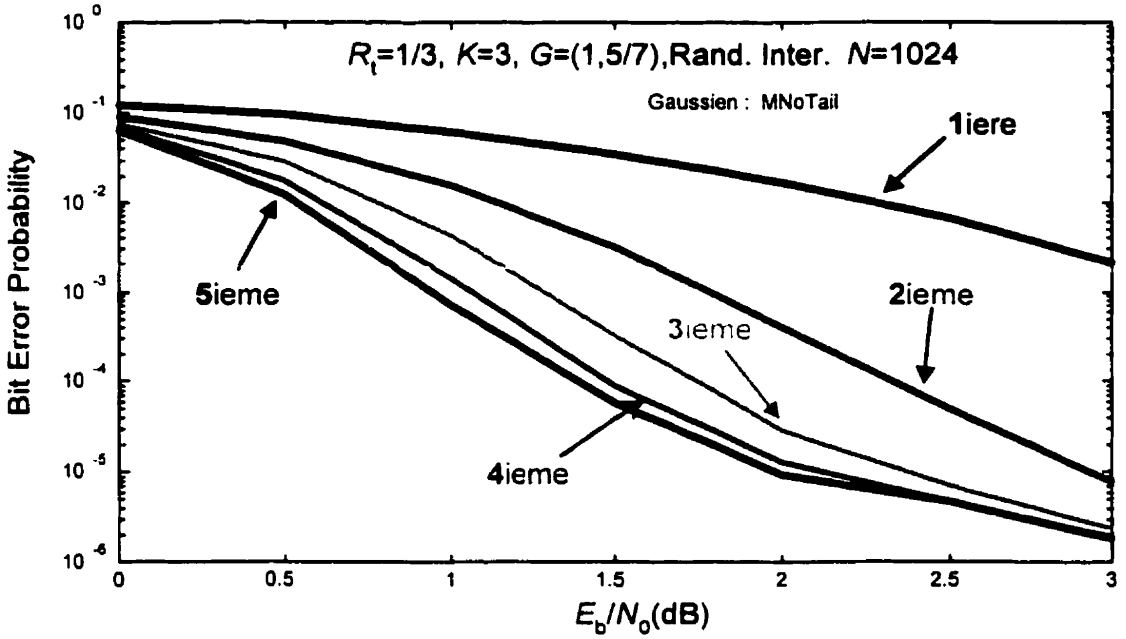

Figure 4.5: BER, $K = 3$, $R_t = 1/3$, $N = 1024$, Entrel. Aléatoire, Canal AWGN, Terminaison : MNoTail.

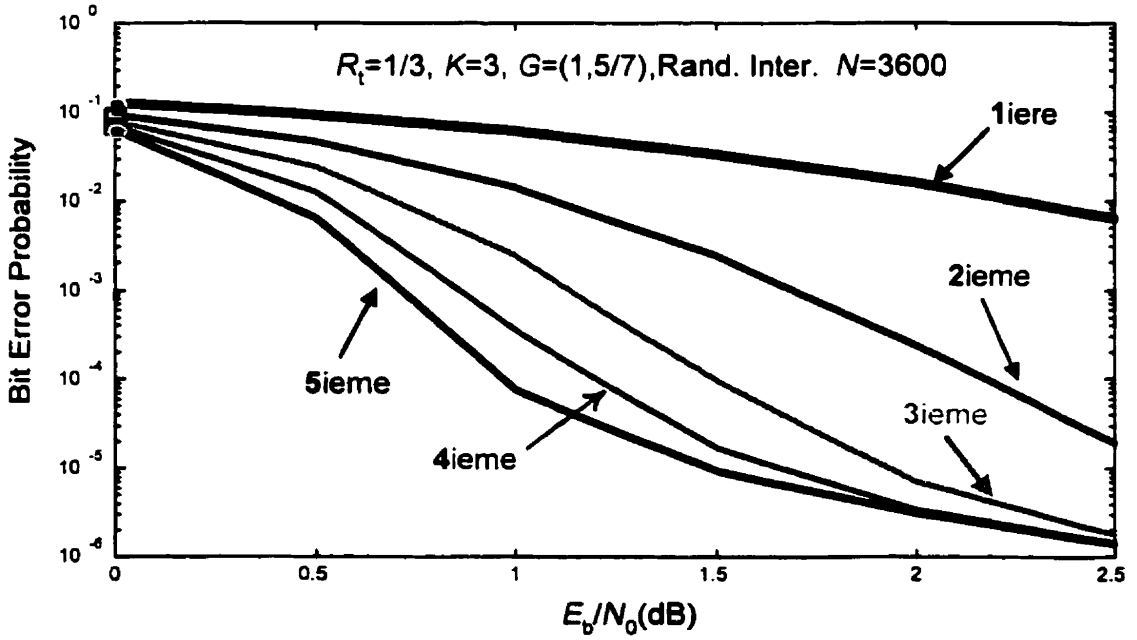

Figure 4.6: BER, $K = 3$, $R_t = 1/3$, $N = 3600$, Entrel. Aléatoire, Canal AWGN, Terminaison : 2Tails.

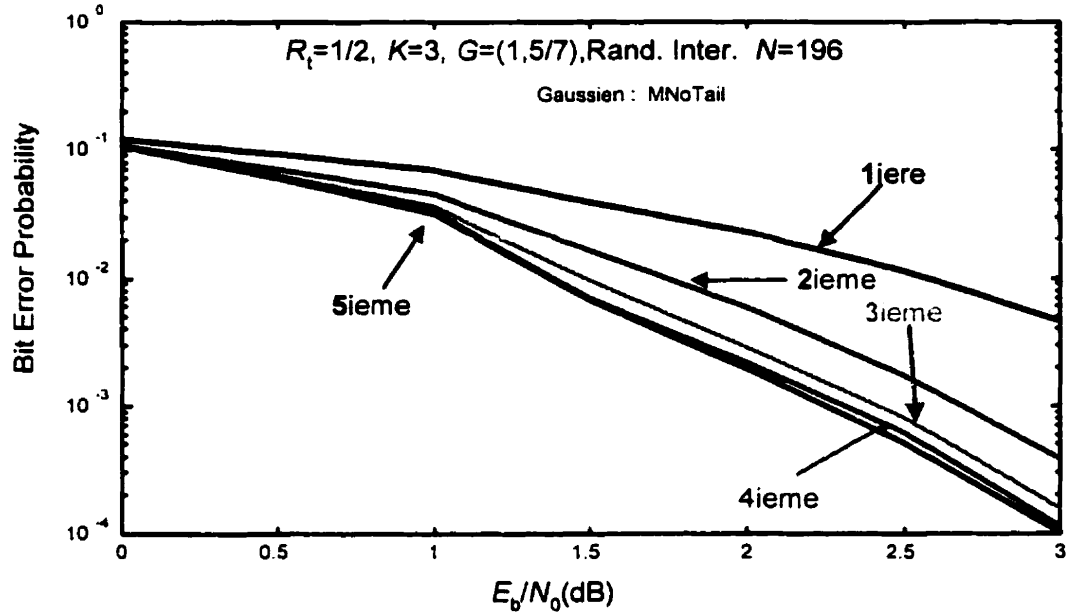

Figure 4.7: BER, $K = 3$, $R_t = 1/2$, $N = 196$, Entrel. Aléatoire, Canal AWGN, Terminaison : MNoTail.

4.4.1.2 Turbo Code de taux $R_t = 1/2$ et $K = 3$

- La première remarque concernant l'influence de la taille d'entrelaceur à $R_t = 1/2$ est identique à celle observée pour un taux de codage $R_t = 1/3$. Il est évident en observant les figures (4.7), (4.8) et (4.9) que la BER décroît avec l'accroissement de la taille du bloc N.
- L'inefficacité de l'entrelaceur bloc classique par rapport à celui qui est pseudo aléatoire est flagrante pour des blocs de moyenne ou grande taille. Il suffit de comparer la figure (4.10) à la figure (4.11) pour constater qu'un gain de 1.2 dB pour $N = 3600$ à $E_b/N_0 = 1.5$ dB ou 3 dB.
- La perte en dB par rapport au taux 1/3 décroît avec l'accroissement E_b/N_0.

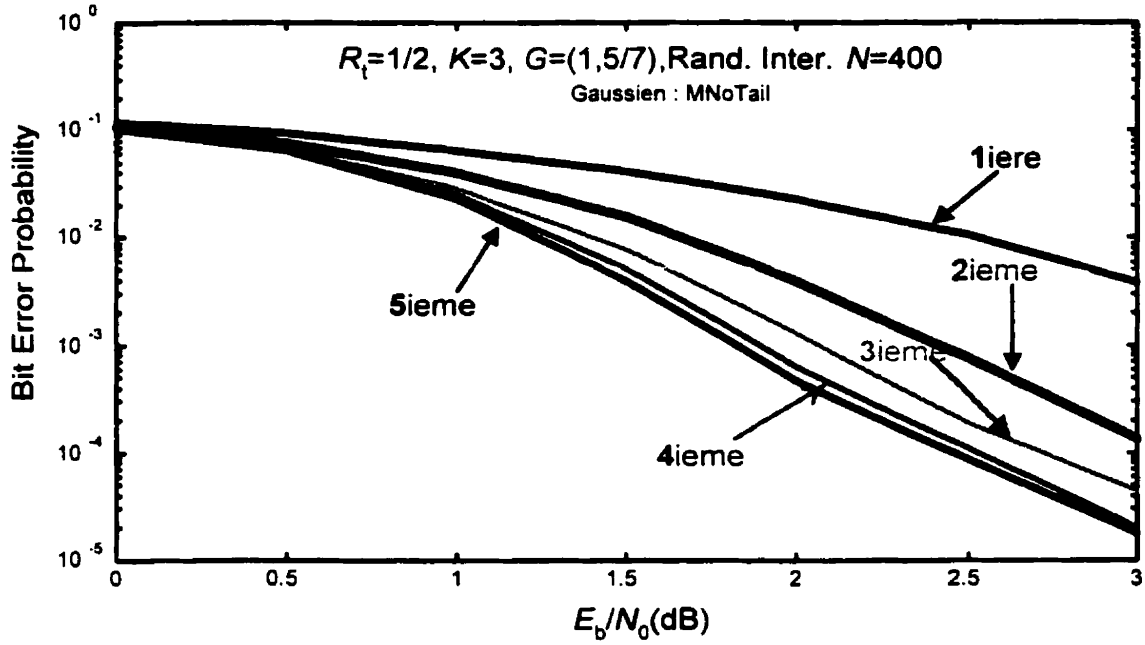

Figure 4.8: BER, $K = 3$, $R_t = 1/2$, $N = 400$, Entrel. Aléatoire, Canal AWGN, Terminaison : MNoTail.

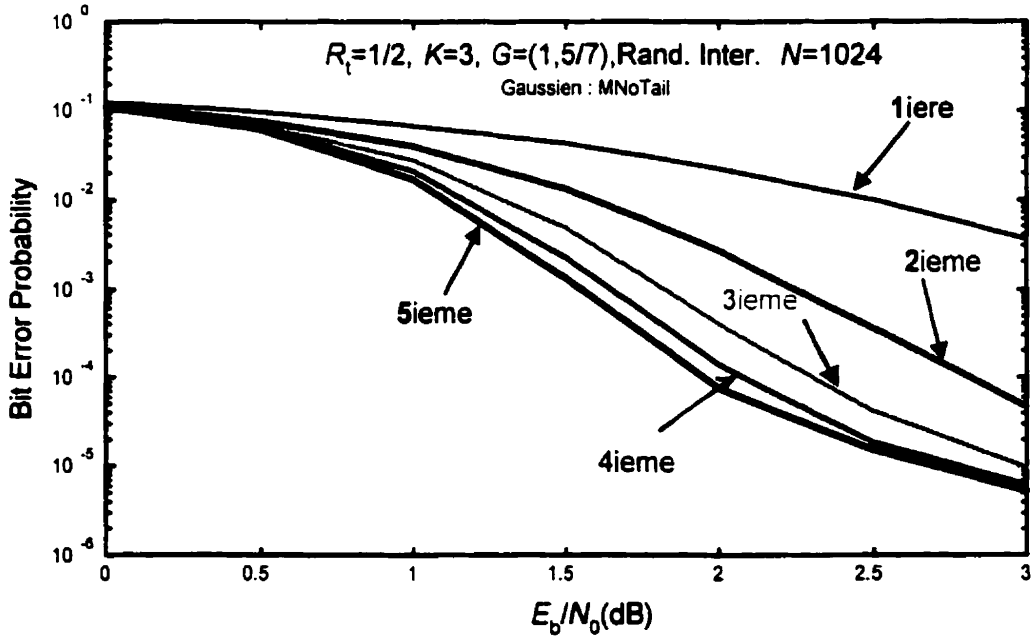

Figure 4.9: BER, $K = 3$, $R_t = 1/2$, $N = 1024$, Entrel. Aléatoire, Canal AWGN, Terminaison : MNoTail.

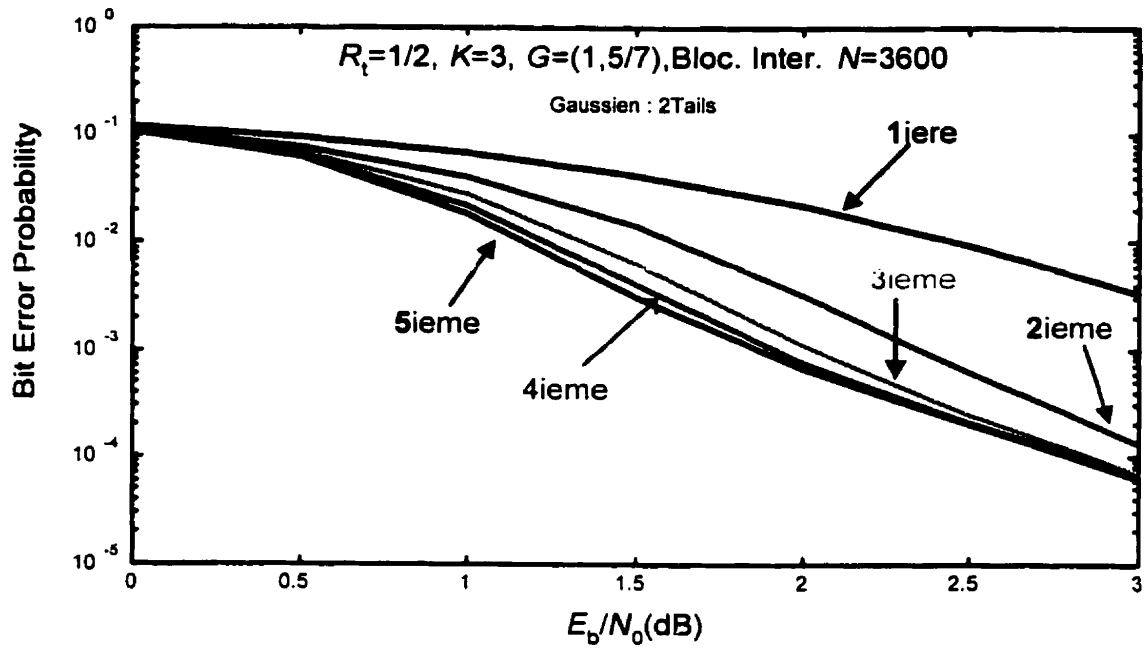

Figure 4.10: BER, $K = 3$, $R_t = 1/2$, $N = 3600$, Entrel. Bloc, Canal AWGN, Terminaison : 2Tails.

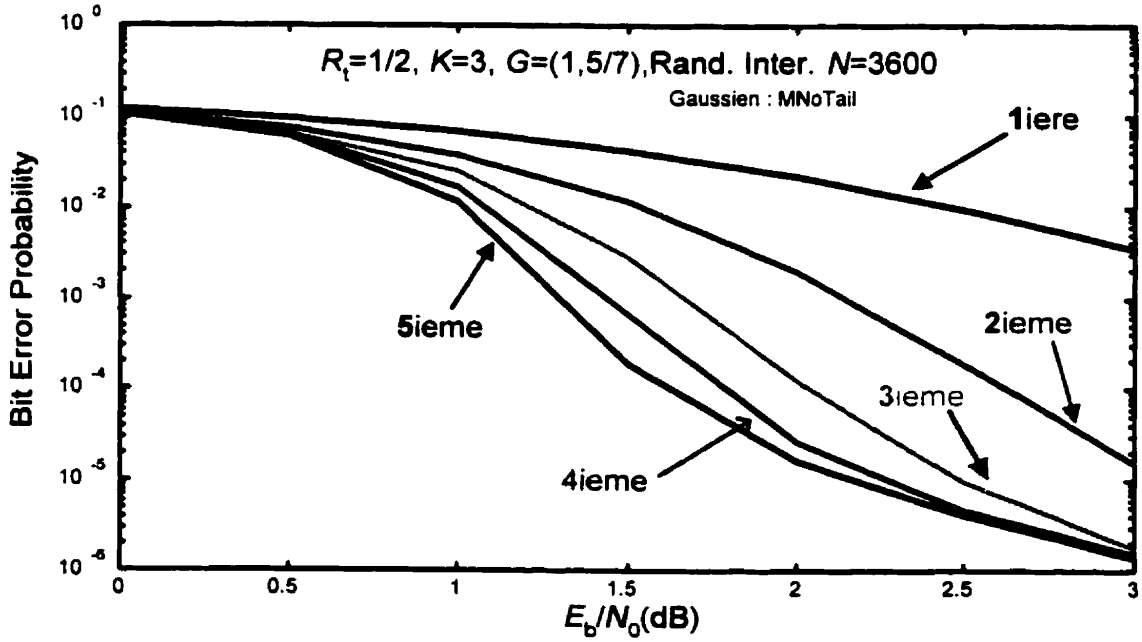

Figure 4.11: BER, $K = 3$, $R_t = 1/2$, $N = 3600$, Entrel. Aléatoire, Canal AWGN, Terminaison : MNoTail.

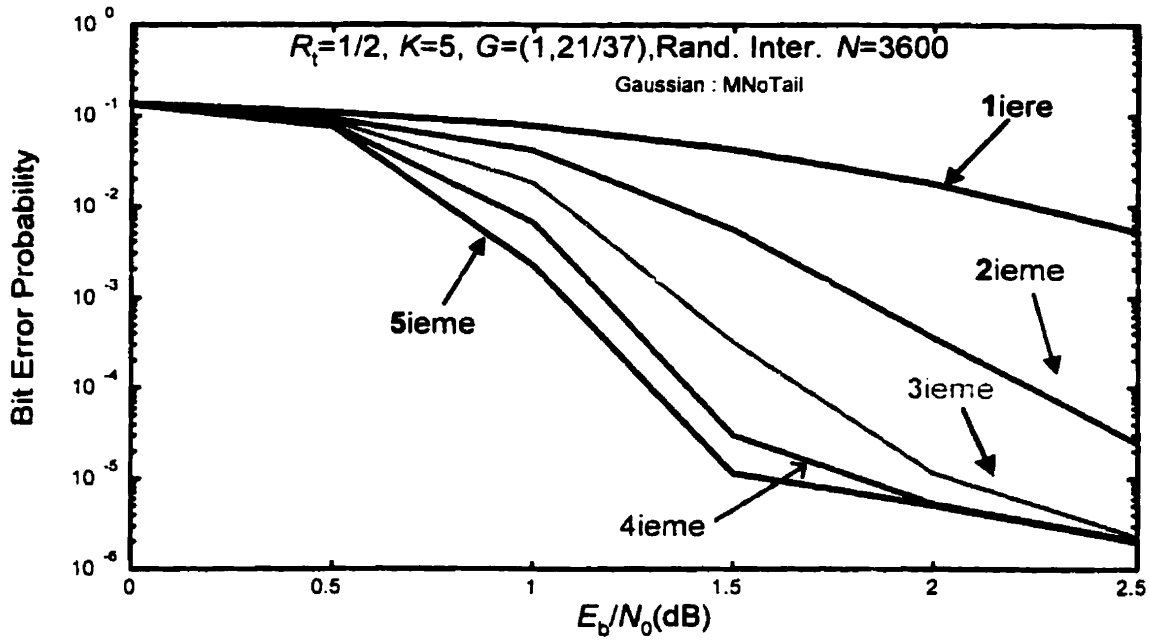

Figure 4.12: BER, $K = 5$, $R_t = 1/2$, $N = 3600$, Entrel. Aléatoire, Canal AWGN, Terminaison : MNoTail.

4.4.1.3 Turbo Code de taux $R_t = 1/2$ et $K = 5$

En comparant, les résultats de simulations de $K = 3$ et $K = 5$ (pour un taux de codage de 1/2, nous pouvons affirmer que les performances des deux cas sont identiques pour E_b/N_0 inférieur à 1 dB surtout dans le cas des blocs de petits tailles. Par contre pour $N = 3600$, le gain devient important pour des E_b/N_0 supérieurs à 1 dB. La figure (4.12) montre qu'un gain de 0.5 dB ou plus est atteint par rapport à $K = 3$ (figure(4.11)).

Notons que le nombre d'itérations nécessaires afin d'atteindre l'effet de seuil est supérieur par rapport à $K = 3$ [Bar96]. Si la BER désirée est faible, il n'y a aucun avantage à augmenter la longueur de contrainte, surtout pour des blocs de petite taille (inférieurs à 400).

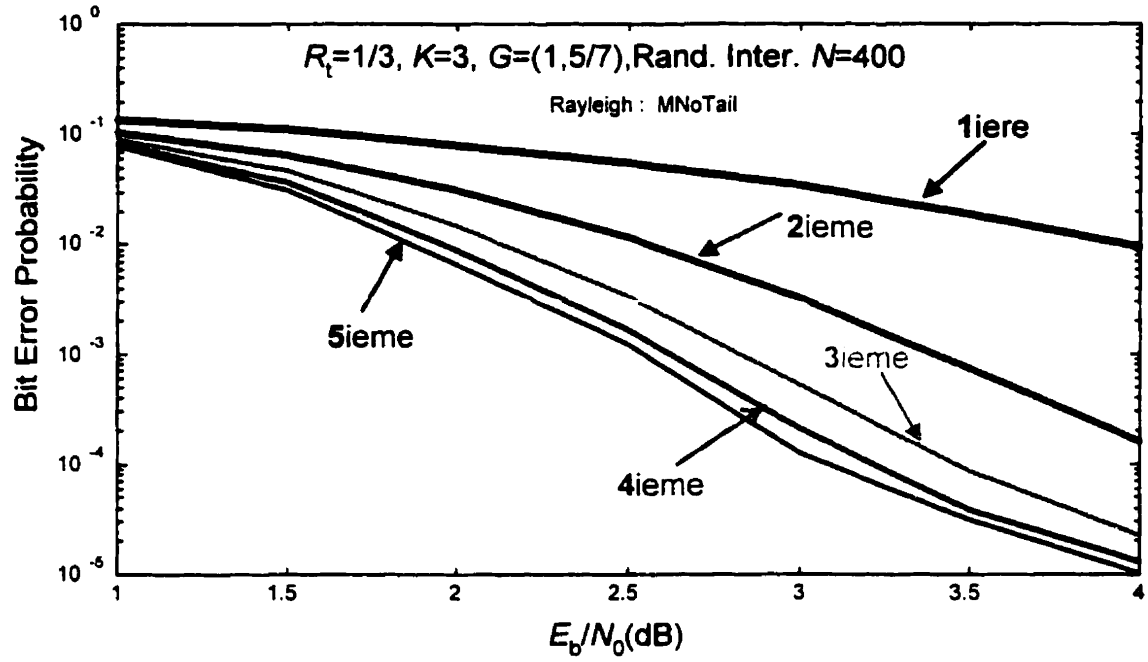

Figure 4.13: BER, $K = 3$, $R_t = 1/3$, $N = 400$, Entrel. Aléatoire, Canal Rayleigh, Terminaison : MNoTail.

4.4.2 Performance des Codes Turbo dans un canal de Rayleigh

Le modèle du canal de Rayleigh utilisé dans les simulations que nous allons analyser a été décrit au paragraphe (2.7).

4.4.2.1 Turbo Code de taux $R_t = 1/3$ et $K = 3$

Évidemment, nous devons nous attendre à une chute de performance par rapport à un canal AWGN, cette chute est au moins de 1.5 dB. La figure (4.13) indique une perte de 2 dB pour une BER=10^{-5} par rapport à un canal AWGN (voir figure (4.4)).

Les remarques citées dans le cas du canal AWGN pour un taux $R_t = 1/3$ et $K = 3$ s'appliquent aussi dans la cas de Rayleigh. Par exemple, la performance d'erreur croît avec la taille du bloc ou de l'entrelaceur, pour $N = 1024$ (figure (4.14), la BER est inférieure à celle de $N = 400$ (figure (4.4)).

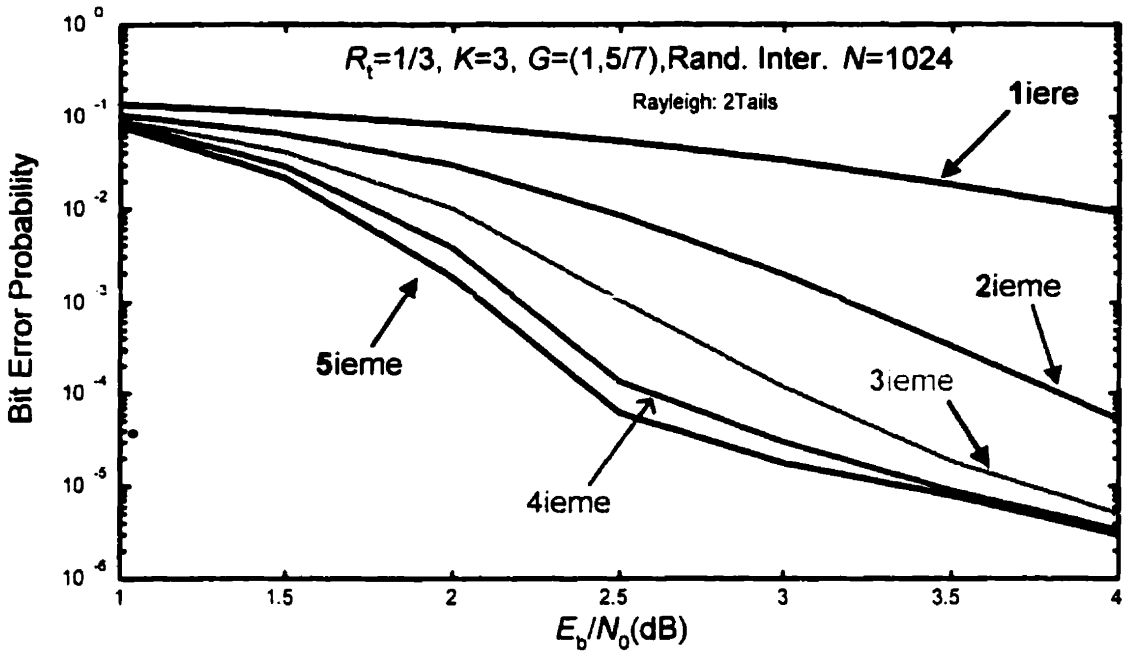

Figure 4.14: BER, $K = 3$, $R_t = 1/3$, $N = 1024$, Entrel. Aléatoire, Canal Rayleigh, Terminaison : MNoTail.

4.4.2.2 Turbo Code de taux $R_t = 1/2$ et $K = 3$

Tel que prévisible les performances chutent lorsque le taux de codage passe de $R_t = 1/3$ à $R_t = 1/2$. Par contre, il faut noter que cette perte est plus marquée dans le cas d'un canal de Rayleigh que dans celui d'un canal AWGN. Le gain obtenu dû au passage de $R_t = 1/2$ à $R_t = 1/3$ dans le cas des canaux à évanouissements est plus grand que celui dans les canaux AWGN. Prenons l'exemple de $N = 1024$; dans le cas d'un canal de Rayleigh le gain est de 1.7 dB pour $N = 1024$, lors du passage à BER=10^{-4}, de $R_t = 1/2$ (figure (4.16)) à $R_t = 1/3$ (figure (4.14)). De même pour $N = 400$ le gain est de 1.6 dB lors du passage de $R_t = 1/2$ pour un BER=10^{-4} (figure (4.15)) à $R_t = 1/3$ (figure (4.4)). Par contre dans le cas de AWGN le gain dû au passage de $R_t = 1/2$ à $R_t = 1/3$ pour $N = 400$ et $N = 1024$ est respectivement de 0.7 dB et 0.5 dB.

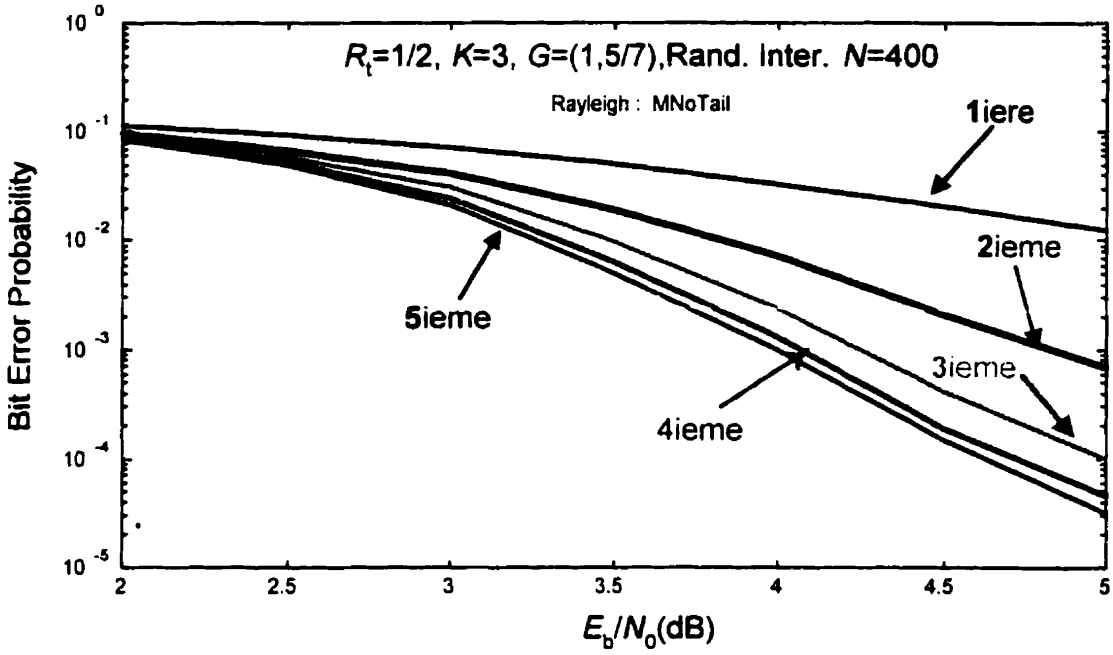

Figure 4.15: BER, $K = 3$, $R_t = 1/2$, $N = 400$, Entrel. Aléatoire, Canal Rayleigh, Terminaison : MNoTail.

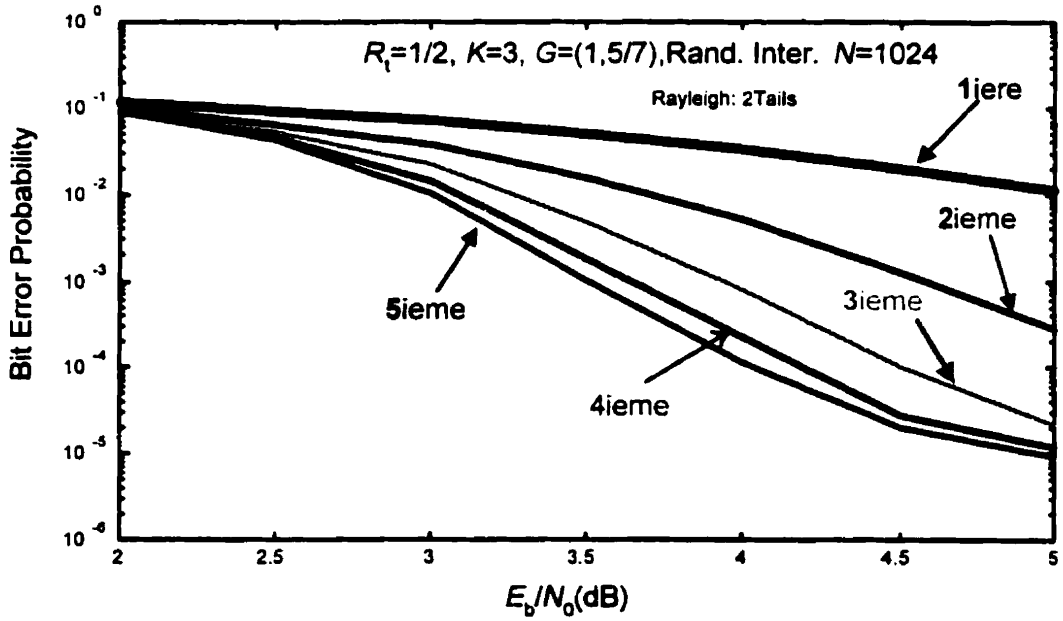

Figure 4.16: BER, $K = 3$, $R_t = 1/2$, $N = 1024$, Entrel. Aléatoire, Canal Rayleigh, Terminaison : MNoTail.

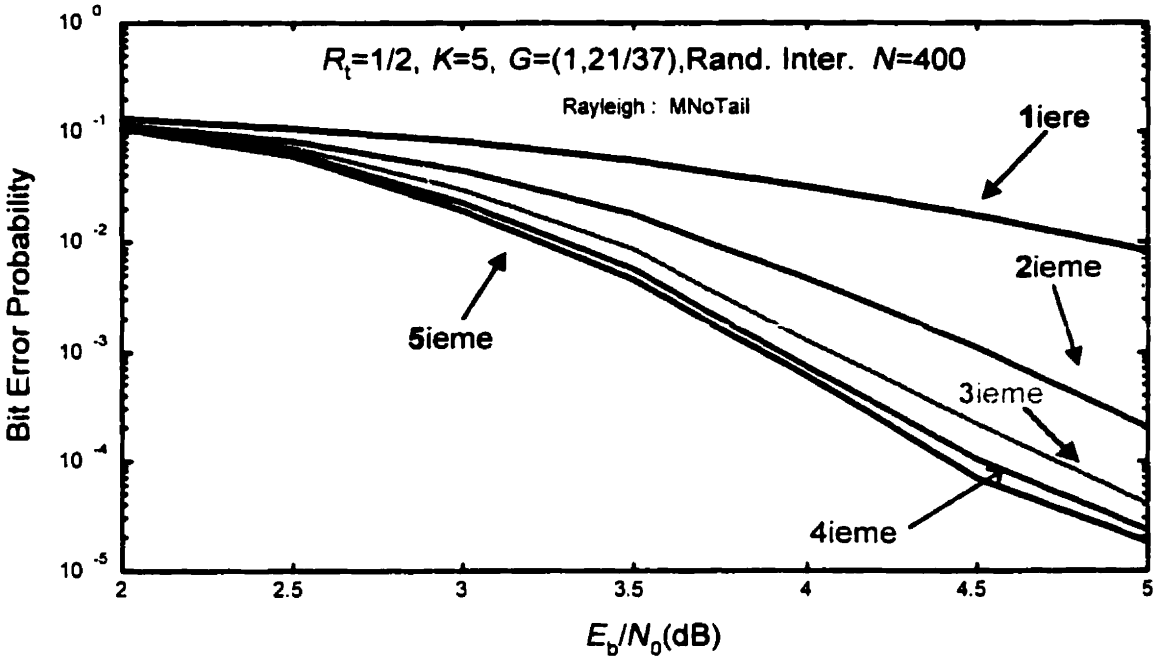

Figure 4.17: BER, $K = 5$, $R_t = 1/2$, $N = 400$, Entrel. Aléatoire, Canal Rayleigh, Terminaison : MNoTail.

4.4.2.3 Turbo Code de taux $R_t = 1/2$ et $K = 5$

La première remarque qui ressort en comparant la figure (4.17) où $K = 5$ à la figure (4.15) où $K = 3$ est que les performances sont quasi identiques et que pour E_b/N_0 inférieur à 3.5 dB les performances sont meilleures dans le cas de $K = 3$. Ce résultat coïncide avec celui de Hagenauer et confirme celui de Berrou [BGl96]. Donc, l'augmentation de la longueur de contrainte ne garantit pas un gain au niveau de la performance mais par contre elle garantie une augmentation de la complexité et de la latence du codeur et du décodeur Turbo.

4.5 Estimation du Canal

Dans un décodeur Turbo, si l'on suppose que l'algorithme MAP (Maximum A Posteriori) est utilisé comme algorithme de décodage, la variance du bruit doit être

connue car elle fait partie des données nécessaires pour l'opération de décodage (voir équation (4.8). En effet, la dérivation exacte des *probabilités a posteriori* (APP) exige une connaissance détaillée du canal. Les vrais problèmes apparaissent lorsque le canal varie avec le temps ce qui nécessite une remise à jour continuelle de la variance. Dans certaines applications radiomobiles très bruitées et où le recepteur se déplaçe rapidement, la connaissance de la variance devient un problème difficile. En pratique, le but est d'abord de repérer l'intervalle des variations dans le canal [Hee99]. Le décodeur résultant est appelée *mismatched decoder* (mauvais appariement) par rapport au canal bien sûr. Donc, la question est de savoir quel est l'impact du *mismatch* sur les performances du décodeur? Avant d'introduire les diverses méthodes qui permettent d'estimer la variance du bruit dans le canal, nous allons discuter brièvement de la robustesse du décodage turbo face au *mismatch* dans un canal Gaussien.

La première étude de ce problème a été présentée par Summer et Wilson [SWi98]. Ces derniers ont supposé que le canal est invariant dans le temps pendant la transmission d'une trame d'information [SWi98]. Ils ont montré l'effet quantitatif du *mismatch* : une sous-estimation de 1 dB ou peut être 2 dB dans le rapport E_b/N_0 est tolérée mais pour des valeurs supérieures, la dégradation dans les performances devient importante. Il faut noter qu'une surestimation du rapport E_b/N_0 est mieux tolérée qu'une sous-estimation. La région ou l'intervalle du *mismatch* E_b/N_0 dans lequel le décodage itératif fournit des performances impressionnantes s'étale de -3 dB à 6 dB. Ces résultats ont été généralisés par Kim et Wicker [Hee99] qui ont montré que le décodage Turbo est aussi robuste dans le cas des canaux qui varient d'un bit transmis à un autre bit, que dans le cas des trames (d'une trame à une autre). Kim et Wicker ont montré qu'une estimation exacte du canal peut se déduire en se servant des paramètres du décodage Turbo. Ils ont aussi observé que la surestimation de la variance pose plus de problèmes que la sous-estimation. En effet, le décodeur Turbo est très robuste dans un large intervalle de valeurs erronées (mismatch). Mentionnons, qu'une sous-estimation de la puissance du bruit (variance) est équivalente à

une surestimation du rapport E_b/N_0.

Nous allons présenter les trois méthodes choisies qui permettent d'estimer la variance du bruit. Mais, avant de traiter les méthodes, nous allons rappeler les conditions dans lesquelles nous avons simulé diverses méthodes.

4.5.1 Modèle mathématique et définitions :

Soit d_k le bit à la sortie du codeur, d_k est modulé suivant la modulation BPSK donc à la sortie du modulateur nous avons :

$$x_k = 2d_k - 1 \tag{4.11}$$

Ensuite le signal est transmis dans un canal AWGN, donc en supposant une démodulation cohérente avec une parfaite synchronisation au récepteur, le signal reçu est de la forme :

$$r_k = x_k + p_k = \pm\sqrt{E_s} + p_k \tag{4.12}$$

où p_k est une variable aléatoire Gaussienne de moyenne zéro et de variance $\sigma^2 = N_0/2$, N_0 étant la densité spectrale du bruit. $E_s = (\log_2 M)\left(\frac{k}{n}\right) E_b$ est l'énergie du signal x_k, où $\frac{k}{n}$ est le taux de codage, M est la taille de l'alphabet c.à.d le nombre total de symboles ($M = 2$ dans le cas de la modulation BPSK) et E_b est l'énergie par bit.

4.5.2 Les méthodes d'estimation

Dans nos simulations des performances d'erreurs, nous avons supposé une estimation parfaite de la variance du bruit dans le canal. Nous avons déduit la valeur de la variance à partir du rapport E_b/N_0 et de R_t. Donc, en connaissant E_b/N_0 et le taux de codage R_t, nous pouvons déduire la variance du bruit. Tout d'abord, nous allons exprimer la variance du bruit en fonction de E_b/N_0. Pour un rapport E_b/N_0 donné,

la variance σ^2 se met sous la forme :

$$\sigma^2 = (2R_t\frac{E_b}{N_0})^{-1} \tag{4.13}$$

dans ce cas, E_s l'energie d'un symbole codé est supposé égal à 1. En effet

$$\begin{aligned} \sigma^2 &= N_0/2 \Longrightarrow \frac{\sigma^2}{E_s} = \frac{N_0}{2E_s} \\ \text{or } E_s &= R_t E_b \\ &\Rightarrow \frac{\sigma^2}{E_s} = \frac{N_0}{2RE_b} = (2R_t\frac{E_b}{N_0})^{-1} \\ \text{or } E_s &= 1 \\ &\Longrightarrow \sigma^2 = (2R_t\frac{E_b}{N_0})^{-1} \end{aligned}$$

Reed [RA097] a démontré qu'une mauvaise estimation (surtout une sous-estimation) de la variance détériore sensiblement la performance du décodeur. D'où la nécessité de trouver une méthode efficace et précise. Dans ce but, nous allons d'abord présenter la méthode classique qui permet d'estimer la variance, ensuite nous allons présenter et valider par simulation la méthode proposée par [RA097]. Enfin une troisième méthode numérique proposée par [RA097] et [Pie97] sera expliquée et simulée par ordinateur.

4.5.2.1 Méthode classique

Pour une modulation BPSK et QPSK, la variance est donnée par [RA097] :

$$\sigma^2 = E[r_k^2] - (E[|r_k|])^2 \tag{4.14}$$

où $E[]$ étant l'espérance mathématique. Si la j$^{\text{ième}}$ séquence reçue est de longueur N alors la variance estimée est donnée par :

$$\sigma_j^2 = \frac{\sum_{k=1}^{N} r_{k,j}^2}{N} - \frac{\sum_{k=1}^{N} |r_{k,j}|^2}{N} \tag{4.15}$$

l'indice j indique que c'est le $j^{ième}$ bloc ou séquence qui a été reçu.

Nous avons testé cette méthode pour des blocs de taille 400 bits ($R_t = 1/3$ donc 1200 symboles codés). Nous nous sommes limités à l'intervalle où $E_b/N_0 \in [0 - 3\text{dB}]$. Afin d'obtenir une estimation fiable, nous avons fait la simulation pour 10000 blocs. La figure (4.18) montre la différence entre la valeur de la moyenne de la variance estimée (sur les dix milles blocs) et la vraie valeur. Nous pouvons déduire de cette figure que l'estimation est assez fiable lorsque le rapport est élevé (variance faible : inférieur à 0.6, ceci n'apparaît pas sur la figure mais nous l'avons vérifié par simulation), or l'intérêt du codage Turbo réside dans sa performance lorsque le rapport E_b/N_0 est faible donc cette méthode représente un certain désavantage. La figure (4.19) montre clairement lorsque la variance estimée s'éloigne de la vraie valeur pour des faibles rapports de E_b/N_0 (correspondant à des variances supérieures à 1). Cependant, la valeur moyenne n'est pas significative si nous ne connaissons pas les écarts entre les divers blocs. En conséquence, le calcul de la fonction de densité de probabilité (PDF) de l'écart (appelé "E_b/N_0 offset")entre la vraie valeur E_b/N_0 et la valeur estimée est essentielle. À la figure (4.19) la PDF est tracée pour plusieurs valeurs de E_b/N_0. Nous pouvons remarquer que la méthode classique produit une surestimation de quelques dB de E_b/N_0 pour la grande majorité des blocs simulés.

Notons que les figures (4.18) et (4.19) sont les résultats d'une simulation en utilisant *Matlab*.

4.5.2.2 Nouvelle estimation

Cette méthode a été proposée par [RA097]. L'astuce de cette méthode consiste à utiliser l'estimation des données à la sortie du décodeur turbo afin d'estimer la variance du bruit. Dans le cas d'un canal AWGN, cette méthode donne des meilleurs performances pour des faibles E_b/N_0 . Comme nous l'avons indiqué précédemment, une mauvaise estimation de la variance détériore les performances d'erreur. Mais cette détérioration est négligeable dans le cas où l'estimation serait légèrement incorrecte.

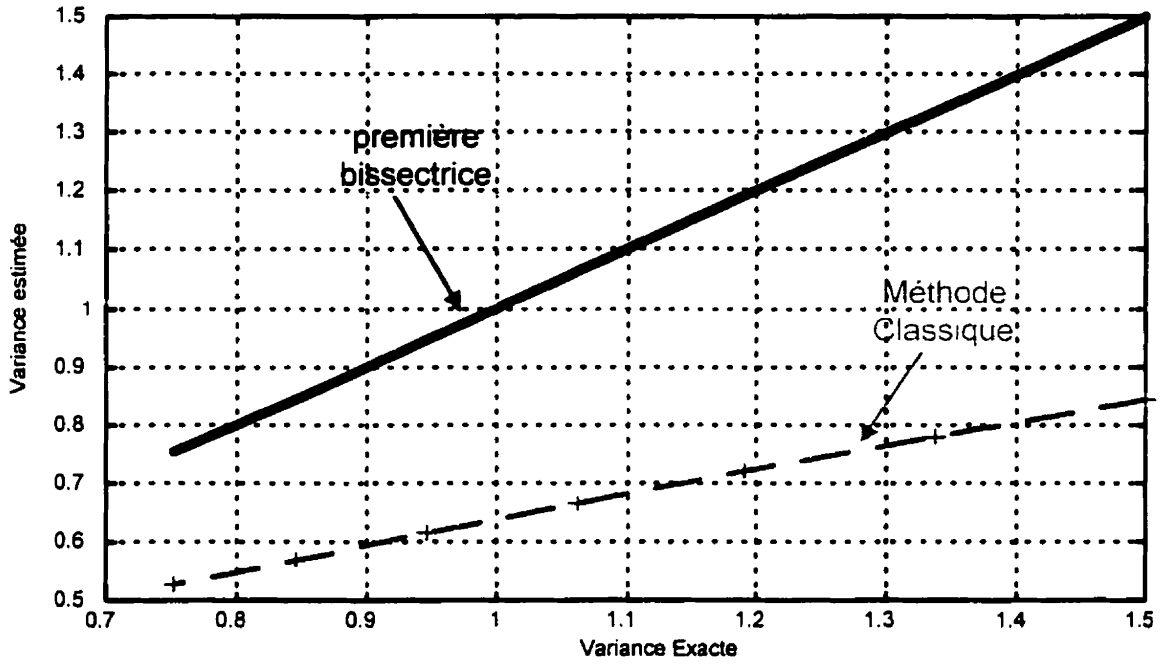

Figure 4.18: Comparaison entre la variance exacte et la variance estimée par la méthode classique.

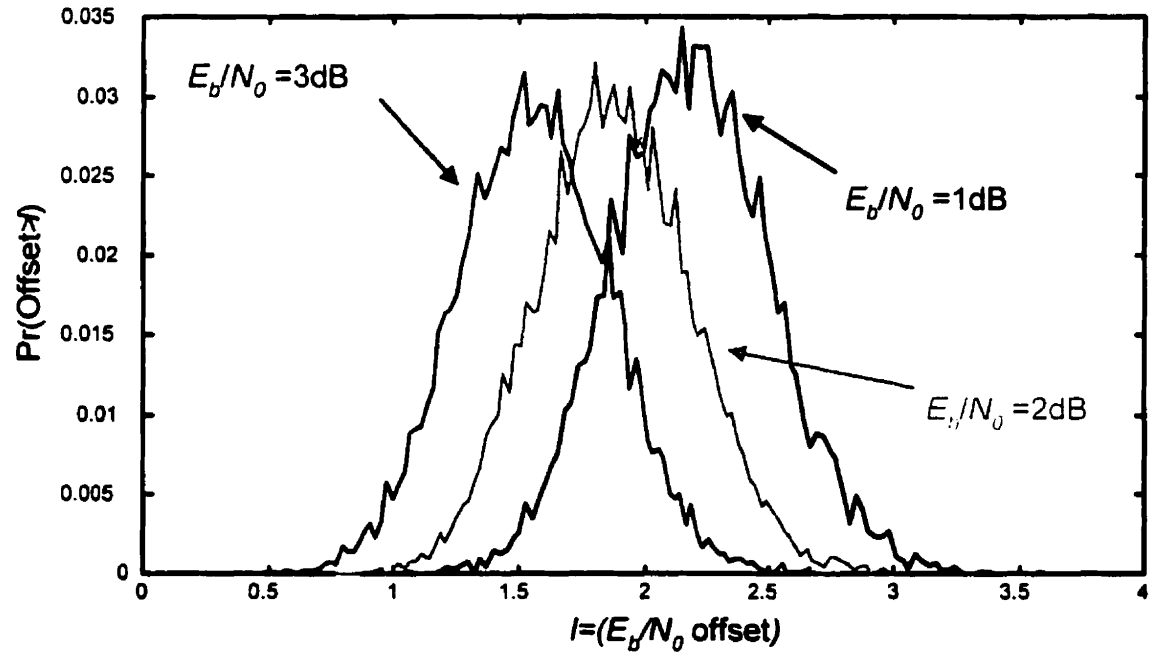

Figure 4.19: PDF de l'écart l pour la méthode classique pour plusieurs E_b/N_0.

La variance est exprimée sous la forme :

$$\sigma_{j+1}^2 = \frac{\sum_{k=1}^{N}(r_{k,j} - [2\hat{d}_{k,j} - 1])^2}{N} \tag{4.16}$$

$\hat{d}_{k,j} \in \{-1, +1\}$ est la sortie dure du décodeur Turbo à l'instant k pour le $j$ième bloc ou séquence. $r_{k,j}$ est le $k$ième bit du $j$ième bloc. La figure (4.20) montre l'amélioration de cette méthode par rapport à la précédente (classique). En effet, la nouvelle valeur estimée est très proche de la valeur exacte lorsque le rapport E_b/N_0 est assez faible. Nous pouvons déduire que cette méthode est plus précise pour un intervalle de E_b/N_0 plus large. Ajoutant que la sous-estimation ou la surestimation de E_b/N_0 de très grande majorité des blocs ne dépasse pas le 1 dB et en plus les PDF sont centrées autour de 0 dB.

L'inconvénient de cette méthode est qu'il faut attendre la décision du décodeur (décision dure c.à.d choisir entre 0 ou 1) avant de pouvoir estimer la variance. La variance du $j$ième peut être ajustée en prenant en compte la variance du bloc précédent suivant la relation :

$$\widehat{\sigma}_j^2 = \sigma_j^2\theta + (1 - \theta)\widehat{\sigma}_{j-1}^2 \tag{4.17}$$

où θ est un nombre réel positif largement plus petit que 1 (nous avons pris $\theta = 0.01$).

Cette méthode est meilleure que la méthode classique pour des faibles E_b/N_0 dans un canal AWGN. Mentionnons que la simulation de la figure (4.20) provient du simulateur (Turbo Codes) programmé en C.

4.5.2.3 Troisième méthode

Cette méthode est actuellement utilisée dans les premières puces d'un décodeur turbo [Pie97]. Elle a l'avantage d'être plus simple à réaliser que la méthode précédente et plus astucieuse que la méthode classique. Cette méthode consiste à calculer le

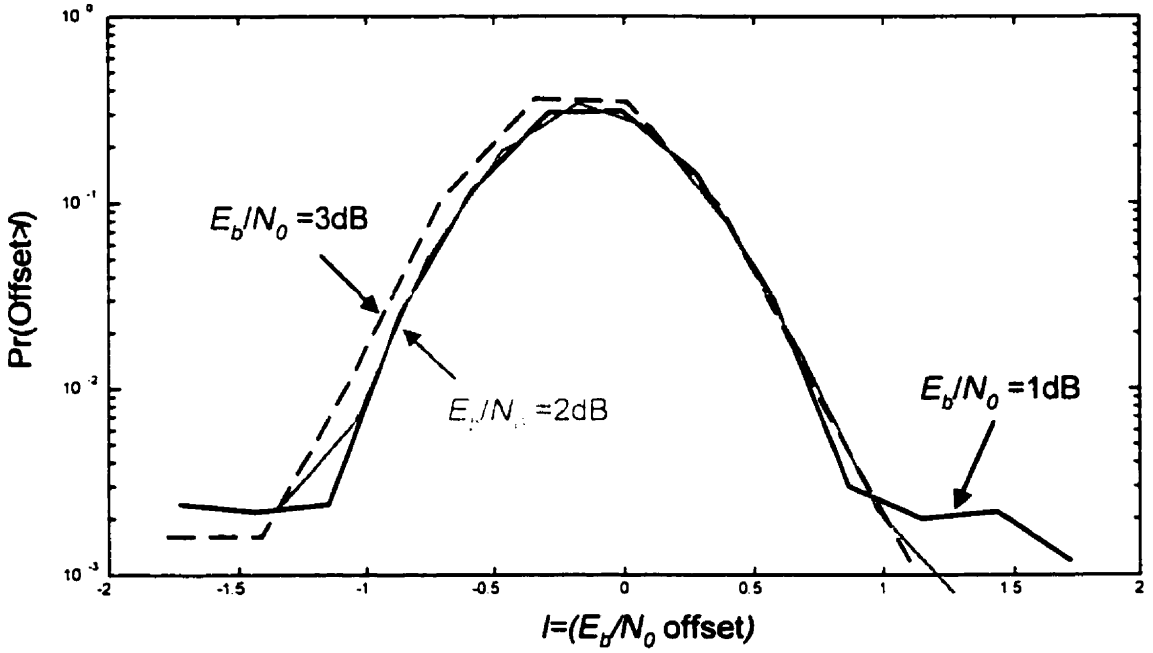

Figure 4.20: PDF de l'écart l pour la deuxième méthode pour plusieurs E_b/N_0.

rapport $z = E[r_k^2]/E[|r_k|]^2$, ensuite la variance est déduite à partir de z. En effet, on peut démontrer (voir appendix A) que [Pie97] :

$$z = \frac{(1+\sigma^2)}{(\sigma\sqrt{\frac{2}{\pi}}\exp(-\frac{1}{2\sigma^2}) + \mathrm{erf}(\sqrt{\frac{1}{2\sigma^2}}))^2} \tag{4.18}$$

Dans l'intervalle [0-6 dB], la variable z a été déterminée pour plusieurs points. En faisant une approximation polynomiale du troisième ordre (largement suffisant) nous pourrons calculer la variance en fonction de z en faisant l'approximation suivante [SWi98] [Pie97] et en supposant que E_s est égale à 1 :

$$\frac{E_s}{\sigma^2} \approx -55.09168 \times z^3 + 247.08933 \times z^2 + -377.2993 \times z + 196.6910 \tag{4.19}$$

L'approximation polynômiale est comparée à la valeur exacte à la figure (4.21). Cette approximation est très efficace. En effet, la courbe en pointillés (voir figure (4.21))

qui est l'approximation est confondue avec la valeur exacte. Donc, afin de pouvoir estimer la variance dans canal AWGN, il faut calculer le rapport z ensuite il suffit de lire dans un tableau (qui est dans la mémoire du processeur) la valeur de la variance correspondante.

Nous avons testé cette méthode pour des blocs de taille 400 bits ($R_t = 1/3$ donc 1200 symboles codés). Nous nous sommes limité à l'intervalle où $E_b/N_0 \in [0 - 3\text{dB}]$. Afin d'obtenir une estimation fiable, nous avons fait la simulation pour un nombre de 10000 blocs. La figure (4.22) montre que la variance moyenne estimée est confondue avec la valeur exacte. À la figure(4.23), nous avons présenté la PDF pour plusieurs valeurs de E_b/N_0 . Nous pouvons déduire que l'écart (offset) l décroît lorsque E_b/N_0 croit. Ajoutant que l'écart entre l est moins important que dans la méthode classique. Par exemple à $E_b/N_0 = 3$ dB , 1% des blocs donnent un écart plus petit que -3 dB ou plus grand que 1.5 dB. Notons aussi que l'écart est quasiment nul pour la majorité des blocs (pour la totalité de l'intervalle $[0 - 3\text{dB}]$) alors que pour la méthode classique la quasi-totalité des estimations fournit un écart positif et non nul.

4.5.2.4 Autres méthodes

Nous avons présenté trois méthodes qui permettent de calculer la variance du bruit dans le canal. Évidemment, d'autres méthodes existent. Nous allons citer, sans entrer dans le détail, trois autres méthodes.

La première a été proposée par Robertson [Rob94]. Roberston a établi une relation entre la variance du logarithme du rapport de vraisemblance LLR et la variance sous la forme suivante :

$$\sigma^2 = \frac{2 + 2\sqrt{1 + m_{2LLR}}}{m_{2LLR}} \tag{4.20}$$

où $m_{2LLR} = E[\{LLR()\}^2]$ est le moment du second ordre (appelé aussi variance) du LLR .L'estimation de la variance se fait à la fin de chaque itération. L'inconvénient de

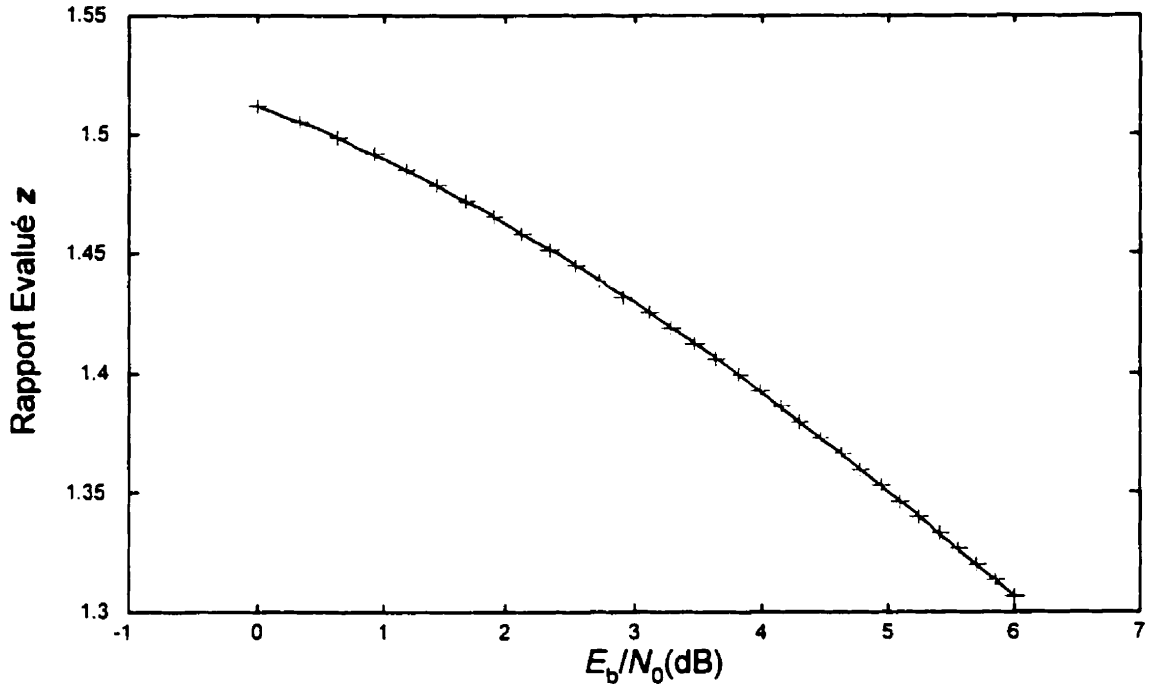

Figure 4.21: Comparaison du rapport z à l'approximation polynomiale.

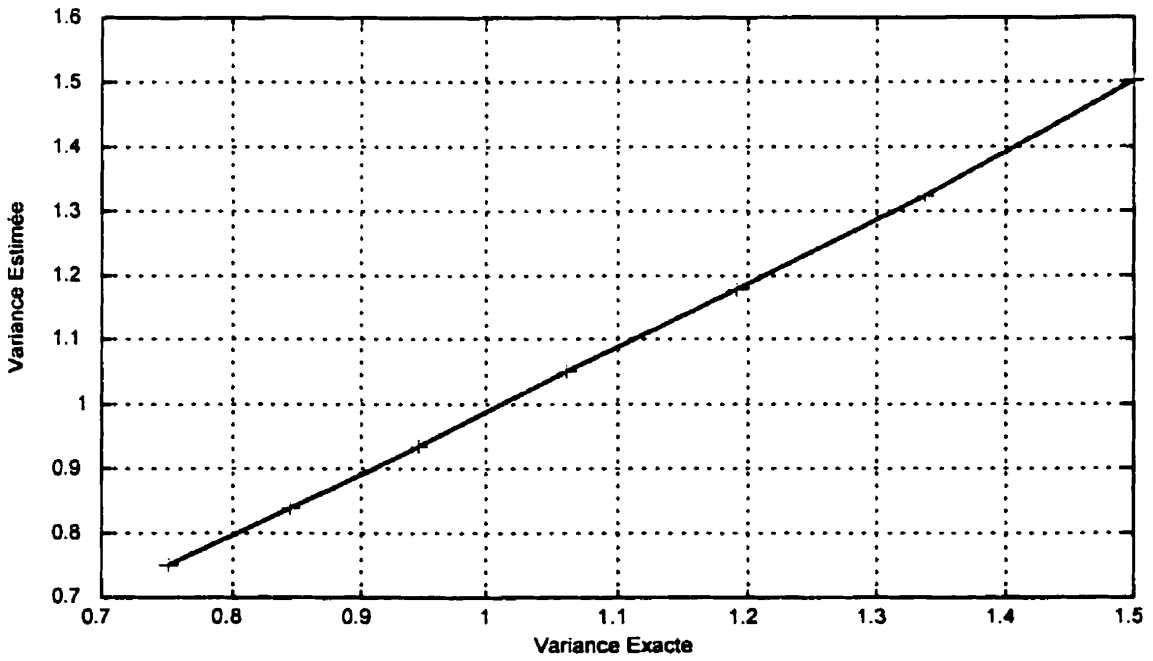

Figure 4.22: Comparaison entre la variance exacte et la variance estimée par la troisième méthode.

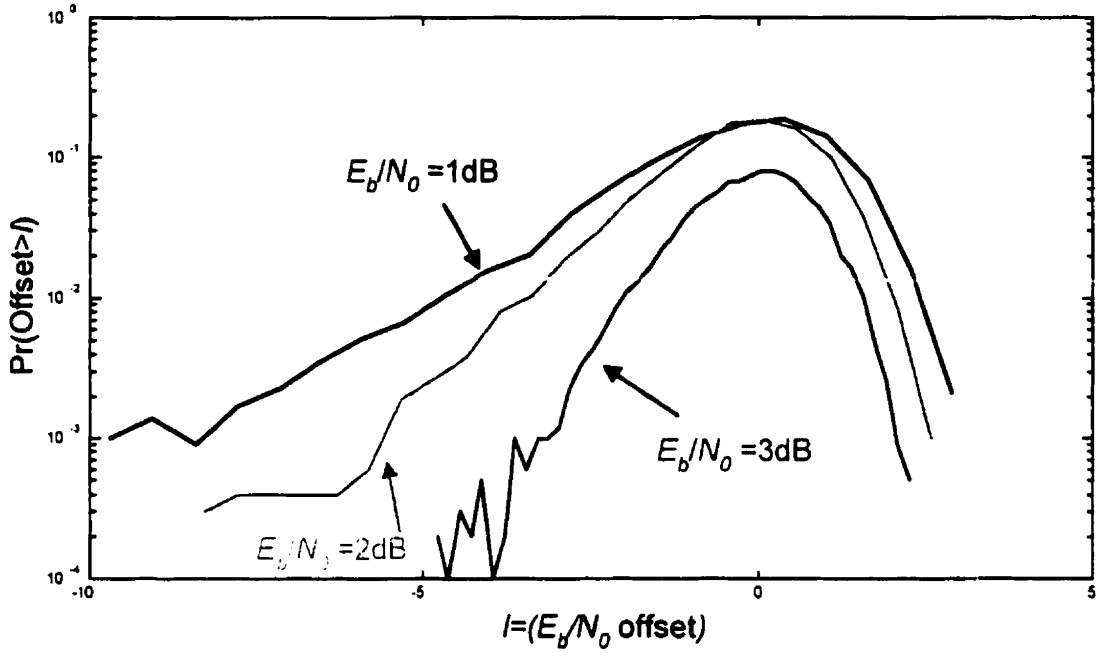

Figure 4.23: PDF de l'écart l pour la troisième méthode pour plusieurs E_b/N_0.

cette méthode est que le LLR est en fonction de la variance, donc la variance sera inconnue pendant la première itération ce qui résultera dans une mauvaise performance d'erreur à la première itération.

La deuxième méthode [RAs97] est similaire à la première mais elle a l'avantage d'être plus simple. En effet, Reed a démontré' la relation suivante :

$$E[LLR()] = \frac{2}{\sigma^2} \tag{4.21}$$

Donc la connaissance de la moyenne du LLR permet de déterminer la variance du bruit à la sortie du décodeur MAP.

La troisième méthode consiste à calculer la variance dans le cas d'un canal AWGN et d'une modulation BPSK, en utilisant l'expression suivante [Hee99] :

$$\sigma_j^2 = [\frac{1}{N}\sum_{k=1}^{N} r_{k,j}]^2 + [\frac{1}{N}\sum_{k=1}^{N}(r_{k,j} - \frac{1}{N}\sum_{k=1}^{N} r_{k,j})^2] - 1 \tag{4.22}$$

D'après Wicker [Hee99], la façon dont l'estimation est faite permet d'éliminer rapidement l'impact d'une fausse estimation au départ de la variance du bruit

4.6 Conclusion

L'échange d'information en général et surtout l'information extrinsèque du LLR entre les décodeurs MAP permet d'améliorer substantiellement les performances des Codes Turbo avec l'accroissement du nombre d'itérations.

Après avoir simulé les performances des Codes Turbo, nous pouvons faire les remarques suivantes :

- L'entrelaceur permet de décorreler et de disperser des erreurs, d'où son rôle vital dans l'amélioration des performances. D'ailleurs, l'augmentation de la taille de l'entrelaceur engendre une amélioration surprenante au niveau des performances d'erreurs.
- Pour les cas simulés ici l'entrelaceur pseudo-aléatoire donne toujours des résultats nettement supérieurs par rapport à l'entrelaceur bloc classique. Son rôle est encore plus important dans le canal de Rayleigh afin de lutter contre les évanouissements.
- Nous pouvons déduire des performances que l'augmentation de la longueur de contrainte ne permet pas forcement d'améliorer les performances, surtout pour des faibles rapports de signal à bruit.
- Une perte de 1.5 dB à 2.5 dB est enregistrée lors du passage d'un canal Gaussian à un canal de Rayleigh.
- La diminution du taux de codage apporte un gain plus grand dans le cas d'un canal de Rayleigh que celui d'un canal AWGN.

Finalement, nous avons aussi présenté les différentes méthodes proposées afin d'estimer la variance du bruit au décodage turbo. La deuxième méthode semble intéressante mais la simplicité de la troisième et son efficacité semble l'avoir emporté pour

la conception des premières puces turbo code. Cependant, lorsque le comportement du bruit dans le canal est inconnu ou le canal n'est pas du type AWGN, la deuxième méthode présente un avantage non négligeable grâce à son efficacité lorsqu'ils'agit de suivre les variations dans le canal grâce en particulier à la boucle de retour (ou de récursion) présentée dans le décodage Turbo. Ceci est particulièrement vrai dans les communications radiomobiles où le canal est très bruité.

Chapitre 5

Terminaison du Treillis

5.1 Introduction

Dans un Code Turbo à concaténation parallèle, la terminaison du treillis est beaucoup plus complexe que celle utilisée pour l'algorithme de Viterbi conventionnel. La terminaison du treillis consiste en une insertion d'une séquence additionnelle connue, aussi appelée une queue du message, dans le but de forcer le codeur à un état connu à la fin du bloc d'information. Dans un schéma de Turbo Code (TC), la séquence de données qui entre dans le premier codeur est entrelacée avant d'être fournie au deuxième codeur. Par conséquent, la queue qui termine le premier codeur ne sera pas adéquate pour terminer le treillis du deuxième codeur. Donc, dans un TC, la présence d'un entrelaceur nous oblige à utiliser deux queues différentes. Etant donnée le caractère récursif des deux codeurs RSC formant le codeur Turbo, la taille de la queue est au plus égale à $(2^{K-1} - 1)$ bits, où K est la longueur de contrainte; par contre pour le VA, la taille de la queue est égale à $(K - 1)$ bits. Ajoutant que dans le cas du VA, la queue consiste simplement à envoyer une séquence de $(K - 1)$ zéros, ce qui n'est pas le cas pour le TC où les bits formant la queue dépendent de l'état où se trouve le codeur à la fin du bloc d'information.

Nous pouvons déjà constater que la transmission de queues ajoute des opérations

supplémentaires à la transmission de l'information et, de plus, présente des difficultés au niveau du deuxième codeur. De nombreux auteurs se sont penchés sur le sujet dans le but de trouver une solution au problème particulier de la terminaison du treillis du deuxième codeur [Rob94][DPo95][RPi96]. Nous allons présenter diverses solutions ou méthodes proposées dans la littérature et ensuite nous allons suggérer une nouvelle alternative qui consiste à ne pas utiliser de queue. Cette alternative est accompagnée d'une modification au niveau de l'initialisation de la BSM. En effet, la question reste de juger de l'utilité de la transmission d'une queue puisque dans plusieurs applications en communications numériques, les données arrivent en continu et, ainsi, la suppression de la queue n'affecte en rien les performances d'erreur. Mais, dans le cas où l'information est subdivisée en petits blocs, la suppression de la queue risque de coûter cher au niveau de la performance d'erreur.

5.2 Les Méthodes De Terminaison

Nous allons décrire diverses méthodes de terminaison de treillis. Tout d'abord nous allons présenter la méthode de terminaison des deux codeurs RSC formant le codeur Turbo, ensuite, nous allons décrire brièvement les diverses façons de terminer l'un des deux codeurs RSC. Finalement, les choix de ne pas terminer (avec ou sans modification de la BSM) le treillis pour aucun des codeurs sont présentés.

5.2.1 Terminaison des deux codeurs (2Tails) :

Lorsque une séquence d'information est transmise, une séquence de bits appelée la queue est envoyée après les symboles codés (représentant la séquence d'information) et ceci afin que l'opération de décodage soit plus fiable. En effet la queue permet au codeur de revenir à l'état initial et donc au décodage, on sait à l'avance que la convergence se fera vers l'état 00. Avec Viterbi, une queue de $(K-1)$ bits est suffisante, mais avec les RSC à cause de la récursivité du codage une queue formée seulement

de zéros n'est pas suffisante pour remettre le codeur à zéro. Par conséquent, il faut calculer la séquence qui permet d'initialiser le codeur lorsque le dernier bit du bloc d'information entre dans le codeur. Ce dernier se trouve dans un état qu'on appelle "l'état final". À partir du diagramme d'état du codeur, on cherche la séquence de bits qui nous ramène de l'état final à l'état zéro initial. Nous pouvons déduire que la queue est, au maximum, de taille $L = (2^{K-1} - 1)$ bits (La pire des possibilités est de parcourir tous les états possibles sauf l'état 0 avant de l'atteindre) et qu'elle contient au moins un bit non nul. Donc, cette méthode consiste à terminer les deux codeurs RSC en utilisant deux queues différentes, chacune de longueur L bits. Soit T_1, la queue de longueur L bits nécessaire à remettre le premier codeur à l'état initial. À cause de l'entrelaceur, T_1 n'est pas la séquence de bits qui permet de terminer le deuxième codeur à l'état initial. En conséquent, une séquence différente T_2 de longueur L est insérée après l'entrelaceur de façon à terminer le deuxième codeur. En supposant que T_1 et T_2 sont transmises au canal, au décodeur Turbo la BSM et la FSM pour les deux décodeurs MAP doivent être inutilisés comme suit :

$$\begin{aligned} A_0(0) &= 0 \\ A_0(m) &= -\infty, m \neq 0 \qquad (5.1) \\ B_{N+L}(0) &= 0 \\ B_{N+L}(m) &= -\infty, m \neq 0 \qquad (5.2) \end{aligned}$$

5.2.1.1 Exemple illustratif :

Soit un codeur de longueur de contrainte $K = 3$; Ce codeur est présenté sous forme systématique et récursive. Ce codeur est illustré à la figure (2.3). Le diagramme d'état du codeur précédent a été présenté à la figure (2.5). S_0 , S_1, S_2 et S_3 correspondent respectivement aux états 00, 01, 10 et 11. Les lignes en gras indiquent que le bit d'information est un 1 et celles en pointillés indiquent que c'est un 0. Supposons que l'état final du codeur est $S_3(11)$. Lorsque le dernier bit d'un bloc d'information est

transmis , une queue est nécessaire afin de remettre les registres du codeur à zéro. On opère de la manière suivante :

1. On vérifie si on est à l'état S_0, si oui, la queue sera formée de zéro.
2. Sinon, on vérifie si l'état suivant (en supposant d'abord, que le bit d'information est un zéro, autrement on suppose que c'est un 1) est l'état initial : Si oui , on s'arrête, autrement l'état final devient l'état suivant telle que le bit d'information est un zéro. Si on est déjà passé par cet état, on prend l'état tel le que le bit d'information est un 1. Si, on est déjà passé par ces deux états précédents, on essaye l'état dont l'état suivant est l'état final (d'abord pour un zéro ensuite pour un 1).
3. On vérifie si on est à l'état S_0, si oui on sarrête, autrement on revient à l'étape 1.

Si on applique ceci au diagramme d'état du codeur de la figure (2.5), la queue est 11 si l'état final du codeur est S_3. D'après l'exemple précédent la longueur de la queue est de 2 bits ($< (2^K - 1) = 3$) donc on va compléter la queue 11 par l'envoi d'un seul 0 et la queue devient la séquence 110. Ceci correspond à la séquence codée (on suppose que le taux de codage est $R = 1/2$) 101100.

5.2.2 Terminaison d'un seul codeur

5.2.2.1 Sans Aucune Modification (1Tail) :

Les conditions 5.1 et 5.2 sont appliquées au premier décodeur. Pour le deuxième décodeur les conditions d'initialisation sont :

$$\begin{aligned} A_0(0) &= 0 \\ A_0(m) &= -\infty, m \neq 0 \end{aligned} \tag{5.3}$$

$$\begin{aligned} B_N(0) &= 0 \\ B_N(m) &= -\infty, m \neq 0 \end{aligned} \tag{5.4}$$

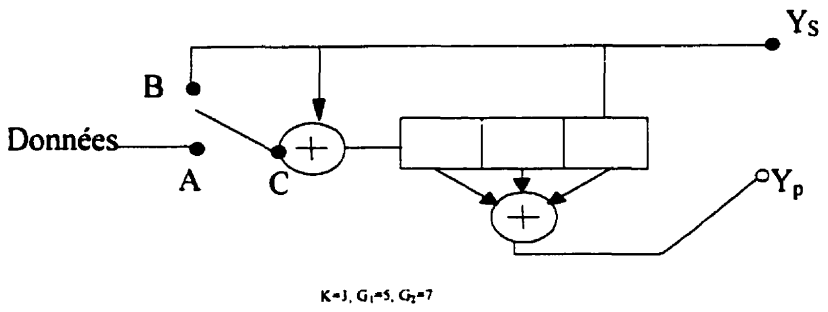

Figure 5.1: Terminaison du treillis suivant la méthode JPL

5.2.2.2 Méthode de Jet Propulsion Lab (JPL) :

L'idée consiste à couper l'information d'entrée lorsque la séquence d'information ou le bloc est complétement transmise. Cette information sera remplacée par le bit provenant de la boucle récursive (du codeur). Donc ce bit va se rajouter à lui même et une queue de $(K-1)$ zéros suffira pour remettre le codeur à l'état initial. La figure (5.1) illustre ceci : le point C est relié au point B pendant la séquence de $(K-1)$ bits (après avoir généré la séquence d'information). D'après [DPo95], le décodeur ignore les $(K-1)$ bits de la queue.

Ceci paraît simple mais une question peut surgir : comment le décodeur va se comporter à la réception de cette queue, d'autant plus que le décodeur ne la connaît pas d'avance la queue.

Méthode de Roberston : Cette méthode consiste à terminer le premier codeur tout en laissant le deuxième codeur "ouvert"[Rob94]. Donc seulement une séquence de longueur N (la longueur du bloc d'information) est transmise pour le deuxième codeur. Au décodage, les paramètres A et B seront initialisés comme suit :

a) au premier décodeur : voir équations (5.1) et (5.2).

b) au deuxième décodeur : la BSM utilise l'état probabiliste généré par la FSM

$A_N.(m)$ du dernier bit N.

$$A_0(0) = 0 \tag{5.5}$$

$$A_0(m) = -\infty, m \neq 0 \tag{5.6}$$

$$B_N(m) = A_N(m) \quad \forall m. \tag{5.7}$$

5.2.2.3 Méthode de Jun et Naßhan

Cette méthode consiste à transmettre une queue pour le deuxième decodeur seulement. L'initialisation des paramètres A et B pour les deux décodeurs se fait suivant les équations (5.1) et (5.2) [JNa94][JNa95].

5.2.3 Aucune terminaison

Aucune terminaison signifie comme son nom l'indique qu'aucune queue a été transmise. Le décodeur de son côté peut ne pas être informé de la suppression de la queue et ceci nous l'avons désigné sous le nom de "Sans aucune modification". Dans le cas où les paramètres initiaux du décodeur MAP sont modifiés afin de signaler d'une manière indirecte que l'état final du codeur est restée inconnue, l'alternative est désignée sous le nom de "Avec modification de la BSM".

5.2.3.1 Aucune terminaison et aucune modification (NoTail) :

Dans cette alternative, aucune queue n'est envoyée pour les deux décodeurs et les conditions (5.8) et (5.9) sont appliquées au premier et deuxième décodeur MAP.

5.2.3.2 Aucune terminaison et modification de la BSM (MNoTail) :

Reed (voir [RPi96]) a trouvé que la suppression de la queue accompagnée d'une légère modification au niveau de la BSM donne d'aussi bonnes performances pour des séquences de petite taille que la possibilité de terminaisons des deux codeurs.

L'avantage de cette méthode est qu'elle élimine la nécessité de transmettre une queue pour les deux décodeurs. La modification de la BSM initiale consiste simplement à l'affecter à une constante égale à $1/2^M$, où 2^M est le nombre d'états dans le treillis.

Revenons à l'expression de la BSM afin de justifier le choix de la modification :

$$\beta_k(m) = P(\mathbf{R}_k^N | S_k = m)$$

L'expression de la BSM dépend de la séquence future $\mathbf{R}_k^N$ qui est inconnue. Par conséquence, le meilleur choix est d'affecter $\beta_k(m)$ à la probabilité d'être à un état m quelconque, $m = 0, 1, ..., (2^M - 1)$. En supposant que les bits de données à l'entrée du codeur soient équiprobables ($\Pr(d_k = 0) = \Pr(d_k = 0) = 0.5$) alors les états dans le treillis seront eux aussi équiprobables et donc l'état final pourrait être un des 2^M états. Pour les deux décodeurs, ceci peut se résumer par :

$$\begin{aligned} A_0(0) &= 0 \\ A_0(m) &= -\infty, m \neq 0 \qquad (5.8) \\ B_N(m) &= M \log 2, \qquad (5.9) \end{aligned}$$

La grande différence entre les équations (5.2) et (5.9) est que dans la condition (5.9) nous ne considérons pas que le codeur s'est terminé à l'état $S = 0$ mais à un état inconnu $S = m$.

5.3 Description des Résultats

Les performances d'erreurs des plusieurs méthodes décrites dans les paragraphes précédents ont été comparées en utilisant des simulations par ordinateurs. Deux codeurs RSC identiques avec une longueur de contrainte $K = 3$ et 5 , taux de codage $R_t = 1/2$ et $1/3$,ont été utilisés. Notons que la longueur des blocs varie de $N = 400$ à $N = 3600$. La perforation a été utilisée afin d'atteindre un taux de codage global

$R_t = 1/2$. La terminologie suivante est utilisée :

2Tails : deux queues différentes de longueur L ont été utilisées pour les deux décodeurs et ce sont les équations 5.1 et 5.2 qui ont été appliquées aux deux décodeurs MAP (voir 5.2.1).

1Tail : Une queue est ajoutée seulement pour le premier décodeur (voir 5.2.2.1). L'autre décodeur n'a pas de queue.

NoTail : Aucune queue n'est envoyée à aucun des deux décodeurs (voir 5.2.3.1).

MNoTail : aucune queue n'est envoyée pour les deux décodeurs mais la modification (5.9) est appliquée (voir 5.2.3.2).

Les résultats des pages suivantes montrent clairement que l'alternative **2Tail** est largement meilleure que les alternatives **1Tail** et **Notail**. Ce constat est vrai quelque soit le type d'entrelacement (bloc ou aléatoire), la longueur de contrainte (3 à 5) et le taux de codage global. Par exemple pour $N = 400$ et $K = 3$ (voir figure 5.6) les alternatives **1Tail** et **NoTail** indiquent au moins 1 dB de dégradation à $E_b/N_0 = 2.5$ dB ce qui est énorme pour des communications par satellite où le moindre dixième de dB est à préserver. De même à $E_b/N_0 = 3$ dB la différence des performances est aussi spectaculaire, elle est de 3 ordres de grandeur. Par ailleurs, les résultats (voir les figures 5.2 à 5.5) indiquent qu'un seuil de performance est atteint rapidement à $E_b/N_0 = 1.5$ dB pour les alternatives **1Tail** et **Notail**. Par conséquent, les alternatives **1Tail** et **NoTail** présentent peu d'intérêt pour le décodeur Turbo et donc nous n'allons pas les prendre en considération comme des techniques de terminaison de treillis car elles ne présentent aucun intérêt pratique.

Nous allons donc nous concentrer sur la comparaison entre l'alternative **2Tails** et la nouvelle alternative proposée **MNoTail**. Comme nous l'avons indiqué précédemment, que la généralisation à partir d'un résultat de simulation est très dangereux surtout dans le codage Turbo où les codeurs et les décodeurs sont très complexes. Afin de pouvoir comparer les deux dernières alternatives, nous avons choisi de faire varier le maximum de paramètres qui entrent dans le schéma des TC. Les paramètres

que nous avons fait varier sont les suivants :

- La longueur de contrainte K
- Le taux de codage global R_t
- Le type d'entrelaceur.
- La longueur du bloc d'information N
- Le type du canal : AWGN ou Rayleigh.

Évidemment, nous aurions pu aussi faire varier les vecteurs générateurs du codeur RSC mais cela nous paraît inutile car l'effet des vecteurs générateurs va influencer les poids du code et donc les performances d'erreur, qu'une queue de bits soit transmise ou non. Tout d'abord, nous allons nous placer dans un canal AWGN où le code n'a pas été perforé : Prenons l'exemple de la figure 5.2, où l'entrelacement utilisé est aléatoire, l'alternative **MNoTail** donne des performances d'erreur identiques à l'alternative de **2Tails**.

La variation du taux de décodage tout en laissant fixe les autres paramètres ($K = 3$, $N = 1024$, type du canal est AWGN, type d'entrelaceur : aléatoire) comme le montre la figure (5.3), n'affecte en rien les conclusions précédentes mais au contraire, nous pouvons remarquer que l'alternative **MNoTail** est légèrement supérieure à l'alternative **2Tails** au niveau de la performance d'erreurs.

La longueur du bloc d'information N est un autre paramètre important qui permet de consolider l'efficacité de l'alternative **MNoTail**. En fixant $K = 3, R_t = 1/2$, en gardant l'entrelaceur aléatoire et le même type de canal, la variation de N (196 (figure 5.4),400 (figure (5.5), 3600 (figure (5.7)) confirme l'efficacité de la **MNoTail**. Nous pouvons remarquer qu'il a une très légère dégradation au niveau de la performance d'erreurs pour une partie des rapports signal à bruit supérieur à 1.5 ou 2 dB (ceci est particulièrement vrai pour $N = 196$ et $N = 400$) tandis que pour les autres valeurs de N les performances d'erreurs pour la **MNoTail** et la **2Tails** sont identiques.

La variation de la longueur de contrainte K (de 3 à 5) n'affecte pas non plus le comportement décrit précédemment au niveau de la performance d'erreur. En effet

l'alternative **MNoTail** est légèrement supérieure au niveau de la performance d'erreur la **2Tails** (voir figures (5.12) et (5.13)).

Nous avons aussi changé le type de canal (AWGN à un canal de Rayleigh) afin d'observer l'effet du changement sur la modification de la queue (**MNoTail**). Les figures (5.9), (5.8) et (5.11), nous confirment une fois de plus la robustesse de l'alternative **MNoTail**. Donc dans un canal radiomobile de type Rayleigh, l'alternative **MNoTail** se montre à la hauteur de celle de **2Tails** au niveau de la performance d'erreur (pour de blocs de petites et moyennes tailles, et pour un entrelacement aléatoire).

Finalement, après avoir utilisé exclusivement un entrelaceur aléatoire, nous avons remplacé ce dernier par un entrelaceur bloc. Malheureusement, la performance d'erreur de l'alternative **MNoTail** se dégrade (voir à la figure 5.10). Nous avons constaté le même comportement pour $N = 196, 400$ dans le cas d'un canal AWGN. Mais dans le cas où N est supérieur à 1000, la dégradation de la **MNoTail** par rapport au **2Tails** est négligeable. Donc, l'entrelaceur bloc démontre une fois de plus ses limites dans le but d'éparpiller les erreurs en bloc (burst error) surtout dans le cas de blocs de petites tailles.

L'inefficacité des entrelaceurs blocs classiques à combattre les erreurs surtout dues à la queue a été mentionnée et étudiée par [LCh97]. Ling et Cheug ont étudié l'effet de l'entrelaceur bloc sur la queue mais pour un décodeur SOVA. Ils affirment que le décodage itératif utilisant SOVA et un entrelacement bloc traditionnel est incapable de réduire la probabilité d'erreur par bit surtout en ce qui concerne les bits qui se trouvent près de la fin du bloc ou séquence d'information. La dégradation de la performance d'erreur s'accentue lorsque le rapport signal à bruit est élevé (nous pouvons remarquer les même effets à la figure 5.10). Lin (voir [LCh97]) propose d'autres types d'entrelaceur que l'aléatoire qui reste le plus performant afin de lutter contre la dégradation :

- Entrelaceur rotationel bloc classique (Rotated classical block).

- Entrelaceur bloc classique en arrière (backward classical bloc).
- s-aléatoire (s-random).

5.4 Conditions de Simulation

Il est important de décrire brièvement les conditions dans lesquelles, nous avons fait nos simulations :

Un simulateur Turbo programmé en language C a été utilisé comme base de nos simulations. Une partie des programmes de ce simulateur est inspirée des programmes de Couleaud (voir [Cou95]). en guise de référence, on peut trouver ce document sur l'internet à l'adresse suivante : "http ://charli.levels.unisa.edu.au/~steven/turbo/" ; nom du fichier : "jyc.ps.gz "ou "jyc.c.gz".

Ce même simulateur a été programmé en language Matlab dans le but de tester rapidement et efficacement les chargements en cours. Notons que le simulateur programmé en C est au mois 5 fois plus rapide que celui programmé en Matlab (les programmes sont fournis à l'annexe C).

Les machines utilisées sont surtout des SUN Sparc (4,5 10 et 20) mais aussi des PC (166 MMX).

5.5 Conclusion

Les différentes façons de terminer un treillis d'un codeur Turbo ont été présentées dans ce chapitre. Nous nous sommes surtout intéressés à comparer quatre alternatives de terminaison dans un premier temps, ensuite nous nous sommes concentrés à comparer l'alternative de terminaison des deux codeurs **2Tails** et l'alternative **MNoTail**. Nous avons ainsi comparé les performances de ces divers méthodes. Nous pouvons conclure des résultats des simulations :

- La **MNoTail** est parmi les meilleures techniques de terminaison. Elle est au

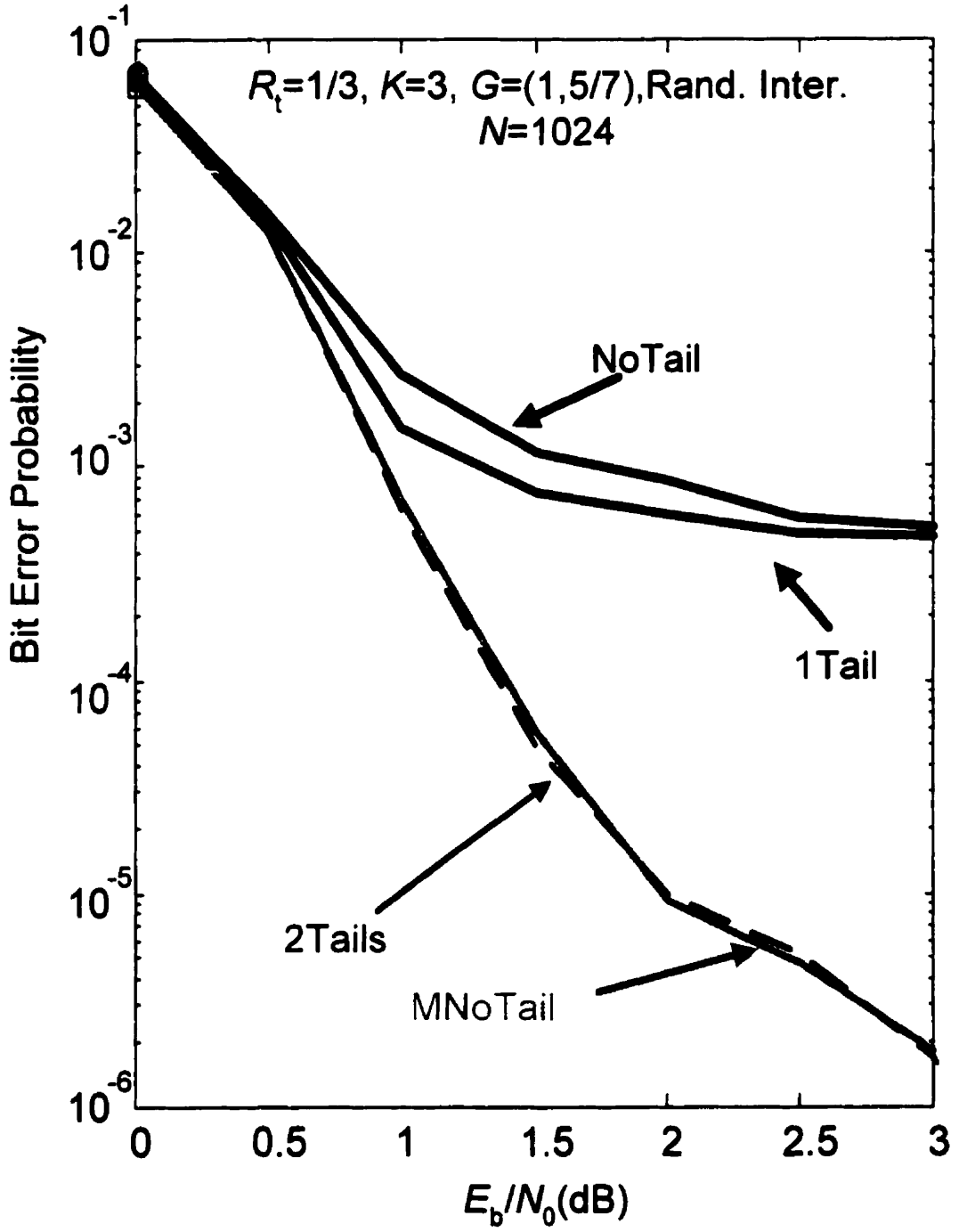

Figure 5.2: Comparaison des quatres alternatives **2Tails**, **1Tail**, **NoTail** et **MNoTail** dans un canal AWGN, pour $K = 3$, $R_t = 1/3$, $N = 1024$, Entr. Aléatoire.

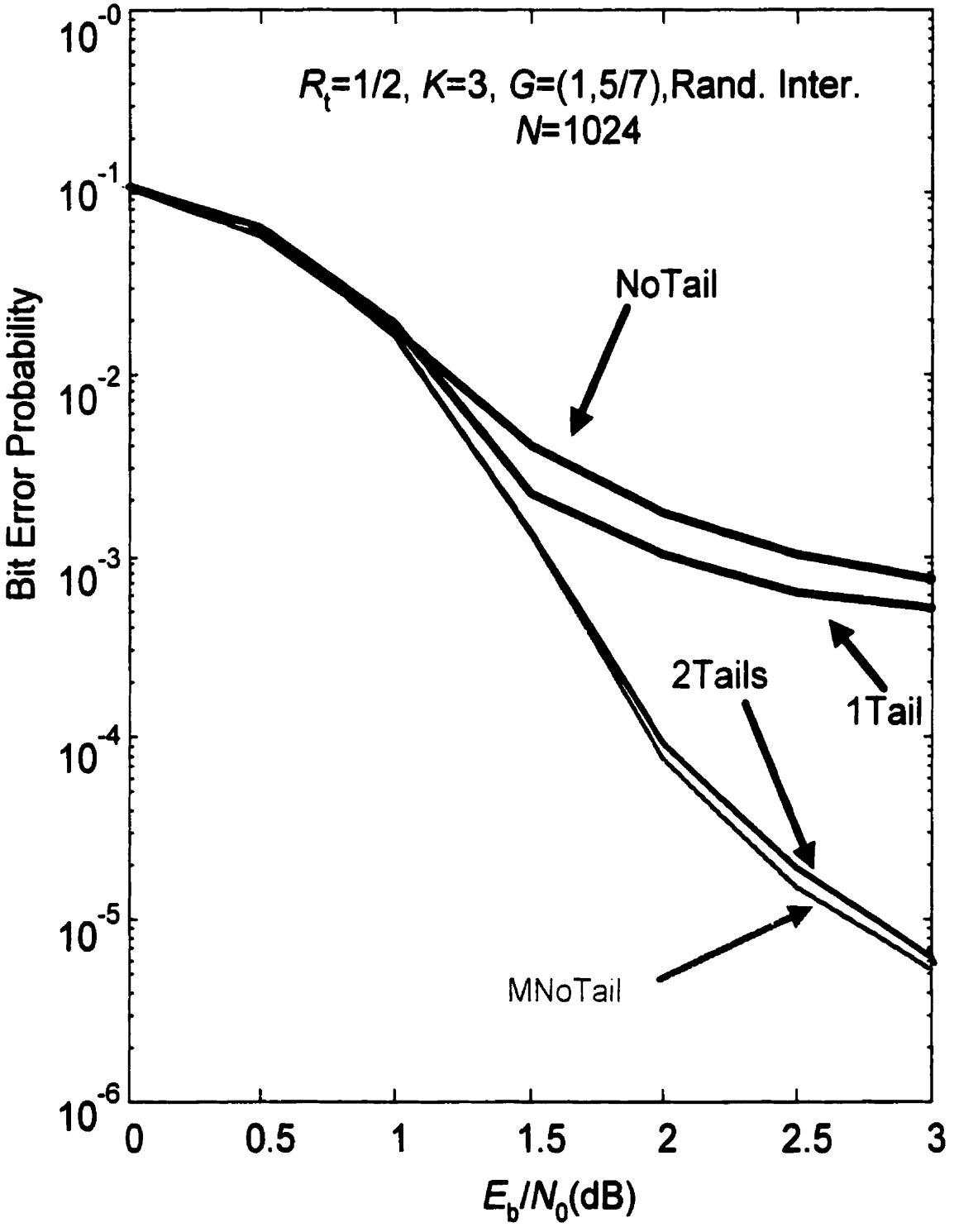

Figure 5.3: Comparaison des quatres alternatives **2Tails**, **1Tail**, **NoTail** et **MNoTail** dans un canal AWGN, pour $K = 3$, $R_t = 1/2$, $N = 1024$, Entr. Aléatoire.

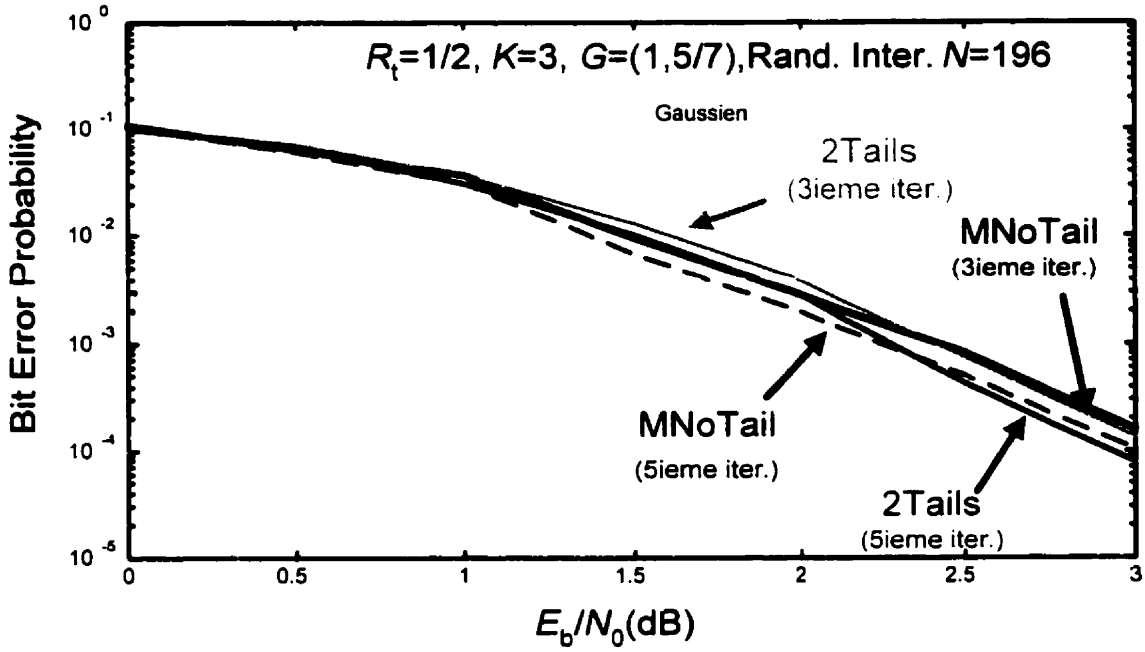

Figure 5.4: Comparaison entre les deux alternatives de terminaison **2Tails** et **MNoTail** dans un canal AWGN, pour $K = 3$, $R_t = 1/2$, $N = 196$, Entr. Aléatoire.

mois aussi performante que l'alternative **2Tails**. La **MNotail** présente l'avantage qu'aucune queue est ajoutée à la séquence d'information et donc un gain de codage de $10log\frac{N+L}{N}$ est atteint. Ce gain est d'autant plus important que la taille de la séquence d'information est petite.

- Nous avons démontré que la **MNoTail** est pratique pour différentes tailles de bloc, longueur de contrainte, taux de codage et type de canal.
- Les alternatives **NoTail** et **1Tail** sont au moins de 1.5 dB moins bien que la **MNOTail** pour E_b/N_0 supérieur à 2.5 dB. En conséquence ces deux approches doivent être éliminées.
- Contrairement au cas des séquences de petites tailles (cf. [RPi96]), la **MNotail** donne les meilleures performances d'erreurs pour les probabilités d'erreurs par bit modérées à élevées dans le cas des séquences de tailles moyennes.
- L'entrelacement bloc est à éliminer dans le cas de l'utilisation de l'alternative

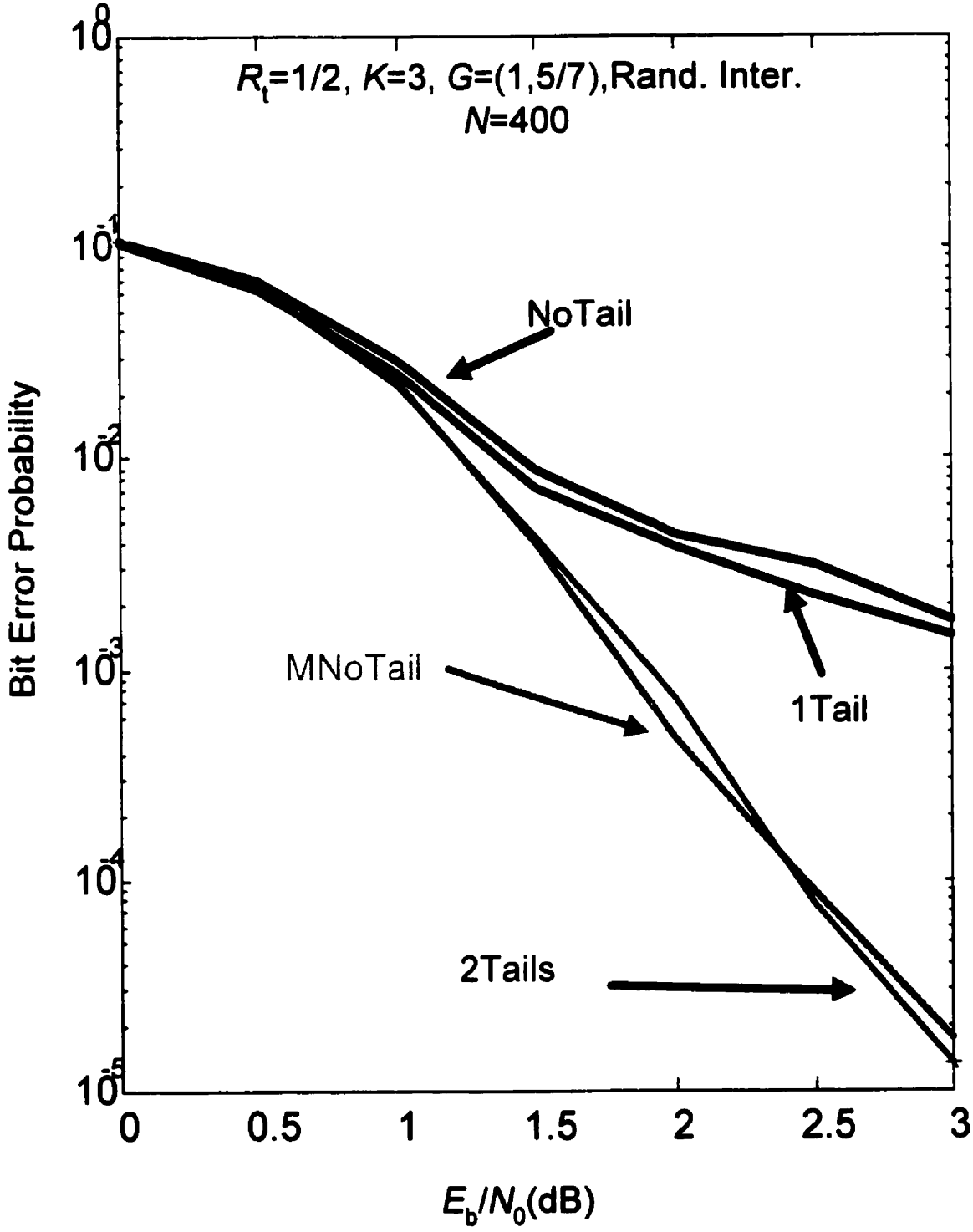

Figure 5.5: Comparaison des quatres alternatives **2Tails**, **1Tail**, **NoTail** et **MNoTail** dans un canal AWGN, pour $K = 3$, $R_t = 1/2$, $N = 400$, Entr. Aléatoire.

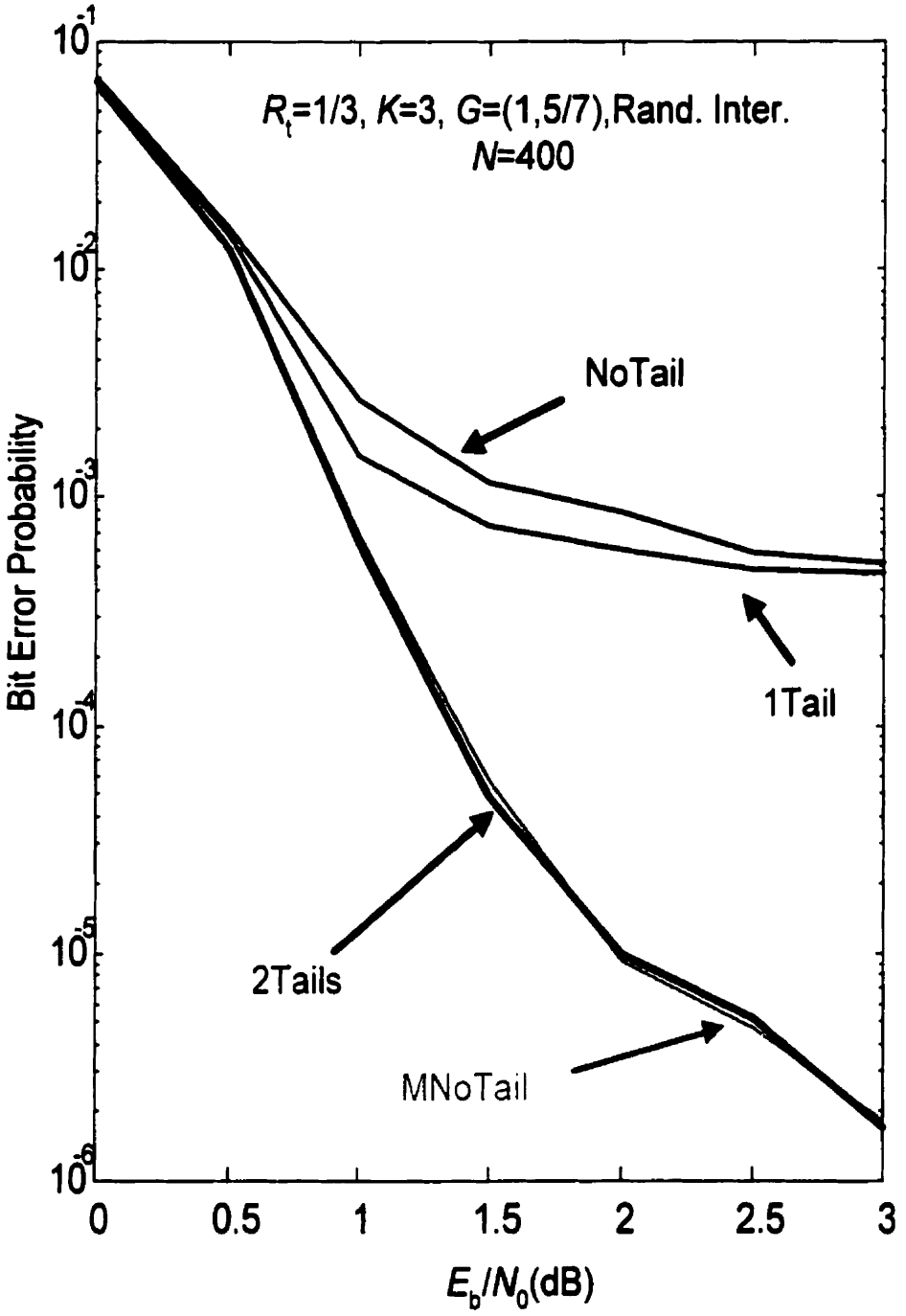

Figure 5.6: Comparaison des quatres alternatives **2Tails**, **1Tail**, **NoTail** et **MNoTail** dans un canal AWGN, pour $K = 3$, $R_t = 1/3$, $N = 400$, Entr. Aléatoire.

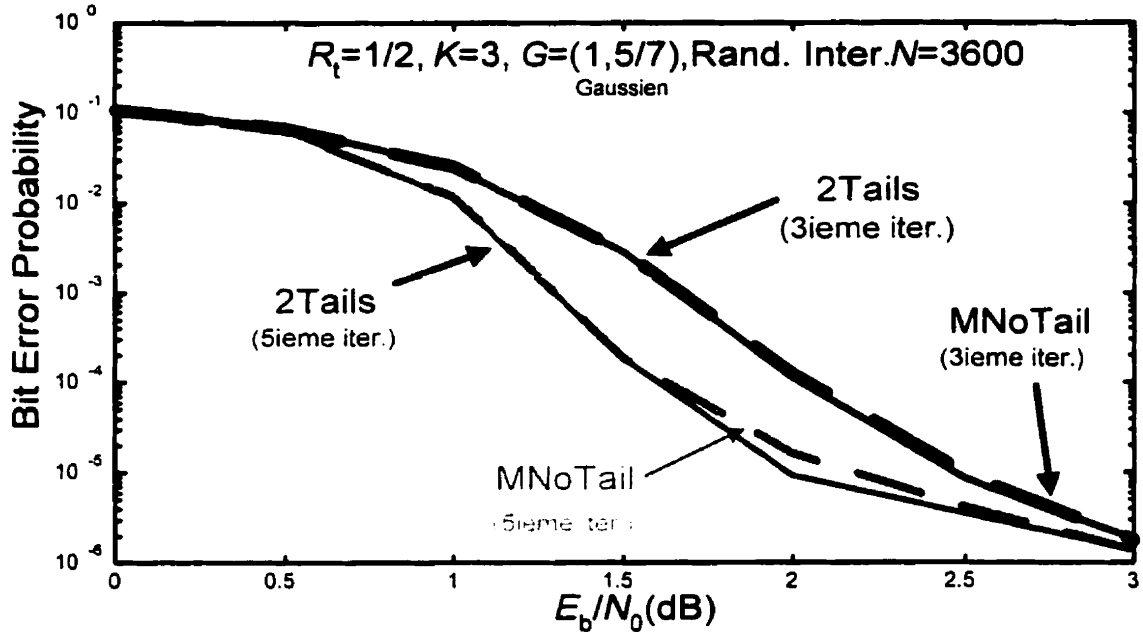

Figure 5.7: Comparaison entre les deux alternatives de terminaison **2Tails** et **MNoTail** dans un canal AWGN, pour $K = 3$, $R_t = 1/2$, $N = 3600$, Entr. Aléatoire.

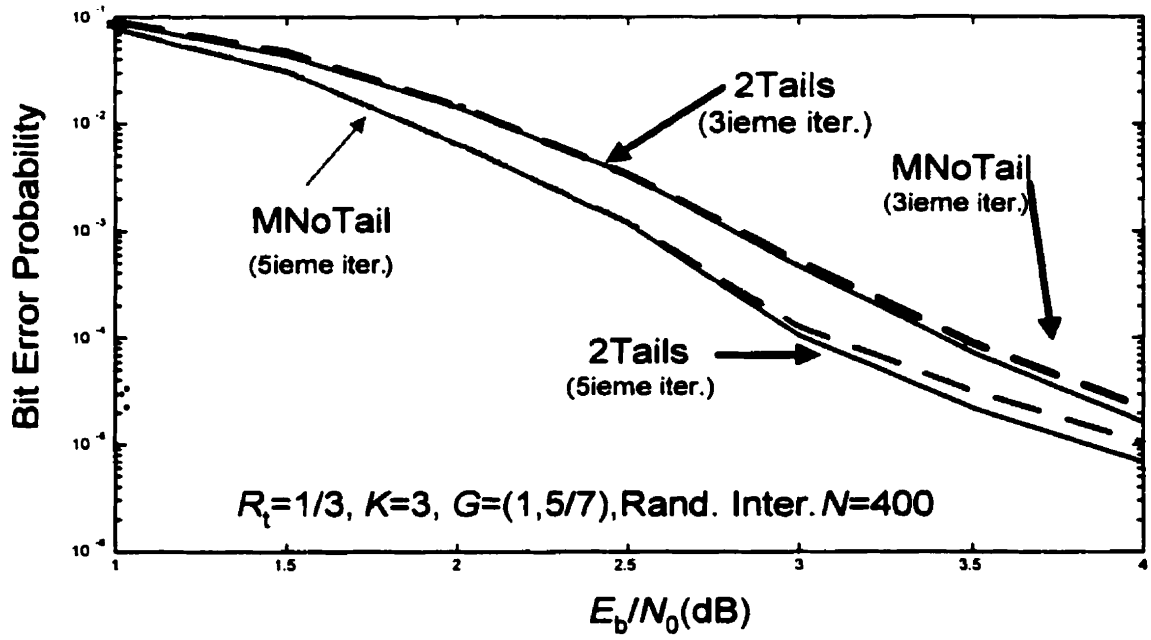

Figure 5.8: Comparaison entre les deux alternatives de terminaison **2Tails** et **MNoTail** dans un canal Rayleigh, pour $K = 3$, $R_t = 1/3$, $N = 400$, Entr. Aléatoire

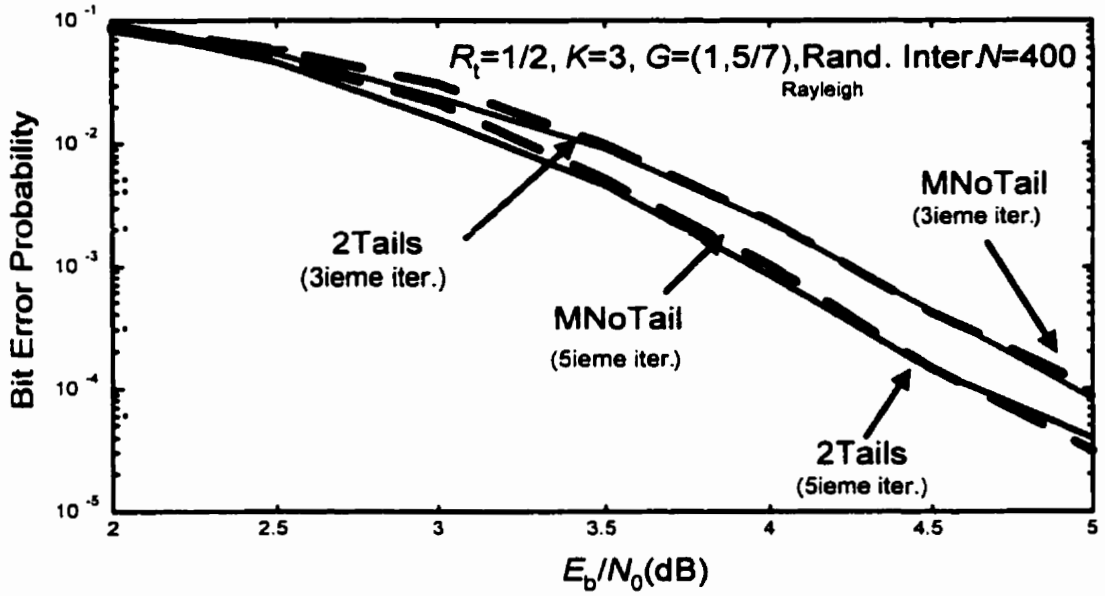

Figure 5.9: Comparaison entre les deux alternatives de terminaison **2Tails** et **MNoTail** dans un canal Rayleigh, pour $K = 3$, $R_t = 1/2$, $N = 400$, Entr. Aléatoire.

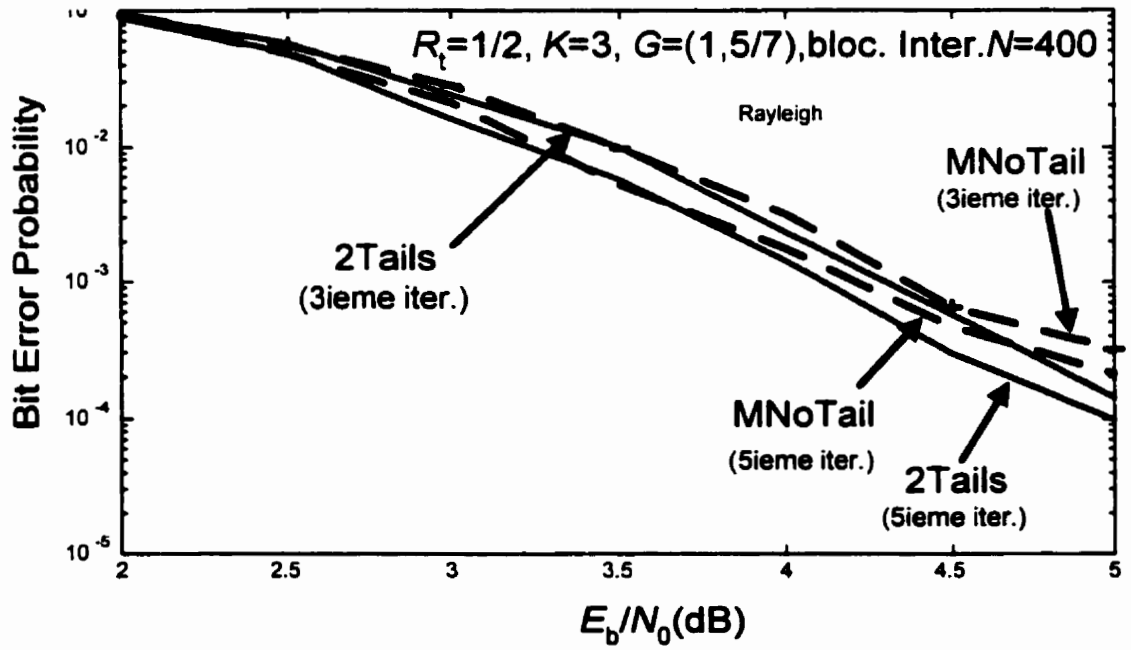

Figure 5.10: Comparaison entre les deux alternatives de terminaison **2Tails** et **MNoTail** dans un canal Rayleigh, pour $K = 3$, $R_t = 1/2$, $N = 400$, Entr. Bloc Classique.

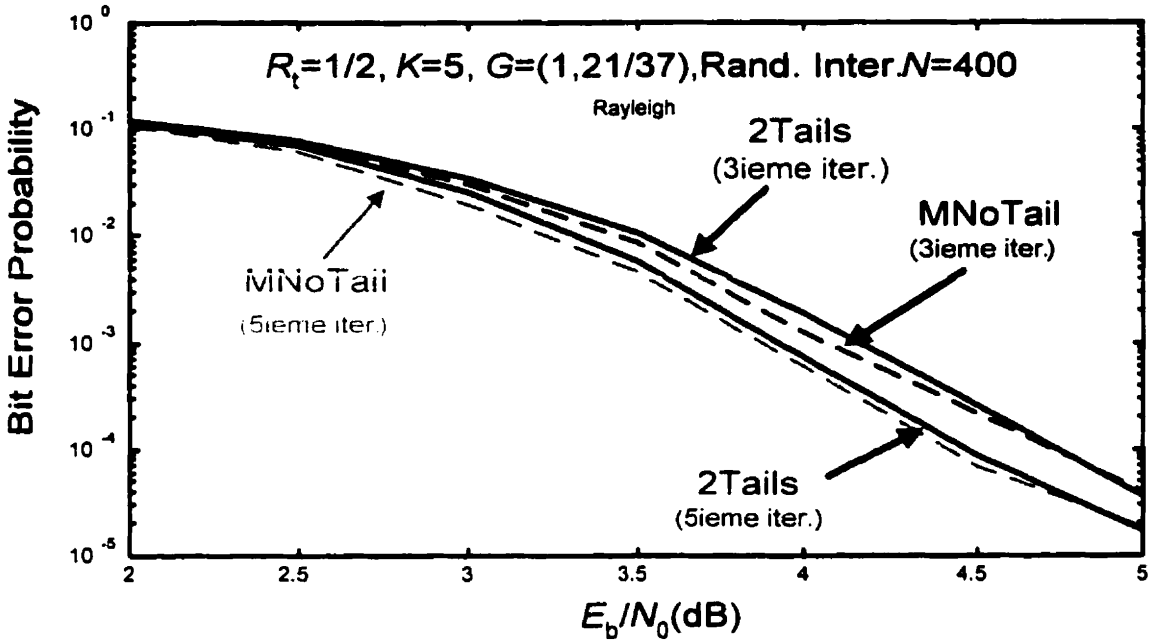

Figure 5.11: Comparaison entre les deux alternatives de terminaison **2Tails** et **MNoTail** dans un canal Rayleigh, pour $K = 5$, $R_t = 1/2$, $N = 400$, Entr. Aléatoire.

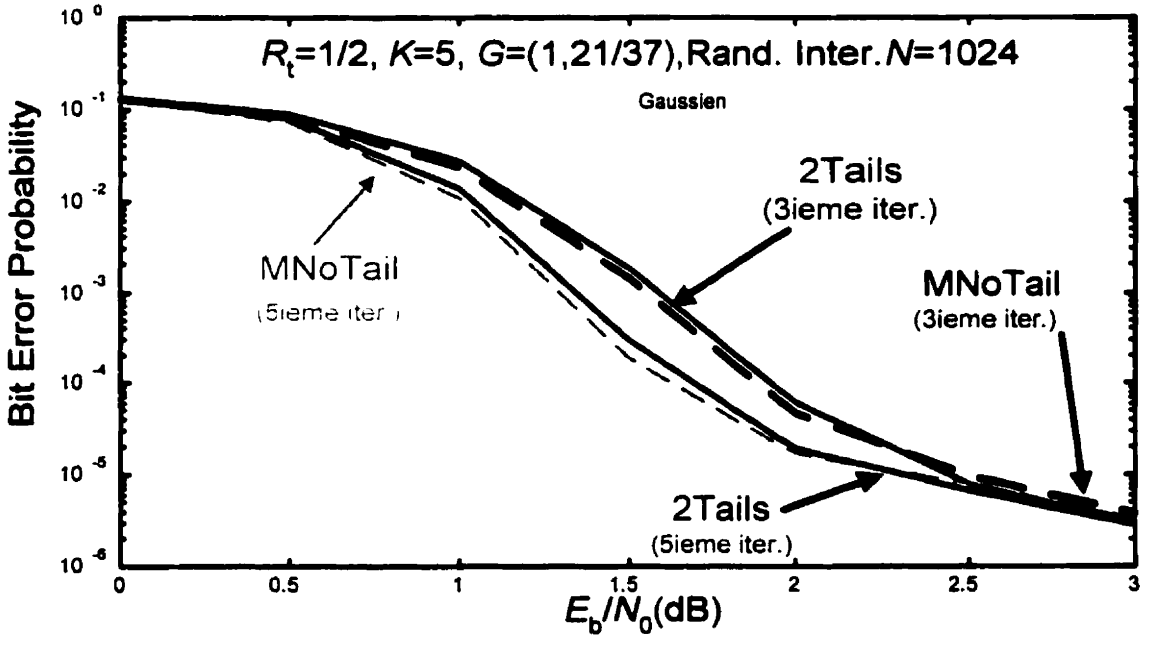

Figure 5.12: Comparaison entre les deux alternatives de terminaison **2Tails** et **MNoTail** dans un canal AWGN, pour $K = 5$, $R_t = 1/2$, $N = 1024$, Entr. Aléatoire.

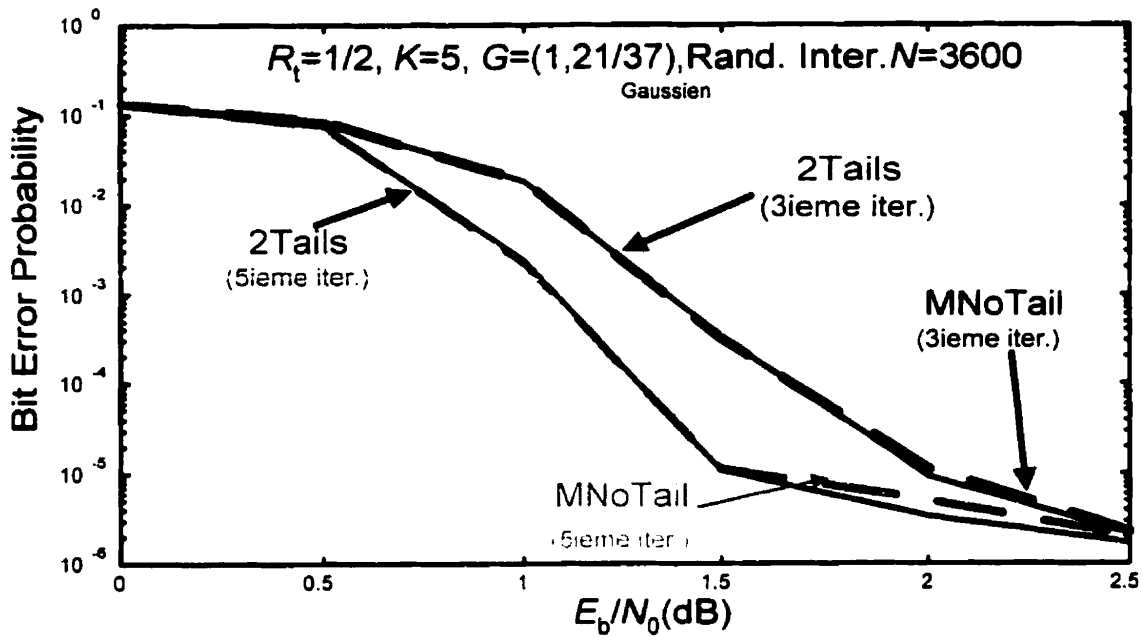

Figure 5.13: Comparaison entre les deux alternatives de terminaison **2Tails** et **MNoTail** dans un canal AWGN, pour $K = 5$, $R_t = 1/2$, $N = 3600$, Entr. Aléatoire.

MNotail.

Chapitre 6

Conclusions

Simplifier l'algorithme de décodage MAP pour les Codes Turbo tout en atteignant des performances d'erreurs approchant la limite de Shannon constitue un grand défi qui fait le sujet des recherches actuelles. Néanmoins, nous pouvons nous demander si l'algorithme MAP va permettre d'atteindre cet objectif. A notre avis, la réponse est plus simple qu'elle ne paraissait en 1993, année de la découverte des Codes Turbo. En effet, la limite de la simplification de l'algorithme MAP est quasiment atteinte. Évidemment, il est toujours possible d'apporter des simplifications minimes dans les prochaines années. Cependant ces simplifications ne permettront probablement pas d'atteindre la simplicité de l'algorithme de Viterbi.

Dans le domaine du VLSI qui est le domaine de la conception matérielle des puces, une nouvelle technologie apparaît tout les six mois. Par conséquent, il serait peut être inutile de simplifier l'algorithme MAP. Surtout dans le cas où les technologies futures permettraient d'augmenter la vitesse de transmission et d'implémenter des algorithmes de grande complexité à un prix de commercialisation peu élevé. Actuellement, pour un taux de codage de 1/3, $K = 5$, $N = 65536$ opérant à un taux de 356 kbit/s, la valeur du rapport E_b/N_0 nécessaire pour atteindre un BER de 10^{-5} est de 0.32 dB.

En plus de la complexité de l'algorithme MAP ou celle de l'algorithme Log-MAP,

l'algorithme MAP nécessite l'estimation de la variance du bruit dans le canal (tel que montré dans le chapitre 4). La méthode classique ou conventionnelle n'est pas très précise. La deuxième méthode d'estimation est la plus performante mais la simplicité de la troisième méthode et son efficacité semble l'avoir emporté pour la conception des premières puces des décodeurs Turbo. Cependant, lorsque le comportement du bruit dans le canal est inconnu ou que le canal n'est pas du type AWGN, la deuxième méthode présente un avantage non négligeable grâce à son efficacité lorsqu'il s'agit de suivre les variations dans le canal grâce en particulier à la boucle de retour (ou de récursion) présente dans le décodage Turbo. Ceci s'applique particulièrement aux communications radiomobiles où le canal est très bruité.

La présence de l'entrelaceur entre les deux décodeurs permet de disperser les erreurs et a posteriori d'augmenter les performances. Cependant, sa présence au niveau de codeur complique la terminaison du treillis (chapitre 5). En effet, deux queues distinctes sont nécessaires pour la terminaison du treillis. La **MNoTail** est parmi les meilleures techniques de terminaison. Elle est au moins aussi performante que l'alternative **2Tails**. La **MNotail** présente l'avantage qu'aucune queue est ajoutée à la séquence d'information et donc un gain de codage est atteint. Ce gain est d'autant plus important que la taille de la séquence d'information est petite. La **MNoTail** est pratique pour différentes tailles de bloc, longueurs de contrainte, taux de codage et type de canal. Cependant, l'entrelacement bloc est à éliminer dans le cas de l'utilisation de l'alternative **MNotail**. Cette dernière donne les meilleures performances d'erreurs pour les probabilités d'erreurs par bit modérées à élevées dans le cas des séquences de tailles moyennes.

6.1 Suggestions pour des recherches futures

Réduire la complexité de l'algorithme de décodage ou trouver un algorithme de décodage sous-optimal sont parmi les principales préoccupations des chercheurs dans

le domaine du codage de canal. C'est dans la même lignée que nos suggestions pour les recherches futures s'orientent :

- Il serait intéressant d'examiner l'idée de Franz et Anderson [FAn98] qui permet de réduire le nombre de calculs dans l'algorithme MAP en imposant un seuil limite aux paramètres de décodage (BSM, FSM). Les auteurs annoncent une réduction par facteur de 4 ou 5 des opérations nécessaires à l'algorithme MAP. Ceci permettra de réduire le délai. Néanmoins au niveau de l'implémentation de l'algorithme la difficulté restera la même.
- McEliece, MacKay et Cheng [MMC98] d'une part, Kschinschang et Frey [KF98] d'autre part ont découvert indépendamment une connexion entre le décodage turbo et Belief Propagation (BP) dans la théorie des graphes qui est du domaine de l'intelligence artificielle. L'interprétation du BP a conduit à des algorithmes parallèles de décodage dont les performances d'erreurs sont meilleurs que celle des codes Turbo. Ces algorithmes de graphes semblent permettrede se rapprocher de la solution [Hee99] et l'atteinte de la limite de Shannon. Il serait grandement intéressant d'étudier ces algorithmes afin de pouvoir élucider les raisons des bonnes performances des Codes Turbo.
- L'idée de combiner le codage de source avec le codage correcteur d'erreur reste intéressante. Plusieurs tentatives dans ce sens ont été faites [Hag95]. C'est une autre voie de recherche qui est loin d'être encore terminée.

Bibliographie

[1] Livres

[Bha85] Bhargava, V., Haccoun, D., *Digital Communications By Satellite.* New York : John Wiley, 1981.

[For66] Forney, G. D., Jr.,*Concatenated Codes.* Cambridge : MIT Press, 1966.

[Hee99] Heegard, C., Wicker, S. B., *Turbo Coding*, Kluwer Academic Publishers, 1999.

[Meh94] Mehrotra, A., *Cellular Radio Performance Engineering.* Artech House, Inc., 1994.

[Pro95] Proakis, J.G., *Digital Communication.* Third edition, Mc Graw-Hill, 1995.

[Skl88] Sklar, B., *Digital Communications, Fundamentals and Applications.* Englewood Cliffs : Prentice Hall, 1988.

[Wic95] Wicker, S. B., *Error Control Systems for Digital Communications and Storage.* Englewood Cliffs : Prentice Hall, 1995.

[2] Articles

[BCJ74] Bahl, L., Cocke, J., Jelinek, F., and Raviv, J., "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inf. Theory*, pp. 284-287, Mar. 1974.

[Bar96] Barbulescu, S. A., "Iterative Decoding of Turbo Codes and other concatenateed codes", *Dissertation, University of South Australia*, February 1996.

[BMD96] Benedetto, S., Montorsi, G., Divsalar, D., and Pollora, F., "Soft-Output Decoding Algorithms in Iterative Decoding of Turbo Codes", *TDA Progress Report* 42-124, February 15, 1996.

[BMo96] Benedetto, S., and Montorsi, G. "Unveiling Turbo-Codes : Some Results on Parrallel Concatenated Coding Schemes", *IEEE Trans. Inform. Theory*, vol. 43, pp. 409-428, Mar. 1996.

[BDM96] Benedetto, S., Divsalar, D., Montorsi, G., and Pollora, F., "Soft Input Soft Output MAP module to decode parrallel and serial concatenated codes", *TDA Progress Report* 42-127, 1996.

[BGT93] Berrou, C., Glavieux, A., and Thitimajshima, P., "Near Shanon limit error-correcting coding : Turbo codes," *Proc.IEEE Int. Conf. Commun.*, Geneva, Switzzerland, pp. 1064-1070, 1993.

[BGl96] Berrou, C., Glavieux, A., "Near optimun error-correcting coding and decoding : turbo codes", *IEEE Trans. Commun.*, vol. 44, pp. 1261-1271, 1996.

[Bou97] Bouzouita, N., "Sur le décodage itératif des codes turbo", mémoire de maîtrise, *Ecole Polytechnique de Montréal*, juin 1997.

[Cou95] Couleaud, J. Y., "High gain coding schemes for space communications," ENSICA Final Year Report, *Uni. of South Australia*, Sep. 1995.

[DPo95] Divsalar, D., and Pollora, F., "Turbo Codes for Deep-Space Communications", *TDA Progress Report* 42-120, February 15, 1995.

[FAn98] Franz, V., Anderson, B., "Concatenated Decoding with a Reduced-Search BCJR Algortihm", *IEEE J.S.A.C*, vol. 16, pp. 186-195, Feb. 1998.

[Hag95] Hagenauer, J., "Source-controlled channel decoding", *IEEE Trans. Commun.*, vol. 43, no. 9, pp. 2449-2457, September 1995.

[Hag96] Hagenauer, J., "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 429-445, Mar. 1996.

[Hag97] Hagenauer, J., "Turbo principle : Tutorial introduction and state of the art", *Proceedings of the Int. Symp. on Turbo Codes & related topics*, Brest, France, 3-5 Sept. 1997.

[Hal96] Hall, E.K., "Peformance and design of turbo codes on Rayleigh fading channels", *Masters thesis, Univ. Virginia*, charlottsville, 1996.

[HWi98] Hall, E.K., and Wilson, S.G., "Design and Analysis of Turbo Codes on Rayleigh Fading Channels", *IEEE J.S.A.C*, vol. 16, pp. 160-174, Feb. 1998.

[HPG98] Ho, M. S. C., Pietrobon, S. S., and Giles, T., "Improving the constituent codes of turbo encoders", *IEEE Global Telecommun. Conf.*, vol. 6, pp. 3525-3529, Sydney, Australia, Nov. 1998.

[JNa94] Jung, P., and Naβan, M., "Performance evaluation of turbo codes for short frame transmission systems", Electronics Letters, vol. 30, pp. 111-112, Jan. 1994.

[JNa95] Jung, P., and Naβan, M., "Comprehensive comparison of turbo-code decoders", *IEEE Vehivular Tech. Conf.*, pp. 645-649, 1995.

[KF98] Kschischang, F.R., Frey, B.J., and Cheng, J.-F., "Iterative Decoding of Compound Codes by Probability Propagation in Graphical Models", *IEEE J.S.A.C*, vol. 16, pp. 219-230, Feb. 1998.

[Lab87] Lab-Volt Ltd., "Digital Communications : Modems and Data Transmisssion", Volume 3, Oct. 1989.

[LCh97] Lin, L., and Cheng, R. S., "On the tail effect of SOVA-based decoding for Turbo codes", *Proc. GLOBECOM 97*, Phoenix, AZ, USA, pp. 644-648, November 1997.

[Mad96] Madarel, F., "Simulation and optimisation of the turbo decoding algorithm", Final Year Project Report, *University of south Australia* Nov. 1996.

[MMC98] McEliece, R.J., MacKay, D.J.C., and Cheng, J.-F., "Turbo Decoding as an instance of Pearl's Belief Propagaion Algorithm", *IEEE J.S.A.C*, vol. 16, pp. 140-152, Feb. 1998.

[PBa96] Pietrobon, S. S., and Barbulescu, S. A., "A simplification of the midified Bahl decoding algorithm for systematic convolutional codes", *Int. symp. Inform. Theory & its applications*, pp. 1073-1077, Sydney, Australia, Revised january 1996.

[Pie96] Pietrobon, S. S., "Efficient implementation of continuous MAP decoders and a synchronisation technique for turbo decoders", *Int. Symp. on Inform. Theory and its applications*, pp. 586-589, Victoria, BC, Canada, Sep. 1996.

[Pie97] Pietrobon, S. S., "Implementation and performance of a turbo/map decoder", *Int.J. Satellite Commun.*, Feb. 1997.

[RPi96] Reed, M., and Pietrobon, S., "Turbo-Code termination schemes and a novel alternative for short frames" *PIMRC 96*, Taipei, Taiwan, Oct. 1996.

[RA097] Reed, M., and Asenstofer, J., "A novel variance estimator for turbo-code decoding", *International Conference on Telecommunications*, pp. 173-178, April 1997.

[RAs97] Reed, M., and Asenstofer, J., "Numerical Analysis of the Maximum A Posteriori Algorithm", Allerton Conf. on Commun., Control, and Computing, Monticello, USA, Sep.-Oct. 1997.

[Rob94] Roberston, P., "Illuminating the structure of decoders for parellel concatenated recursive systematic (turbo) codes", *IEEE Globecom conf.*, pp.1298-1303, San fransisco, 1994.

[RHV97] Roberston, P., Hoeher, P., and Villebrun, E., "Optimal and sub-optimal maximun a Posteriori algorithms suitable for turbo decoding", *European Transactions on Telecommunications, ETT* vol.8, Mar.-Apr. 1997.

[Ryan] Ryan, W. E., "A turbo code tutorial" Unpublished Papers, http ://www.ece.arizona.edu/~ryan/.

[SWi98] Summers, T. A., and Wilson, S. G., "SNR Mismatch and On-Line Estimation in Turbo", *IEEE Trans. Commun.*, vol. 46, pp. 421-423, 1998.

[Skl97] Sklar. B.. "A primer on turbo code concepts", *IEEE Comm. Magazine*, pp. 94-102, Dec. 1997.

[Vit98] Viterbi, A. J., "An intuitive justification and a simplified implementation of the MAP decoder for convolutional codes", *IEEE J.S.A.C*, vol. 16, pp. 260-264, Feb. 1998.

[Zha96] Zhang, L., "On the MAP algorithm, iterative decoding and turbo codes," in *DSP for Communications Systems*, Forth inteernational symposium, Perth, Sept. 1996.

Annexe A

Pages Web. Utiles

Voici des sites sur l'internet où des articles sur les Codes Turbo sont diponibles, on pourra trouver aussi quelques noms de companies qui fabriquent des CodDecs turbo :

- Virginia Tech, Virginia Polytechnic Institute and State University :

http ://www.ee.virginia.edu/CSL/turbo_codes/

- JPL Turbo Code HomePage :

http ://www331.jpl.nasa.gov/public/JPLtcodes.html

- TDA journal (TDA est une magazine trimestrielle affiliée à la NASA) :

http ://tda.jpl.nasa.gov/progress_report/index.html

- University of South Australia Turbo Code Homepage :

http ://charli.levels.unisa.edu.au/~steven/turbo/

Annexe B

Calcul du rapport z

Calculons d'abord $E\{x_k^2\}$ ensuite $E\{|x_k|\}$

$$\begin{aligned} E\{x_k^2\} &= E\{(\pm\sqrt{E_s} + p_k)^2\} \\ &= E_s + E\{p_k^2\} \pm 2\sqrt{E_s}\underbrace{E\{p_k\}}_{0} \\ &= 1 + \sigma^2 \end{aligned}$$

Remarque : on suppose que $E_s = 1$;

$$
\begin{aligned}
E\{|x_k|\} &= \frac{1}{2}\lfloor E\{|x_k|\}\rfloor_{X_k=1} + \frac{1}{2}\lfloor E\{|x_k|\}\rfloor_{X_k=-1} \\
&= \frac{1}{2}\int_{-\infty}^{+\infty} |x_k| \frac{1}{\sqrt{2\pi}\sigma} \exp\{\frac{-(x_k-1)^2}{2\sigma^2}\} dx_k \\
&\quad + \frac{1}{2}\int_{-\infty}^{+\infty} |x_k| \frac{1}{\sqrt{2\pi}\sigma} \exp\{\frac{-(x_k+1)^2}{2\sigma^2}\} dx_k \\
&= \frac{1}{2} \times 2 \int_{0}^{+\infty} x_k \frac{1}{\sqrt{2\pi}\sigma} \exp\{\frac{-(x_k-1)^2}{2\sigma^2}\} dx_k \\
&\quad + \frac{1}{2} \times 2 \int_{0}^{+\infty} x_k \frac{1}{\sqrt{2\pi}\sigma} \exp\{\frac{-(x_k+1)^2}{2\sigma^2}\} dx
\end{aligned}
$$

Calculons l'expression

$$
I_m = \int_{0}^{+\infty} x \frac{1}{\sqrt{2\pi}\sigma} \exp\{\frac{-(x-m)^2}{2\sigma^2}\} dx
$$

posons $u = x - m$ donc $du = dx$ et

$$
\begin{aligned}
I_m &= \int_{0}^{+\infty} (u+m) \frac{1}{\sqrt{2\pi}\sigma} \exp\{\frac{-u^2}{2\sigma^2}\} du \\
&= \underbrace{\frac{1}{\sqrt{2\pi}\sigma} \int_{-m}^{+\infty} u \exp\{\frac{-u^2}{2\sigma^2}\} du}_{I_{m_1}} + \underbrace{m \int_{-m}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma} \exp\{\frac{-u^2}{2\sigma^2}\} du}_{I_{m_2}}
\end{aligned}
$$

$$
\begin{aligned}
I_{m_1} &= \frac{1}{\sqrt{2\pi}\sigma}\int_{-m}^{+\infty} u\exp\{\frac{-u^2}{2\sigma^2}\}du \underbrace{=}_{v=-\frac{u^2}{2\sigma^2}} -\frac{\sigma}{\sqrt{2\pi}}\int_{-\frac{m^2}{2\sigma^2}}^{-\infty}\exp\{v\}dv \\
&= \frac{\sigma}{\sqrt{2\pi}}\exp\{-\frac{m^2}{2\sigma^2}\}
\end{aligned}
$$

$$
\begin{aligned}
I_{m_2} &= m\int_{-m}^{+\infty}\frac{1}{\sqrt{2\pi}\sigma}\exp\{\frac{-u^2}{2\sigma^2}\}du \underbrace{=}_{v=\frac{u}{\sqrt{2}\sigma}} m\int_{-\frac{m}{\sqrt{2}\sigma}}^{+\infty}\frac{1}{\sqrt{2\pi}\sigma}\exp\{-v^2\}\sqrt{2}\sigma dv \\
&= \frac{m}{2}\times\frac{2}{\sqrt{\pi}}\int_{-\frac{m}{\sqrt{2}\sigma}}^{+\infty}\exp\{-v^2\}dv \\
&= \frac{m}{2}\,\mathrm{erf}\,c\{-\frac{m}{\sqrt{2}\sigma}\}
\end{aligned}
$$

Finalement

$$
\begin{aligned}
E\{|x_k|\} &= I_{-1} + I_{+1} \\
&= \frac{\sigma}{\sqrt{2\pi}}\exp\{-\frac{1}{2\sigma^2}\} - \frac{1}{2}\,\mathrm{erf}\,c\{-\frac{1}{\sqrt{2}\sigma}\} \\
&\quad + \frac{\sigma}{\sqrt{2\pi}}\exp\{-\frac{1}{2\sigma^2}\} + \frac{1}{2}\,\mathrm{erf}\,c\{\frac{1}{\sqrt{2}\sigma}\} \\
&= \sigma\sqrt{\frac{2}{\pi}}\exp\{-\frac{1}{2\sigma^2}\} + \frac{1}{2}(2 - 2\,\mathrm{erf}\,c\{\frac{1}{\sqrt{2}\sigma}\}) \\
&= \sigma\sqrt{\frac{2}{\pi}}\exp\{-\frac{1}{2\sigma^2}\} + \mathrm{erf}\{\sqrt{\frac{1}{2\sigma^2}}\}
\end{aligned}
$$

et $z = E\{x_k^2\}/E\{|x_k|\}^2$ est bien égale à

$$
\frac{(1+\sigma^2)}{(\sigma\sqrt{\frac{2}{\pi}}\exp(-\frac{1}{2\sigma^2}) + \mathrm{erf}(\sqrt{\frac{1}{2\sigma^2}}))^2}
$$

Annexe C

Programme en Matlab

```
##########################################
############ Main Program ################
##########################################
clear;
clf;
format long;
global g;
global M;
global bg;
global Vg;
global Y;
global Sb;
global Sf;
global L_data;
global SystInf;
global ParityEncoder1List;
global ParityEncoder2List;
global alpha;
```

```
global Pbsum;
global NumBlocErr;
%%%%%%%% %%%%%%%%%
%%%%%%%% Input %%%%%%
%%%%%%%%%%%%%%%%%%
% k and l are the dimension of the interleaver
k=20;%number of matrix raw
l=20;%number of matrix column
m=0;%Tail length
EbN0dB=3;
bg=1;
Vg=3;
CodingRate=bg/Vg;%Coding Rate
NumIteration=3;%Number of iteration
Bloc=1;%Bloc Interleave
Rand=2;%Variable assigned to the random interleave
g=[1 1 1;1 0 1];
BlocNumMax=5;
Pbsum=0;
NumErr=0;
NumBlocErr=0;
TotNumErr=0;
%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%    End of Input    %%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%
tic
[n,K]=size(g);
M=2^(K-1);
```

```
N=k*l;%List length
[alpha]=Interleaver(Bloc,k,l);
%%%%%%%%%%% %%%%%%%%%%%%
%%%%%%%%%%% Main %%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%
for EbN0Ind=1 :length(EbN0dB);%Signal over Noise ratio
for BlocNum=1 :BlocNumMax
[L_data]=generate(N);%L is the generated list
[EL]=TurboEncoder(L_data);%EL is the encoded list
[ML]=Modulate(EL);%ML is the modulated list
[sigma]=NoiseVariance(EbN0dB,EbN0Ind,CodingRate);
[NL Enveloppe]=AddNoise(ML,sigma,'Rayleigh');%NL is the noisy list
Lc=(2*Enveloppe)/sigma^2;
NL=-Lc.*NL;
[SystInf ParityEncoder1List ParityEncoder2List]=Demultiplex(NL);
[NumErr]=Decoding(NumIteration,alpha);
TotNumErr=TotNumErr+NumErr
[Pb]=Statistics(NumErr,BlocNum,NumIteration);%under Constru
end
%[Pe]=Total_Statistics(Pb,NumBlocErr);%under Constru
BlocNum
Pb
NumBlocErr
end
toc
%Plot_Results(Pb,Pe,EbN0dB);%under Construc
%subplot(2,2,1);
semilogy(EbN0dB,Pb,'*r');
```

```
xlabel ('Eb/N0');
ylabel ('Pb');
title('BPSK');
%subplot(2,1,2);
%semilogy(EbN0dB,abs(Diff));
%xlabel ('');
%ylabel ('');
##############End of Main.m ##################


#########################################################
################ Interleaver.m  ###############
#########################################################
#########################################################
% Calculate the function alpha corresponding to a specific type of interleaving
%29March99
%k and l are the dimension of the interleaver.
function [alpha]=Interleaver(type,k,l)
if type==1
alpha=Bloc_Interleaver(k,l);
elseif type==2
alpha=Rand_Interleaver(k,l);
end
%alpha contains the postion the interleaved bits
%alpha(i)=j means that the ieme bit a list L was
%interleaved to the jieme postion
%28March99
function [alpha]=Bloc_Interleaver(k,l)
%write in an increasing order into the matrix row by row
```

```
for i=1 :k
for j=1 :l
M(i,j)=j+(i-1)*l;
end
end
%read out of matrix column by colunm from the first to the last
for j=1 :l
for i=1 :k
alpha(i+(j-1)*k)=M(i,j);
end
end
%alpha contains the postion a randomly interleaved bits
%alpha(i)=j means that the ieme bit a list L was
%interleaved to the jieme postion
%28March99
function [alpha]=Rand_Interleaver(k,l)
for i=1 :k*l
test=1;
while (test == 1)
alpha(i)=round(k*l*rand(1,1));%random variable between 0 and k*l
if(alpha(i)<=k*l)&(alpha(i)>0)
test=0;
for j=1 :(i-1)
if (alpha(i)==alpha(j)) %test if the current alpha(i) have been used
test=1;
end
end
end
```

```
end
end
################## End of Interleaver ##########

##########################################
################ generate.m ################
##########################################
##########################################
%generate a List of N randons varaibles 0 or 1
function [List]=generate(N)
List=round(rand(1,N));
###############End of generate.m ############

##########################################
################## TurboEncoder.m ##########
##########################################
##########################################
function [TEL]=TurboEncoder(L_data)
% Copyright 1997 Yufei Wu
% modified April 1999 Afif
% uses interleaver map 'alpha'
% if puncture = -1, unpunctured, produces a rate 1/3 output of fixed length
% if puncture = 1, punctured, produces a rate 1/2 output
% multiplexer chooses odd check bits from RSC1
% and even check bits from RSC2
% determine the constraint length (K), memory (m)
% and number of information bits plus tail bits.
```

```
global g;
global bg;
global Vg;
global Y;
global Sb;
global Sf;
global alpha;
[n,K] = size(g);
L_total = length(L_data);
% generate the codeword corresponding to the 1st RSC coder
%
output1 = rsc_encode(L_data);
% make a matrix with first row corresponing to info sequence
% second row corresponsing to RSC #1's check bits.
% third row corresponsing to RSC #2's check bits.
y(1, :) = L_data;
y(2, :) = output1;
% interleave input to second encoder
%%%
for i = 1 :L_total
input1(1,i) = y(1,alpha(i));
end
output2 = rsc_encode(input1);
%%%temp = interleave(output2, L_total, 2);
y(3, :) = output2;
[Output Sf Sb]=state_diagramme(g);
Y( :,1)=Output( :,2);
Y( :,2)=Output( :,4);
```

```
[TEL]=multiplex(y,n);
%%%%%%%%%%%%%end of Turbo
%%%%%%%%%%%%%%%%%%%%%
function [Parity]= rsc_encode(x)
% encodes a block of data x (0/1)with a recursive systematic
% convolutional code with generator vectors in g, and
% returns the output in y (0/1).
% determine the constraint length (K), memory (m), and rate (1/n)
% and number of information bits.
global g;
[n,K] = size(g);
m = K - 1;
L_total = length(x);
% initialize the state vector
state = zeros(1,m);
% generate the codeword
for i = 1 :L_total
d_k = x(1,i);
feedback = g(1, :)&[d_k state];
a_k = rem( sum( feedback ), 2 );
[output_bits, state] = encode_bit(g, a_k, state);
% since systematic, first output is input bit
%output_bits(1,1) = d_k;
%y(n*(i-1)+1 :n*i) = output_bits
Parity(i)=output_bits(1,2);
end
%%%%%%%%%%% end of rsc_encode.m %%%%%%%%
% paralell to serial multiplex to get output vector
```

```
% punctured : rate increase from 1/3 to 1/2;
% unpunctured, rate = 1/3;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [TELM] = multiplex(y,n)
global bg;
global Vg;
if bg==1 & Vg==3 % unpunctured
for i = 1 :length(y)
for j = 1 :3
TELM(1,(3)*(i-1)+j) = y(j,i);
end
end
else % punctured into rate 1/2
for i=1 :length(y)
TELM(1,n*(i-1)+1) = y(1,i);
if rem(i,2)
% odd check bits from RSC1
TELM(1,n*i) = y(2,i);
else
% even check bits from RSC2
TELM(1,n*i) = y(3,i);
end
end
end
%%%%%%%% End of multiplex.m %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [output,transition ,inv_state] = state_diagramme(g)
% copyright Yufei Wu
```

```
% set up the trellis given code generator g
% g given in binary form. e.g. g = [ 1 1 1; 1 0 1 ];
% output(i,1) : trellis output when input = 0, state = i;
% output(i,2) : trellis output when input = 1, state = i;
% transition(i,1) : next state when input = 0, state = i;
% transition(i,2) : next state when input = 1, state = i;
% inv_state(i, :) : two states that can come to state i;
[n,K] = size(g);
m = K - 1;
max_state = 2^m;
% set up output and transition matrices for systematic code
for state=1 :max_state
state_vector = bin_state( state-1, m );
% hypothesis : receive a 0
d_k = 0;
feedback = g(1, :)&[0 state_vector];
a_k = rem( sum( feedback ), 2 );
[out_0, state_0] = encode_bit(g, a_k, state_vector);
out_0(1) = 0;
% hypothesis : receive a 1
d_k = 1;
feedback = g(1, :)&[1 state_vector];
a_k = rem( sum( feedback ), 2 );
[out_1, state_1] = encode_bit(g, a_k, state_vector);
out_1(1) = 1;
output(state, :) = [out_0 out_1];
transition(state, :) = [(int_state(state_0)+1) (int_state(state_1)+1)];
end
```

```
% find out which two states can come to present state
inv_state = zeros(max_state,2);
for state=1 :max_state
for bit=0 :1
if inv_state(transition(state,bit+1),bit+1)==0
inv_state(transition(state,bit+1),bit+1)=state;
else
inv_state(transition(state,bit+1),2-bit)=state;
end
end
end
%%%%%%%%% End of state_diagramme.m %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function bin_state = bin_state( int_state, m )
% Copyright 1997 Yufei Wu
% for academic use only
% converts an vector of integer into a matrix; the i-th row is the binary form
% of m bits for the i-th integer
for j = 1 :length( int_state )
for i = m :-1 :1
state(j,m-i+1) = fix( int_state(j)/ (2^(i-1)) );
int_state(j) = int_state(j) - state(j,m-i+1)*2^(i-1);
end
end
bin_state = state;
%%%%%%%%%%%% End of bin_state %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function int_state = int_state( state )
```

```
% Copyright 1996 Matthew C. Valenti
% for academic use only
% converts a row vector of m bits into a integer (base 10)
[dummy, m] = size( state );
for i = 1 :m
vect(i) = 2^(m-i);
end
int_state = state*vect';
%%%%%%%%%%%% End of int_state %%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [output, state] = encode_bit(g, input, state)
% Copyright 1996 Matthew C. Valenti
% for academic use only
% This function takes as an input a single bit to be encoded,
% as well as the coeficients of the generator polynomials and
% the current state vector.
% It returns as output n encoded data bits, where 1/n is the
% code rate.
% the rate is 1/n
% k is the constraint length
% m is the amount of memory
[n,k] = size(g);
m = k-1;
% determine the next output bit
for i=1 :n
output(i) = g(i,1)*input;
for j = 2 :k
output(i) = xor(output(i),g(i,j)*state(j-1));
```

```
end;
end
state = [input, state(1 :m-1)];
%%%%%%%%% End of encode_bit %%%%%%%%%
############ End of TurboEncoder.m ##########
#########################################################
#### Modulate.m, NoiseVariance.m & Addnoise.m ###
#########################################################
#########################################################
function [ML]= Modulate(L);
ML=2*L-1;%modulation BPSK
%Calculate the standard deviation of AWGN noise
function [sigma]=NoiseVariance(EbN0dB,EbN0Ind,CodingRate)
EbSN0dB=EbN0dB(EbN0Ind);%EbSN0dB is the current Eb over N0 in dB
EbSN0=10.^(EbSN0dB./10);%Conversion from log scale to the linear scale
variance=1./(2*CodingRate*EbSN0);%we assume that Es is equal to 1
sigma =sqrt(variance);
function [NoisyList,Enveloppe] = AddNoise(List,sigma,Type)
sz=size(List);
if strcmp(Type,'Rayleigh')==1
b=randn(sz);
c=randn(sz);
Enveloppe=sqrt((b.^2+c.^2)*0.5);
else%gaussien noise
Enveloppe=ones(sz);
end
NoisyList=Enveloppe.*List+sigma*randn(sz);
####End of Modulate.m, NoiseVariance.m & Addnoise.m####
```

```
##########################################
############### Demultiplex.m ##############
##########################################
##########################################
function [sys,p1,p2] = demultiplex(r);
% copywright 1997, Yufei Wu
% at receiver, serial to paralle demultiplex to get the code word of each
% encoder
% dimension : number of encoder/decoder;
% alpha : interleaver map
% puncture = 1 : rate increase from 1/(1+dimension) to 1/2;
% puncture = -1; unpunctured, rate = 1/(1+dimension);
global bg;
global Vg;
if bg==1 & Vg==3
L_total = length(r)/3;
for i = 1 :L_total
sys(i) = r(3*(i-1)+1);
for j = 1 :2
parity(j,i) = r(3*(i-1)+1+j);
end
end
else
L_total = length(r)/2;
for i = 1 :L_total
sys(i) = r(2*(i-1)+1);
for j = 1 :2
parity(j,i) = 0;
```

```
end
if rem(i,2)>0
parity(rem(i,2),i) = r(2*i);
else
parity(2,i) = r(2*i);
end
end
end
p1=parity(1, :);
p2=parity(2, :);
##############End of Demultiplex.m ##############
##########################################################
################## Decoding functions ##########
##########################################################
##########################################################
%%%%%%%%%%% Decoding.m %%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%L the received list from the channel
%MumIteration is the number of iteration to do
%March99
function [NumErr]= Decoding(NumIteration,alpha)
global g;
global Y;
global Sb;
global Sf;
global L_data;
global SystInf;
global ParityEncoder1List;
```

```
global ParityEncoder2List;
%assign the apriori information of the
%first decoderLapriori1 to zero. This is done only ones.
%N is the length of the data bloc at the source
N=length(SystInf);
Lapriori1=zeros(1,N);
for Iter=1 :NumIteration
L_inDec1=Lapriori1+SystInf;
tic%%%%%%%
L_decoder1=MAP(L_inDec1,ParityEncoder1List);
toc%%%%%%%%%%%
%L_decoder1=L_in1+L_extrinsec1
clear L_inDec1
%
L_extrinc1PlusSysInf=L_decoder1-Lapriori1;
L_inDec2=Interleave(L_extrinc1PlusSysInf,alpha);
%
tic
L_decoder2=MAP(L_inDec2,ParityEncoder2List);
toc
%
Lextrinsec2=L_decoder2-L_inDec2;
Lapriori1=DeInterleave(Lextrinsec2,alpha);
%
DL_dec_Iter=(-sign(L_decoder2( :))+1)/2;
%DL_dec_Iter contains the Decoded list
%for the current iteration
De_DL_dec_Iter=DeInterleave(DL_dec_Iter,alpha);
```

```
%De_DL_dec_Iter is the deinterleaved decoded list
%which should be L_data
errList=find(De_DL_dec_Iter ~= L_data);
NumErr(Iter)=length(errList);
%errList contain the position of the bits in error for the list L_data
end
%DL= De_DL_dec_Iter;
%DL is the decoded list after MumIteration iterations
%%%%%%%% End of Decoding.m %%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%% MAP.m %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%
function L_decoder=MAP(L_inDec,ParityEncoderList)
global Y;
global Sb;
global Sf;
global M;
N=length(L_inDec);
Huge=1e6;
%1-Calcul des metriques de branches (delta)
D=zeros(4,N);%(i,Yk(i,m))=(0,0)
D(2, :)=ParityEncoderList;%(0,1)
D(3, :)=L_inDec;%(1,0)
D(4, :)=L_inDec+ParityEncoderList;%(1,1)
%2-Initialisation et calcul de la recursion en arriere(Beta)
Beta=0*ones(M,N,2);%-log(0)=Huge;
i0=1;
i1=2;
```

```
j0=1;
j1=2;
S00=1;
k1=1;
Beta( :,N, :)=log(M);
%Beta(Sb(S00,i0),N,i0)=0;%-log(1)=0
%Beta(Sb(S00,i1),N,i1)=0;%Beta_i_m_k
for k=(N-1) :-1 :k1
%Beta(m,k)=E{D[(R_(k+1),Sf_jm),k+1]+Beta(Sf_jm),k+1)}(j=0,1)
for m=1 :M
Beta(m,k,i0)=E_function((D(1+Y(Sf(m,i0),j0),k+1)+Beta(Sf(m,i0),k+1,j0)),
                        (D(3+Y(Sf(m,i0),j1),k+1)+Beta(Sf(m,i0),k+1,j1)));
Beta(m,k,i1)=E_function((D(1+Y(Sf(m,i1),j0),k+1)+Beta(Sf(m,i1),k+1,j0)),
                        (D(3+Y(Sf(m,i1),j1),k+1)+Beta(Sf(m,i1),k+1,j1)));
%la premiere (resp. deuxieme)ligne correspond a i=0(resp. i=1),
%la presence de(1+Y()) est a la necessite que l'indice
%commence a 1 et non plus a 0
end
Beta( :,k, :)=(Beta( :,k, :)-min(min(Beta( :,k, :))));%+min(min(Beta( :,k, :))))/2);
end
%Normalisation%
%Max=max(max(max(Beta)));
%Min=min(min(min(Beta)));
%Beta=Beta-(Max+Min)/2;
%3-Initailisation et calcul de la recursion en avant(Alpha)
Alpha=Huge*ones(M,N,2);
Alpha(S00,k1,i0)=D(1+Y(S00,i0),k1);
Alpha(S00,k1,i1)=D(3+Y(S00,i1),k1);
```

```
for k=2 :1 :N
for m=1 :M
Alpha(m,k,i0)=(D(1+Y(m,i0),k)+
                E_function((Alpha(Sb(m,j0),k-1,j0)),(Alpha(Sb(m,j1),k-1,j1))) ;
Alpha(m,k,i1)=(D(3+Y(m,i1),k)+Alpha(m,k,i0)-D(1+Y(m,i0),k)) ;
end
Alpha( :,k, :)=(Alpha( :,k, :)-min(min(Alpha( :,k, :))) ;%+min(min(Alpha( :,k, :))))/2) ;
end
%Normalisation%
% Max=max(max(max(Alpha))) ;
%Min=min(min(min(Alpha))) ;
%Alpha=Alpha-(Max+Min)/2 ;
%4-LLR
L_decoder=E_functions(Alpha+Beta,i1)-E_functions(Alpha+Beta,i0) ;
clear D ;
clear Alpha ;
clear Beta ;
%%%%%%%%%%%%%%%%%%%%%
%E_function(A,B)=log(exp(A)+exp(B))
%%%%%%%%%%%%%%%%%%%%%
function [Res]=E_function(A,B)
ab=abs(A-B) ;
if ab>16
Res=min(A,B) ;
else
Res=min(A,B)-log(1+exp(-ab)) ;
end
%%% End of E_function %%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%
%%%E_functions
%%%%%%%%%%%%%%%%%%%%%
function [Res]=E_functions(V,i)
Res=-log(sum(exp(-V( :, :,i)),1));
%summation of all columns(each colunme represents the all state of the variable
V
%in order to calculate the E_function for every k
%%%%%% End of E_functions %%%%%%

%%%%%%% End of MAP.m %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%
% Interleave the L_in list by using an interleaver of type alpha
%29March99
function [L_interleaved]=Interleave(L_in,alpha)
for i=1 :length(L_in)
L_interleaved(i)=L_in(alpha(i));
end
%%%%% end of interleave.m %%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%
% DeInterleave the L_in list by using an interleaver of type alpha
%29March99
function [L_Deinterleaved]=DeInterleave(L_in,alpha)
for i=1 :length(alpha)
L_Deinterleaved(alpha(i))=L_in(i);
end
%%%%%%End of DeInterleave.m%%%%%%%%
########## End of Decoding functions#########
%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function [Pb]=Statistics(NumErr,BlocNum,NumIteration)
global Pbsum;
global L_data;
global NumBlocErr;
PbBloc=NumErr/length(L_data);
if NumErr(NumIteration)~=0
NumBlocErr=(NumBlocErr+1)
end
%Pb is the probability of bit error for the list L_data
Pbsum=PbBloc+Pbsum;
Pb=Pbsum/BlocNum;
%%%%%% End of Statistics.m %%%%%%%%
```