Titre: Title:	Sur le décodage itératif des codes Turbo
Auteur: Author:	Naoufel Bouzouita
Date:	1997
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Bouzouita, N. (1997). Sur le décodage itératif des codes Turbo [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/6706/

UNIVERSITÉ DE MONTRÉAL

SUR LE DÉCODAGE ITÉRATIF DES CODES TURBO

NAOUFEL BOUZOUITA

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)
JUIN 1997

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

SUR LE DÉCODAGE ITÉRATIF DES CODES TURBO

présenté par: BOUZOUITA Naoufel

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

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

M. GHANNOUCHI Fadhel M., Ph.D., président

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

M. GAGNON François, Ph.D., membre

REMERCIEMENTS

Le travail présenté dans ce mémoire n'aurait pu être mené à terme sans l'aide et le soutien de nombreuses personnes, que je me dois de remercier.

Je tiens pour commencer à exprimer toute ma gratitude à mon directeur de recherche, Dr. David Haccoun, pour le soutien constant qu'il m'a apporté tout au long de ce travail. Je lui suis très reconnaissant pour tous ses conseils et directives qui m'ont permis de franchir divers obstacles, pour enfin aboutir au contenu de ce mémoire. Je le remercie encore pour m'avoir soutenu financièrement durant toute la période de ma recherche.

Mes remerciement vont également à l'Agence Canadienne de Développement International et au Ministère de l'Éducation Supérieure de Tunisie pour la bourse qu'ils m'ont octroyée.

J'aimerais aussi remercier tous mes collègues de travail, qui ont su créer une ambiance amicale au sein de notre laboratoire, propice à un travail efficace et productif. Merci à Véronique, Violeta, Pierre, Jean-Émile, Afif, Martin et Éric. Je remercie plus particulièrement Melita Adamovic, Zohair Gherbi et Christian Cardinal pour les discussions fructueuses que nous avons eues.

Je tiens finalement à remercier tous mes proches et tous mes amis pour leur soutien pendant ces deux années passées à Montréal. Je pense en particulier à mes parents, à Adel, Sandra, Zied et Sarra.

RÉSUMÉ

Ce mémoire porte sur le décodage (turbo) itératif appliqué à la concaténation parallèle de deux codeurs convolutionnels récursifs et systématiques à travers un entrelaceur. Les codes turbo constituent à ce jour la technique la plus puissante pour la détection et la correction d'erreurs: une probabilité d'erreur par bit de 10^{-5} a pu être obtenue à une valeur de E_b/N_0 à 0,7dB de la valeur à la capacité du canal, pour un taux de codage R=1/2.

Afin d'obtenir les meilleures performances du décodeur turbo, l'algorithme de décodage MAP (*Maximum A Posteriori*) est utilisé pour décoder chacun des codes élémentaires constituant le codeur turbo.

Nous montrons que la génération d'une information de fiabilité par l'algorithme MAP, et l'utilisation d'un entrelaceur pseudo-aléatoire pour transférer cette information d'un décodeur à l'autre, sont à l'origine de l'amélioration progressive des performances d'erreur, d'itération en itération.

L'analyse des performances des codes turbo pour des blocs de faibles à moyennes dimensions (inférieures à 1000 bits par bloc), montre l'importance du choix des codes élémentaires et de l'entrelaceur pour l'obtention des meilleurs résultats. Nous démontrons en particulier la nécessité de réinitialiser les deux codeurs élémentaires lorsque les blocs considérés sont de faibles dimensions.

La comparaison des performances des codes convolutionnels à celles des codes turbo, montre que ces derniers permettent encore d'obtenir de meilleurs résultats lorsque la dimension des blocs est comprise entre 100 et 200 bits, à condition cependant de choisir convenablement les paramètres des codes élémentaires et de l'entrelaceur.

Pour terminer, nous proposons de nouveaux codes de taux variables résultant de la perforation des codes turbo. Les performances de codes turbo perforés de taux élevés sont comparées à celles des codes convolutionnels perforés. Bien que les codes turbo surclassent là encore les codes convolutionnels, il s'avère que la dimension minimale recommandée pour les blocs croît avec le taux de codage.

ABSTRACT

This thesis investigates iterative (turbo) decoding applied to the parallel concatenation of two recursive systematic convolutional codes through an interleaver. Among the all known error control techniques, turbo coding is currently the most powerful one: it is shown that it is possible to come within 0.7dB of the channel capacity at a rate 1/2 code for a bit error rate of 10^{-5}.

To obtain the best performance from the turbo decoder, the Maximum A Posteriori (MAP) decoding algorithm is used to decode each of the two individual constituent encoders of the turbo encoder.

Generation of a reliability information by the MAP algorithm, and the choice of a pseudo-random interleaver when passing this information from one decoder to the other one, are shown to be the keys to the improvement in performance from one iteration to the next.

Turbo codes are investigated in this thesis for short to medium frame transmission systems (less than 1,000 bits per frame), for which we show, using computer simulations, how the interleaver and the constituent codes contribute to the obtained performance. We prove in particular the importance of terminating the trellises of both constituent codes when short block sizes are considered.

Performance comparisons with convolutional codes show that turbo codes still constitute an advantageous solution when block sizes are between 100 and 200 bits, provided that constituent codes and interleaver parameters are adequately chosen.

Variable-rate turbo codes are examined. High-rate punctured turbo code performances are compared to those of punctured convolutional codes. It is than shown that the minimum block size required to obtain better results with turbo codes increases with the coding rate.

TABLE DES MATIÈRES

Chapitre 3
L'ALGORITHME DE DÉCODAGE

Chapitre 4
DÉCODAGE ITÉRATIF PAR L'ALGORITHME MAP: LE DÉCODEUR TURBO

Chapitre 5

ANALYSE DES PERFORMANCES DES CODES TURBO

Chapitre 6
CODES TURBO DE TAUX VARIABLES

Chapitre 7
CONCLUSIONS ET PERSPERCTIVES DE RECHERCHE

LISTE DES FIGURES

LISTE DES ANNEXES

Chapitre 1
INTRODUCTION

Le développement fulgurant qu'ont connu les télécommunications pendant ces deux dernières décennies a donné naissance à un besoin sans cesse croissant d'assurer un échange de plus en plus fiable d'informations de toutes sortes. Ainsi a-t-on assisté à l'apparition de techniques de plus en plus élaborées, visant l'obtention à la réception, d'une parfaite réplique des données transmises par la source. Le schéma typique d'un système de transmission numérique est illustré par la figure 1.1.

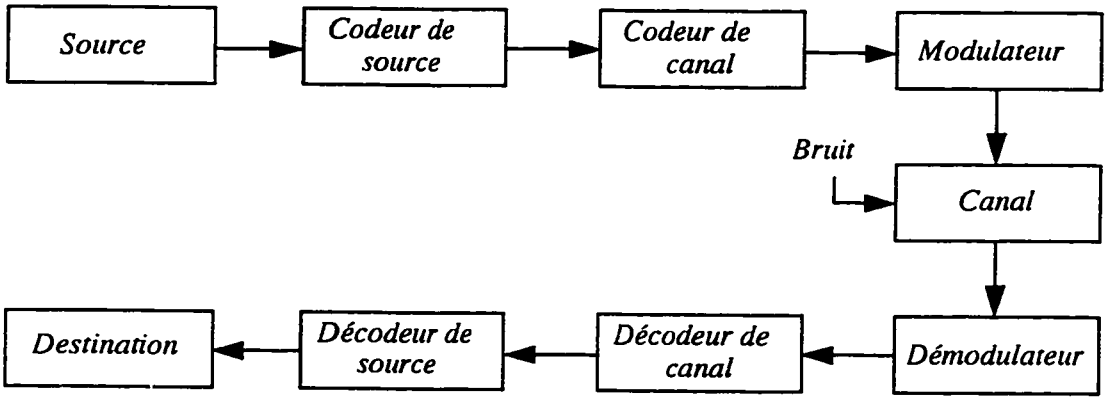


Figure 1.1 Modèle d'un système de communication numérique

La première étape pour la transmission des données, consiste à numériser, au niveau du codeur de source, les données émises par la source, qui peuvent être de nature quelconque. Divers codes peuvent être utilisés à ce niveau; les plus répandus demeurent cependant les codes binaires, dont les digits prennent les valeurs 0 et 1.

Le but de l'utilisation d'un codeur de canal est d'ajouter de la redondance à l'information à transmettre afin de la protéger contre les perturbations introduites dans le milieu constituant le canal; perturbations souvent modélisées par un bruit blanc gaussien

additif, caractérisé par sa densité spectrale de puissance unilatérale N_0. La modulation permet d'adapter les symboles codés au canal physique.

A la réception des signaux bruités, ceux-ci sont d'abord démodulés; un filtre adapté peut par exemple être utilisé à cette fin. Le décodage de canal, deuxième étape dans le traitement de l'information reçue, est basé sur un algorithme dont la fonction est d'utiliser la redondance ajoutée par le codeur de canal pour détecter et corriger les éventuelles erreurs introduites lors de la transmission. Le décodage de canal vise ainsi la minimisation de la probabilité d'erreur. A partir des bits décodés, le décodeur de source reconstitue les données émises par la source avant de les soumettre au destinataire.

Nous nous intéressons dans le cadre de ce mémoire aux codage et décodage de canal, que nous désignerons dans la suite simplement par codage et décodage. L'entrée de notre système est donc constituée par les bits d'information générés par le codeur de source, et à la sortie, une estimation de ces mêmes bits est délivrée au décodeur de source.

Les différents types de codes correcteurs d'erreurs peuvent être classifiés en deux grandes catégories. Dans la première, nous retrouvons les codes en blocs qui, comme leur nom l'indique, requièrent la subdivision au préalable des bits d'information en paquets, ou blocs, dont le codage se fait de façon indépendante. Quant à la deuxième catégorie, elle est constituée par les codes convolutionnels, qui permettent un codage continu de l'information à transmettre.

Les codes convolutionnels figurent parmi les codes les plus utilisés, notamment pour des applications, telles que les systèmes de communications personnelles sans fils, les communications par satellite, etc, où une très faible probabilité d'erreur par bit est requise. Cela a été rendu possible grâce à l'utilisation d'un algorithme de décodage puissant et pratique appelé algorithme de Viterbi. Cet algorithme à maximum de vraisemblance est optimal et permet de minimiser la probabilité d'erreur par séquence. Le

principal inconvénient de ce type de décodeur cependant, est que l'augmentation de la longueur de contrainte pour les codes convolutionnels, résultant en une augmentation du gain de codage, et donc une amélioration des performances d'erreur, entraîne un accroissement exponentiel de la complexité du décodage, au point qu'au delà d'une valeur de la longueur de contrainte, typiquement K=9, l'implémentation peut devenir physiquement impossible.

Ainsi la recherche dans le domaine du codage a-t-elle connu plusieurs tentatives visant la construction de codes puissants, dont le décodage serait structuré de manière à subdiviser le décodage à maximum de vraisemblance en un ensemble d'étapes plus simples de décodage partiel. Les plus importantes de ces tentatives ont conduit à l'apparition des codes produit [14], des codes concaténés [15], des codes multi-niveaux [21], ainsi que d'algorithmes sous-optimaux pour le décodage de codes convolutionnels avec de grandes longueurs de contrainte, tels que le décodage séquentiel [9], le décodage de Viterbi adaptatif [13], et le décodage bidirectionnel [10].

Toutes ces techniques, bien qu'ayant apporté une amélioration des performances avec une augmentation de complexité relativement faible, présentaient toutefois l'inconvénient d'un fonctionnement indépendant des différents modules qui peuvent constituer le décodeur: aucun échange d'information n'est effectué entre les différentes parties.

En 1993, une nouvelle classe de codes correcteurs d'erreur, appelés *codes turbo*, vit le jour [7]. Il s'agit de la concaténation parallèle de deux codeurs récursifs et systématiques à travers un entrelaceur, dont le décodage est effectué de façon itérative par deux décodeurs concaténés en série. Une probabilité d'erreur par bit $P_b=10^{-5}$ était annoncée pour un rapport signal sur bruit E_b/N_0=0,7dB, soit à 0,7dB de la capacité du canal. Ces résultats de simulation ont été obtenus avec la concaténation de deux codeurs à 16 états, un entrelaceur de 65536 bits, et avec un décodage en 18 itérations utilisant l'algorithme MAP (*Maximum A Posteriori*).

Les principaux inconvénients que l'on reprochait aux codes turbo, portaient d'une part sur la complexité élevée de l'algorithme de décodage utilisé, et d'autre part sur le délai, proportionnel au nombre d'itérations et à la longueur de l'entrelaceur, engendré par le décodage itératif. Par ailleurs, l'étude analytique de la concaténation parallèle de codes, surtout lorsqu'un entrelaceur est utilisé, s'avérant très complexe, la majorité des analyses de performance des codes turbo étaient basées sur des résultats de simulation, ce qui entravait l'interprétation de certains comportements caractéristiques de cette classe de codes.

Nous présentons dans ce mémoire une étude du principe de fonctionnement des codes turbo, ainsi qu'une analyse de leurs performances en fonction des différentes composantes du système, pour de faibles longueurs de l'entrelaceur (inférieures à 1000 bits). Notre analyse est basée essentiellement sur des résultats de simulation, ainsi que sur certains résultats théoriques récemment publiés [4,5,29].

Toutes nos simulations ont été effectuées sur des stations de travail SPARC 10 et SPARC 20. La simulation de l'algorithme MAP a été effectuée sous le logiciel SPW (*Signal Processing Worksystem*) [37], tandis que tous les résultats relatifs au décodage itératif ont été obtenus avec un simulateur développé en langage C.

Ce mémoire est structuré comme suit:

Le chapitre 2 introduit le principe du codage convolutionnel et décrit en particulier les codes convolutionnels récursifs et systématiques. La présentation de la structure du système et du principe de fonctionnement de ses différentes composantes est également faite dans ce chapitre.

Le chapitre 3 est consacré à l'algorithme de décodage MAP. Nous y présentons la dérivation détaillée de l'algorithme, ainsi que les simplifications qui peuvent y être apportées. Quelques résultats de simulation sont donnés à la fin du chapitre, et une brève comparaison à l'algorithme SOVA est effectuée.

Au chapitre 4, nous analysons le fonctionnement du décodeur turbo. Nous exposons alors les modifications à apporter à l'algorithme MAP afin de l'adapter au décodage itératif, et démontrons l'importance du rôle que jouent le type et la longueur de l'entrelaceur dans l'amélioration progressive des performances d'erreur.

L'analyse des performances des codes turbo est effectuée au chapitre 5, où nous démontrons l'importance du choix des codeurs élémentaires et de l'entrelaceur en vue de l'optimisation des performances obtenues pour de faibles longueurs de blocs. Nous fournissons également une interprétation de certains comportements caractéristiques des codes turbo en fonction des éléments qui les constituent. Le rapport performances -complexité des codes turbo, utilisant de faibles dimensions de blocs, est examiné en comparaison à celui des codes convolutionnels. Nous fournissions à ce propos une liste d'arguments qui contredisent la thèse d'un article récemment publié [26], et selon laquelle les codes turbo donneraint de moins bonnes performances que les codes convolutionnels dès que la taille des blocs est inférieure à 200 bits.

Au chapitre 6, nous examinons les performances des codes turbo de taux de codage élevés, résultant de l'application de la technique de perforation. Les résultats obtenus sont comparés à ceux des codes convolutionnels perforés de taux de codage équivalents. Nous montrons alors que là encore, les codes turbo surclassent les codes convolutionnels, à condition de respecter certaines limites de fonctionnement, portant notamment sur la longueur des blocs utilisés.

La conclusion de ce mémoire et un nombre de perspectives de recherche sont présentés au chapitre 7.

Cette recherche a permis d'apporter les contributions suivantes:

1. Nous démontrons, par le biais de simulations, l'importance du rôle de l'entrelaceur dans la décorrélation des entrées des décodeurs MAP constituant le décodeur turbo. Un critère pour le choix de la longueur et du type de l'entrelaceur, en

fonction de la longueur de contrainte des codes élémentaires, a pu ainsi être suggéré.

2. L'importance de la réinitialisation des deux codeurs élémentaires est illustrée, en particulier si les blocs utilisés sont de faibles dimensions.

3. Un ensemble de recommandations est donné pour le choix des codes élémentaires et de l'entrelaceur, afin d'obtenir les meilleures performances avec les codes turbo utilisant de faibles longueurs de blocs.

4. L'usage de la perforation pour augmenter le taux de codage, est bien connu pour les codes convolutionnels. Nous avons appliqué cette technique aux codes turbo, aboutissant à des codes de taux variables, dont les performances d'erreur sont encore meilleures que celles des codes convolutionnels. Nous montrons cependant que la longueur des blocs est dans ce cas soumise à une valeur minimale qui dépend du taux de codage considéré.

Chapitre 2
STRUCTURE DU SYSTÈME

Nous présentons dans ce chapitre les différents éléments qui constituent le système utilisé tout au long du présent mémoire, notamment le codeur, le canal et le décodeur turbo ainsi que le principe de fonctionnement de chacune de ces composantes. Par ailleurs, les codes convolutionnels, et en particulier les codes récursifs et systématiques, étant à la base des codes turbo, nous commençons par rappeler les principes de base du codage convolutionnel, et décrivons la procédure de construction des codes convolutionnel récursifs et systématiques.

2.1 Principes de codage convolutionnel

Les codes convolutionnels constituent une famille de codes correcteurs d'erreurs, qui génèrent pour chaque *bit d'information* à transmettre, un ensemble de *symboles codés*, appelés aussi *symboles de parité*. Un codeur convolutionnel est un système de connexions que l'on peut caractériser par trois paramètres:

- sa longueur de contrainte, K;
- son taux de codage, R;
- et ses générateurs $G_1, ..., G_V$.

La figure 2.1 représente l'exemple simple d'un codeur convolutionnel de taux R=1/2 et de longueur de contrainte K=3. Les bits d'information d_k alimentent, un bit à la fois, un registre à décalage dont la longueur est égale è la longueur de contrainte K du code. On définit également la mémoire du code par $\mu = K$-1.

Les symboles de parité générés par le codeur sont obtenus par additions modulo 2, effectuées par V additionneurs. La connexion d'un additionneur i (i=1, 2, ..., V) à une

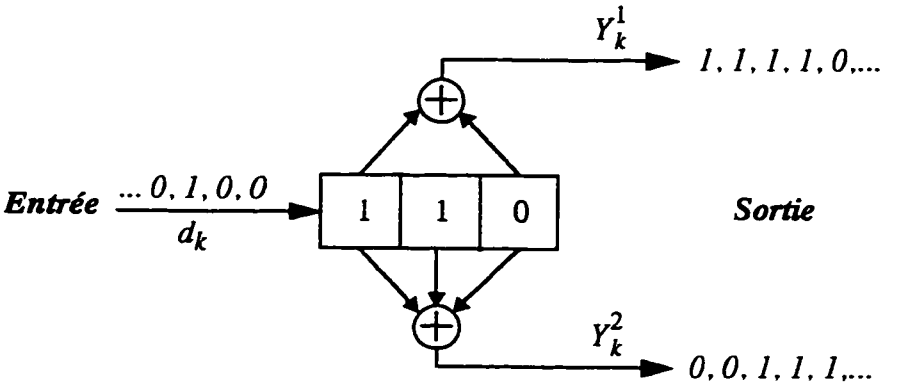


Figure 2.1 Codeur convolutionnel de taux R=1/2
avec K=3, G_1=5, G_2=7.

cellule j (j=0, 1, ..., K-1), est définie par le paramètre binaire g_{ij}. Celui-ci est égal à 1 s'il y a connexion, et à 0 sinon. On définit ainsi pour chaque additionneur, un vecteur générateur $G_i=(g_{i,0}, ..., g_{i,K-1})$; i=1, ..., V, généralement exprimé dans la base octale. Ainsi, dans le cas du code de la figure 2.1, V=2, $G_1=(101)_2=5_8$ et $G_2=(111)_2=7_8$.

Le taux de codage $R=b/V$ est égal au rapport du nombre b de bits à l'entrée du codeur, par le nombre V de symboles codés qui leur sont associés. Lorsque le codeur est à entrée unique, le taux de codage se ramène donc à l'inverse du nombre d'additionneurs: $R=1/V$. Pour chaque bit d'information, V symboles de parité sont générés.

Soit d_k le bit d'information à l'entrée du codeur à l'instant k. Les symboles codés Y_k^i (i=1, ..., V) peuvent alors s'écrire sous la forme

$$Y_k^i = \sum_{j=0}^{K-1} g_{ij} d_{K-j} \pmod 2 \qquad i = 1, ..., V \qquad (2.1)$$

La séquence de sortie est donc une combinaison linéaire des entrées présente et passées. Cette séquence peut s'exprimer sous la forme du produit de convolution de la séquence d'entrée et de la réponse impultionnelle du codeur (réponse à l'entrée 1000...); d'où le nom des codes convolutionnels.

Une représentation équivalente du codeur de la figure 2.1 est illustrée par la figure 2.2. Cette représentation permet de mettre plus en évidence l'aspect mémoire du code. Le registre à décalage à K cellules correspond à K-1 opérateurs de retard D; et on définit alors l'état du codeur à un instant k par la valeur des bits en mémoire à cet instant. Le nombre d'états que peut prendre le codeur est donc $M=2^{K-1}$.

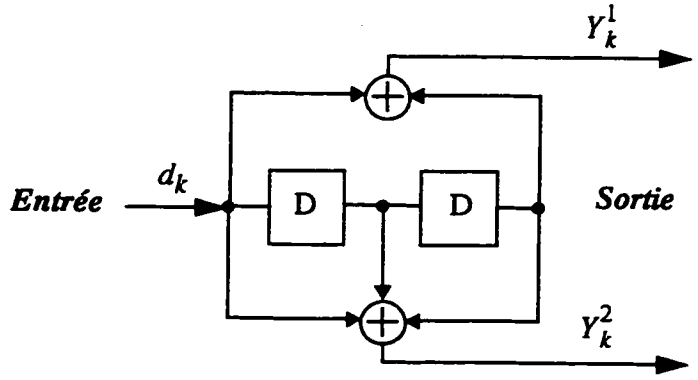


Figure 2.2 Autre représentation du code R=1/2, K=3, G_1=5, G_2=7.

Afin de décoder correctement un code convolutionnel, l'état initial S_0 du codeur et son état final S_N, après codage de N bits, doivent être égaux à 0. Pour ce faire, une séquence de K-1 zéros est ajoutée à la fin de la séquence d'information. Si L est la longueur de la séquence d'information, alors la séquence de sortie est formée de $(L+K-1)V$ symboles. Par conséquent, l'expression exacte du taux de codage est

$$R = \frac{L}{(L+K-1)V} \text{ bits/symboles,} \qquad (2.2)$$

et l'approximation à $1/V$ provient du fait qu'en général $L>>K$.

2.2 Les codes convolutionnels récursifs et systématiques

Un code convolutionnel est dit *systématique* si l'une de ses sorties Y^i (i=1, ..., V), appelée d'ailleurs *sortie systématique*, reproduit la séquence d'information. Si Y^s est

la sortie systématique, alors à tout instant k, $Y_k^s = d_k$. En général, s=1.

Selon Forney (1970) [16], il existe pour chaque code non systématique, un code systématique avec une boucle de contre-réaction, possédant les mêmes propriétés de distance. Avant d'examiner cependant la construction des codes convolutionnels récursifs et systématiques (CRS) à partir des codes convolutionnels non systématiques (CNS), nous commençons par donner la représentation polynomiale d'un code convolutionnel.

L'opérateur de retard D représenté sur la figure 2.2 peut être utilisé comme indicateur de temps. Par exemple, la séquence binaire (1011...) correspond au polynôme $(1+D^2+D^3+...)$. L'équation (2.1) peut alors se mettre sous la forme

$$Y^i(D) = G_i(D)\,d(D) \qquad i = 1, ..., V \qquad (2.3)$$

où $G_i(D)$ sont les polynômes générateurs en D du code, définis par

$$G_i(D) = \sum_{j=0}^{K-1} g_{ij} D^j \qquad i = 1, ..., V \qquad (2.4)$$

Le codeur de la figure 2.1 est repris sur la figure 2.3

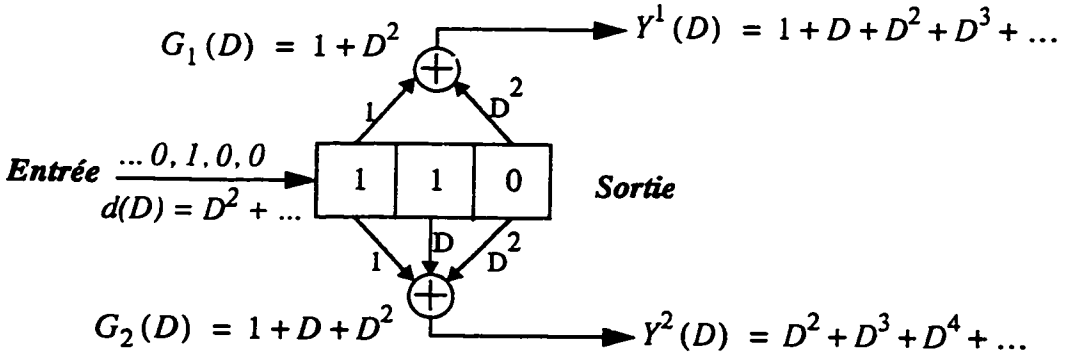


Figure 2.3 Représentation polynomiale du code
R=1/2, K=3, G_1=5, G_2=7

Pour obtenir la version systématique du code de la figure 2.3, l'une des deux sorties $Y^i(D)$, i=1, 2, doit être égale à $d(D)$. Pour ce faire, nous pouvons diviser les termes de l'équation (2.3) par $G_1(D)$ par exemple. On obtient alors

$$\overline{Y}^1(D) = d(D) = \overline{G}_1(D)\,d(D) \tag{2.5a}$$

$$\overline{Y}^2(D) = \frac{G_2(D)}{G_1(D)}d(D) = \overline{G}_2(D)\,d(D) \tag{2.5b}$$

avec $$\overline{Y}^i(D) = \frac{Y^i(D)}{G_1(D)} \qquad i = 1, 2 \tag{2.6}$$

Les polynômes générateurs du code ainsi obtenu sont alors

$$\overline{G}_1(D) = 1 \tag{2.7a}$$

$$\overline{G}_2(D) = \frac{G_2(D)}{G_1(D)} \tag{2.7b}$$

Dans toute la suite, nous adoptons la notation $G = (\overline{G}_1, \overline{G}_2)$, soit pour un code systématique $G = (1, \overline{G}_2) = (1, G_2/G_1)$.

En introduisant le polynôme A(D) défini par

$$A(D) = \frac{d(D)}{G_1(D)}, \tag{2.8}$$

les équations (2.5a) et (2.5b) deviennent

$$\overline{Y}^1(D) = d(D) \tag{2.9a}$$

$$\overline{Y}^2(D) = G_2(D)\,A(D) \tag{2.9b}$$

Le code ainsi construit, dont les sorties sont définies par les équation (2.9a) et (2.9b), est systématique et *récursif*. Le caractère récursif apparait plus clairement en ramenant l'équation (2.8) dans l'espace temporel. Elle devient alors

$$d_k = \sum_{j=0}^{K-1} g_{1j} a_{K-j} \tag{2.10}$$

En tenant compte du fait que toutes les opérations sont faites modulo 2, et en faisant l'hypothèse que g_{10}=1, le symbole a_k peut s'exprimer récursivement en fonction des symboles a_{K-j} (j=1, 2, ..., K-1) et du symbole d_k:

$$a_k = d_k + \sum_{j=1}^{K-1} g_{1j} a_{K-j} \tag{2.11}$$

Par conséquent, dans le cas d'un code CRS, ce sont désormais les symboles a_k qui sont contenus dans le registre à décalage du codeur. A partir des relations (2.9a), (2.9b) et (2.10), les sorties Y^1 et Y^2 du code CRS à un instant k peuvent s'écrire sous la forme

$$Y_k^1 = d_k = \sum_{j=0}^{K-1} g_{1j} a_{K-j} \tag{2.12a}$$

$$Y_k^2 = \sum_{j=0}^{K-1} g_{2j} a_{K-j} \tag{2.12b}$$

La construction du code CRS à partir du code CNS s'est donc faite en conservant les mêmes séquences génératrices, et en substituant simplement les symboles a_k aux symboles d_k.

La figure 2.4 illustre la structure du codeur CRS obtenu à partir du codeur CNS de la figure 2.1

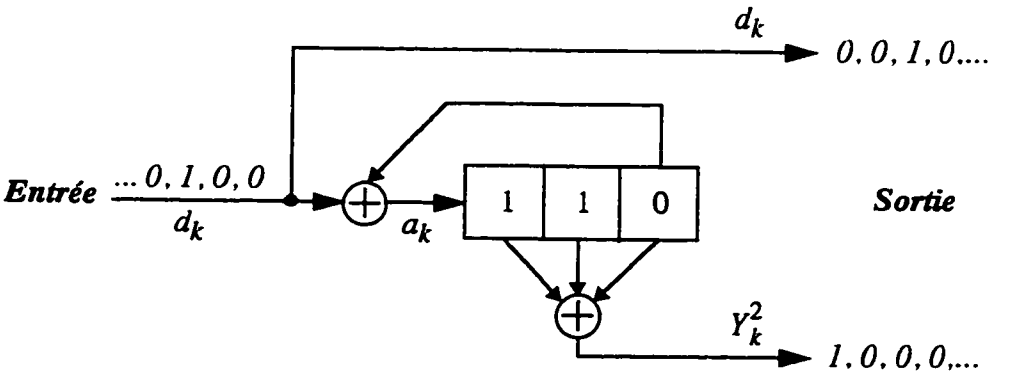


Figure 2.4 Codeur convolutionnel récursif systématique
R=1/2, K=3, G=(1, 7/5).

A l'arrivée d'un bit d_k, celui-ci est additionné modulo 2 aux cellules du registre selon les connexions de l'additionneur à l'entrée du codeur. Le bit a_k résultant de l'addition est alors inséré dans le registre à décalage, et la sortie Y_k^2 calculée.

Contrairement au cas d'un codeur non systématique, la réinitialisation d'un codeur récursif et systématique (remise à zéro du registre à décalage) ne peut se faire par la simple transmission d'une queue de *K-1* zéros. En effet, à cause de la boucle de contre réaction, la séquence nécessaire à la réinitialisation est constituée de *K-1* bits dont un au moins est non nul, et dépend de l'état dans lequel se trouve le codeur après le codage du dernier bit de la séquence d'information.

2.3 Concaténation parallèle de codes

En réalité, comme nous le démontrerons au paragraphe 2.5, le terme "turbo" s'applique plutôt au décodeur qu'au codeur. Néanmoins, la concaténation parallèle de deux codeurs à travers un *entrelaceur*, telle que représentée sur la figure 2.5, est souvent désignée par *codeur turbo*.

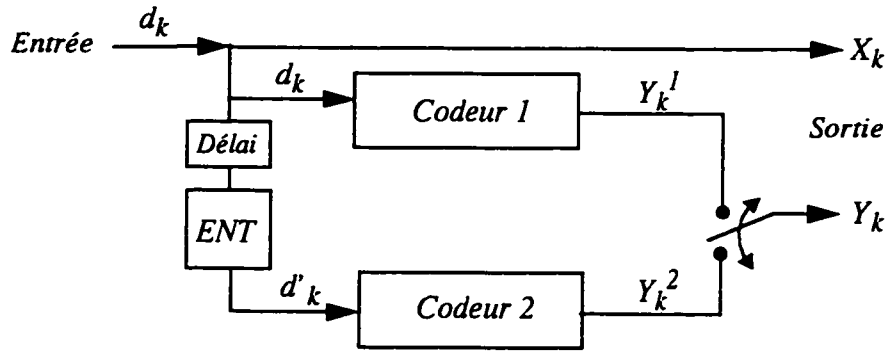


Figure 2.5 Structure du codeur turbo

Cette structure du codeur est appelée concaténation parallèle, du fait que les deux codeurs élémentaires la constituant opèrent sur le même ensemble de bits plutôt que l'un sur la sortie de l'autre, comme cela est le cas par exemple pour une concaténation

en série. En effet, le rôle de l'entrelaceur (ENT) qui sépare les entrées des deux codeurs n'est autre que de permuter l'ordre des bits, de sorte que le même ensemble de bits d'entrée alimente les deux codeurs, mais selon deux séquences différentes. Le délai introduit à l'entrée de Codeur 2 est nécessaire pour la synchronisation de la procédure de décodage. Son rôle sera plus clair au chapitre 4, lorsque nous analyserons le fonctionnement du décodeur turbo.

Ainsi, à un instant k donné, le premier codeur reçoit le bit d_k pour générer la paire (X_k, Y_k^1), tandis que le deuxième codeur reçoit le bit d'_k pour produire la paire (X'_k, Y_k^2). Les bits d'information sont regroupés par blocs de longueur N, qui correspond aussi à la longueur de l'entrelaceur. Le modèle le plus simple de ce dernier peut être représenté par une matrice, où les bits sont écrits ligne par ligne et lus colonne par colonne. Ce type d'entrelaceur est connu sous le nom d'*entrelaceur bloc* ou encore d'*entrelaceur rectangulaire*. La longueur de l'entrelaceur est alors égale au produit des dimensions de la matrice.

Il est à noter que le nombre de codeurs élémentaires formant le codeur turbo peut être supérieur à deux, et que ces derniers ne sont pas nécessairement identiques. Toutefois, ils sont généralement choisis systématiques. Cette dernière caractéristique du codeur turbo, ajoutée au fait que les deux codeurs élémentaires opèrent sur le même ensemble de bits d'entrée, fait qu'il n'est nécessaire de transmettre les sorties systématiques qu'une seule fois. L'utilisation de codeurs élémentaires non systématiques est possible, mais engendre une légère complication du décodeur, sans qu'il n'y ait par ailleurs une modification quelconque des performances d'erreur.

Considérons la concaténation parallèle de deux codeurs systématiques dont les taux de codage sont $R_1 = \frac{b}{V_1}$ et $R_2 = \frac{b}{V_2}$. Le taux de codage global du codeur turbo est alors

$$R = \frac{b}{V_1 + V_2 - b} = \frac{b}{b/R_1 + b/R_2 - b} \text{ bits/symboles.} \qquad (2.13)$$

La soustraction de b au dénominateur est due au fait que les symboles systématiques

ne sont transmis qu'une seule fois, permettant de la sorte d'augmenter le taux de codage global. L'équation (2.13) peut encore s'écrire

$$\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2} - 1. \tag{2.14}$$

Dans le cas du codeur représenté sur la figure 2.5, les taux de codage R_1 et R_2 sont tous deux égaux à 1/2, résultant en un taux global R=1/3. Celui-ci peut toutefois être augmenté à 1/2 par la simple insertion, à la sortie du codeur turbo, d'un commutateur dont la fonction serait de *sélectionner* les séquences de symboles codés $\{Y^1\}$ et $\{Y^2\}$ générées par les deux codeurs élémentaires. La transmission se fait alors de manière à ce que chaque symbole X_k soit accompagné d'un seul symbole de parité Y_k, généré alternativement par l'un ou l'autre des codeurs. Cette technique est appelée *perforation* de codes, et sera présentée plus en détail au chapitre 6 de ce mémoire.

2.4 Modulation et transmission des symboles codés

Pour toutes les simulations que nous avons effectuées et que nous présentons dans la suite, nous avons utilisé un modulateur de type BPSK (*Binary Phase Shift Keying*) suivi d'un canal à bruit blanc additif gaussien.

2.4.1 Modulation BPSK

Le nombre de bits par symbole pour ce type de modulation est égal à 1. En appliquant les sorties X_k et Y_k du codeur turbo à l'entrée du modulateur de la figure 2.6, les sorties générées par celui-ci sont $\tilde{X}_k$ et $\tilde{Y}_k$, avec

$$\begin{cases} \tilde{X}_k = 2X_k - 1 \\ \tilde{Y}_k = 2Y_k - 1 \end{cases} \tag{2.15}$$

Les valeurs -1 et +1 de $\tilde{X}_k$ et $\tilde{Y}_k$, transmises dans le canal, sont ainsi respectivement associées aux symboles codés X_k et Y_k valant '0' et '1'.

Figure 2.6 Le modulateur BPSK

2.4.2 Le canal

Les sorties du modulateur sont à leur tour transmises sur un canal sans mémoire, dont un modèle est représenté sur la figure 2.7, où vient s'ajouter un bruit blanc gaussien de moyenne nulle, défini par sa densité spectrale de puissance unilatérale N_0.

.

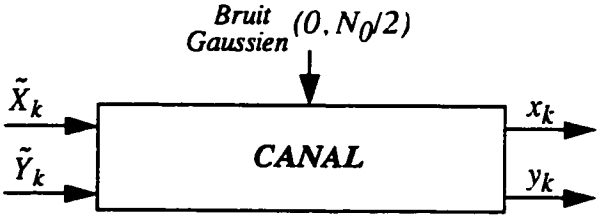


Figure 2.7 Modèle du canal à bruit blanc gaussien additif

Les sorties x_k et y_k du canal, qui comprend un filtre adapté ou autre récepteur optimal, peuvent alors s'écrire comme suit

$$\begin{cases} x_k = \tilde{X}_k + p_k \\ y_k = \tilde{Y}_k + q_k \end{cases} \tag{2.16}$$

où p_k et q_k sont deux variables aléatoires gaussiennes indépendantes, de variance σ^2. Leur densité de probabilité est définie par la fonction

$$p(y) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{1}{2\sigma^2}y^2\right) \tag{2.17}$$

avec $$\sigma^2 = \left(2R\frac{E_b}{N_0}\right)^{-1} \tag{2.18}$$

Sur la figure 2.8, nous avons représenté le schéma global du système à l'émission, incluant le canal.

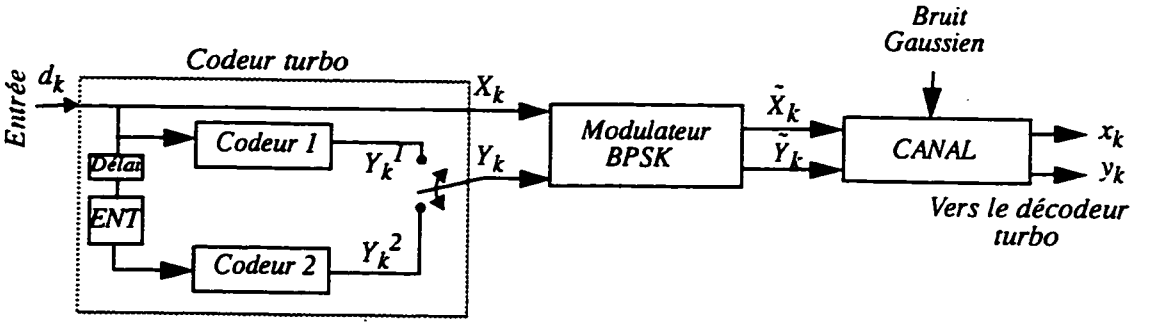


Figure 2.8 Structure du système à l'émission, codage turbo, R=1/2

2.5 Le décodeur turbo

S'agissant d'abord et avant tout d'une concaténation de codes, l'objectif premier du décodage turbo est de décomposer la procédure de décodage en plusieurs étapes, afin de réduire la complexité du système global, tout en gardant un haut niveau de performance. La figure 2.9 illustre l'idée du décodage turbo, proposée par Berrou, Glavieux et Thitimajshima en 1993 [7], dans le cas d'une concaténation de deux codeurs de taux R=1/2 chacun, et avec un taux global également de 1/2.

La première remarque que l'on peut faire relativement à la structure de ce décodeur, est qu'il s'agit non plus d'une concaténation parallèle, mais d'une concaténation en série des décodeurs. Rappelons que les deux codeurs élémentaires qui constituent le codeur turbo codent le même ensemble de bits; il s'agit donc de la même façon ici de décoder "deux fois" ce même ensemble. Ainsi, le premier décodeur (Décodeur 1) reçoit du canal les séquences $\{x_k\}$ et $\{y_k^1\}$, correspondant respectivement aux symboles systématiques et aux symboles de parité générés par Codeur 1.

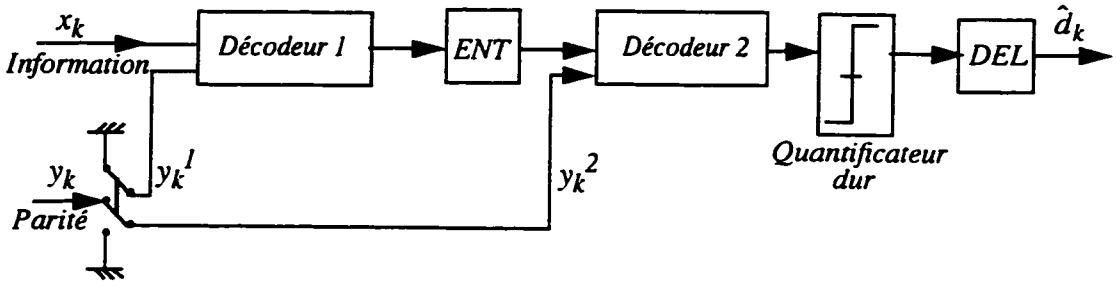


Figure 2.9 Concaténation en série des décodeurs

On remarquera à ce niveau la présence à l'entrée du décodeur d'un module de démultiplexage qui permet, quand le taux de codage global est égal à 1/2, d'acheminer vers chaque décodeur élémentaire les symboles de parité générés par le codeur qui lui est associé. Ce même module insère des '0' à la place des symboles qui n'ont pas été transmis.

A partir des symboles qu'il reçoit, Décodeur 1 génère pour chaque symbole décodé une valeur mesurant son degré de fiabilité, appelée *Probabilité A Posteriori* (PAP). Cette dernière peut être vue comme étant la probabilité d'avoir pris ou non la bonne décision lors du décodage du symbole en question. L'algorithme utilisé par Décodeur 1 devra donc nécessairement effectuer un décodage par symbole et non un décodage par séquence. Le même entrelaceur que celui employé au niveau du codeur est utilisé pour entrelacer les PAP générées par Décodeur 1 avant de les passer à Décodeur 2. Celui-ci dispose alors de la part de décision du premier décodeur dans le décodage du bloc reçu, plus les symboles de parité retardés, générés par le deuxième codeur à l'émission. Un algorithme de décodage de séquences peut alors être utilisé à ce niveau pour générer la séquence de bits décodés; séquence qui sera d'abord *délacée*, c'est à dire remise dans l'ordre original, avant d'être envoyée à la destination.

Dans la description que nous venons de faire de la concaténation des deux décodeurs, un point important est à noter: le premier décodeur n'utilise qu'une partie des symboles de parité, générés par Codeur 1. Le décodage global n'est donc pas optimal. Cette imperfection peut toutefois être éliminée par l'ajout d'une boucle de contre-réaction dans le système, ainsi que par l'utilisation d'un algorithme de décodage de symboles pour le deuxième décodeur. Le décodeur résultant est illustré par la figure 2.10

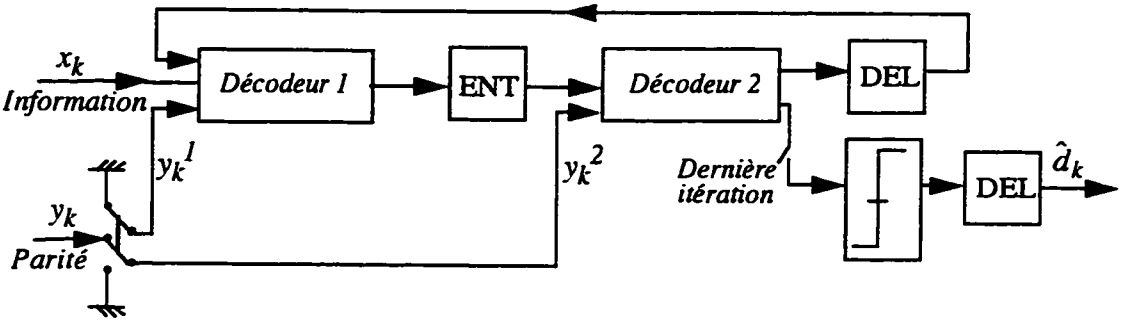


Figure 2.10 Structure du décodeur turbo: schéma de principe

A la première itération, le fonctionnement du décodeur turbo se fera de façon identique à ce qui a été décrit plus haut, à la différence que le deuxième décodeur élémentaire utilise maintenant, comme le premier, un algorithme de décodage de symboles. Une information de fiabilité est donc également générée par Décodeur 2. Cette information est passée dans le délaceur (DEL), puis retransmise au premier décodeur à travers la boucle de retour, afin d'être utilisée comme *information à priori* à la prochaine itération. Ainsi, à partir de la deuxième itération, Décodeur 1 dispose non plus de deux, mais de trois entrées; et grâce à l'information supplémentaire qu'il reçoit de Décodeur 2, sa performance peut s'améliorer de façon appréciable d'itération en itération; d'où le nom des codes turbo par référence au principe des moteurs turbo. Au bout d'un nombre déterminé d'itérations, la sortie de Décodeur 2 est quantifiée et une estimation de la séquence transmise délivrée.

2.6 Conclusion

Nous avons présenté dans ce chapitre une description de la structure globale des codeur et décodeur turbo ainsi qu'un modèle du canal de transmission utilisé.

Dans les chapitre 3 à 5 qui suivent, nous procéderons à l'analyse des différents paramètres du système. Nous présenterons ainsi l'algorithme de décodage qui constitue le noyau du décodeur turbo, et démontrerons l'importance du choix des codeurs élémentaires et de l'entrelaceur.

Nous garderons cependant inchangé, tout au long de ce mémoire, le modèle du canal présenté dans ce chapitre, avec une modulation BPSK et un bruit blanc gaussien additif. Par ailleurs, à moins qu'il n'en soit indiqué autrement, nous supposerons que les codeurs élémentaires sont convolutionnels, récursifs et systématiques. Une justification de ce choix sera fournie au chapitre 5.

Chapitre 3
L'ALGORITHME DE DÉCODAGE

3.1 Introduction

A la lumière du principe de fonctionnement du décodeur turbo présenté au chapitre 2, il ressort que la condition première que doit remplir un algorithme susceptible d'être utilisé dans un tel système est la génération d'une information de fiabilité pour chaque bit décodé. Cette information, appelée probabilité a posteriori (PAP), est une mesure de la probabilité d'avoir pris ou non la bonne décision lors du décodage.

L'algorithme de Viterbi, très utilisé jusqu'à aujourd'hui pour le décodage des codes convolutionnels, est un algorithme à maximum de vraisemblance qui minimise la probabilité d'erreur par séquence. Il ne garantit pas cependant la minimisation de la probabilité d'erreur par bit (ou symbole).

En 1966, Chang et Hancock [12], ont développé un algorithme minimisant la probabilité d'erreur par symbole, ou encore maximisant la probabilité à posteriori, pour un canal à interférences inter-symboles; d'où d'ailleurs son nom: MAP, pour Maximum A Posteriori. En 1972, Bahl *et al.* [1] et McAdam *et al.* [28] ont simultanément adapté l'algorithme MAP aux codes correcteurs d'erreurs. A de faibles taux d'erreur par bit (TEB), la différence de performance entre l'algorithme MAP et l'algorithme de Viterbi est quasi négligeable. L'algorithme MAP étant cependant beaucoup plus complexe, il a été longtemps ignoré. Toutefois, à de faibles rapports signal sur bruit (RSB), et pour des TEB élevés, les performances de MAP sont supérieures à celles de l'algorithme de Viterbi de quelque 0,3dB. Il est vrai qu'à priori même cette différence de performance n'est pas très avantageuse si l'on tient compte de la différence de complexité; mais il n'en demeure pas moins que dans un processus itératif, où le TEB est généralement très élevé à la fin de la première itération, l'amélioration apportée

par l'algorithme MAP résulte en une croissance rapide des performances aux itérations suivantes. C'est ainsi que cet algorithme fut adapté au décodage des codes convolutionnels systématiques et utilisé pour le décodage turbo [7]. Malgré les performances exceptionnelles présentées dans [7] cependant, un inconvénient majeur subsistait: la complexité du décodage. La version originale de l'algorithme MAP nécessitait en effet énormément d'espace mémoire et les calculs y étaient, inutilement parfois, très complexes.

Une première simplification de l'algorithme a été proposée par Robertson [34], et reprise d'ailleurs par Berrou *et al.* dans [8]. D'autres simplifications suivirent, dont la plus importante consistait à exprimer l'algorithme dans le domaine logarithmique [6,20,31,35]. La version qui en a résulté est souvent désignée par log-MAP. D'autres versions sous-optimales de l'algorithme ont encore été proposées [22,25,32], mais parfois au prix d'une légère baisse des performances.

Un autre algorithme de décodage minimisant la probabilité d'erreur par symbole a été proposé [18]. Il s'agit d'une version modifiée de l'algorithme de Viterbi, connue sous le nom de SOVA (*Soft Output Viterbi Algorithm*). Cet algorithme, dont la complexité est moindre que celle de l'algorithme MAP, a été également utilisé pour le décodage des codes turbo [19,20,38]. Toutefois, les propriétés statistiques des sorties générées par SOVA sont moins bonnes que celles fournies par l'algorithne MAP, ce qui résulte en une diminution globale des performances de l'ordre de 0,8dB.

Dans ce chapitre, nous présentons la dérivation et les différentes étapes de la version optimale de l'algorithme MAP, ainsi que les simplifications qui mènent respectivement à la version log-MAP, et à une version sous-optimale de l'algorithme. Nous fournissons également une brève comparaison des performances obtenues avec les algorithmes MAP et SOVA. Une description des résultats obtenus par simulation de l'algorithme sera faite à la fin du chapitre.

3.2 L'algorithme MAP

La version de l'algorithme que nous présentons ici comporte une simple modification par rapport à la dérivation faite dans [7]. Modification apportée par S. Pietrobon [31], dont nous utiliserons d'ailleurs les notations, qui s'avère, comme nous le démontrerons, très précieuse pour la simplification des calculs. Par ailleurs, bien que l'algorithme puisse être utilisé pour différents types de codes, tels que les codes en blocs [20], nous nous limiterons dans notre cas aux codes convolutionnels systématiques (et récursifs). La généralisation à d'autres types de codes reste cependant relativement simple.

3.2.1 Description de l'algorithme

Considérons un codeur convolutionnel systématique de taux $R=1/2$, à M états ($M=2^{K-1}$, K étant la longueur de contrainte du code). Supposons qu'à un instant k, le bit d'information d_k soit présent à l'entrée du codeur, dont l'état est $S_k=m$; $m=0, 1, ..., M-1$. Les sorties du codeur seraient alors, le symbole systématique $X_k=d_k$, et le symbole de parité Y_k. Nous définissons alors, au niveau du récepteur, le logarithme du rapport de vraisemblance (LRV), $L(d_k)$, par

$$L(d_k) = ln\frac{Pr(d_k = 1|observation)}{Pr(d_k = 0|observation)}, \qquad (3.1)$$

où $Pr(d_k = i|observation)$ est la PAP du bit d'information d_k. Nous supposons également que la séquence de bits d'information $\{d_k\}$ est formée de N bits indépendants, prenant les valeurs 0 et 1 avec égales probabilités, et que l'état initial S_0 du codeur et son état final S_N sont tous deux égaux à zéro.

A la sortie du canal, la séquence reçue est

$$R_1^N = (R_1, ..., R_k, ..., R_N), \qquad (3.2)$$

où $R_k = (x_k, y_k)$ est la paire de symboles reçus pour le bit d_k, et où

$$x_k = (2d_k - 1) + p_k, \tag{3.3}$$

$$y_k = (2Y_k - 1) + q_k, \tag{3.4}$$

p_k et q_k étant deux variables aléatoires gaussiennes centrées et indépendantes, de variance égale à σ^2

Nous définissons à présent la probabilité conjointe suivante:

$$\lambda_k^i(m) = Pr(d_k = i, S_k = m | R_1^N), \tag{3.5}$$

avec $i=0, 1$ et $m=0, 1, \ldots, M-1$. La PAP associée au bit décodé d_k est alors égale à

$$Pr(d_k = i | R_1^N) = \sum_{m=0}^{M-1} \lambda_k^i(m), \qquad i=0, 1. \tag{3.6}$$

En remplaçant l'expression de la PAP dans (3.1) par celle obtenue dans (3.6), le LRV associé au bit d_k devient

$$L(d_k) = ln \frac{\sum_m \lambda_k^1(m)}{\sum_m \lambda_k^0(m)} \tag{3.7}$$

Cette valeur représente la sortie non quantifiée du décodeur MAP. Elle peut être utilisée comme donnée d'entrée pour un autre décodeur dans le cas d'une concaténation, ou encore à la prochaine itération dans un décodeur itératif. A la fin de la dernière itération, le décodage se fait selon la règle de décision suivante:

si $L(d_k) \geq 0$, alors le bit décodé est égal à 1;

si $L(d_k) < 0$, alors le bit décodé est égal à 0.

3.2.2 Définition des fonctions α et β

Afin de calculer la probabilité conjointe définie en (3.5), il est utile d'introduire les fonctions de probabilité suivantes, proposées par Pietrobon [31]

$$\alpha_k^i(m) = Pr(d_k = i, S_k = m, R_1^k), \tag{3.8}$$

$$\beta_k^i(m) = Pr(R_{k+1}^N \Big| d_k = i, S_k = m), \tag{3.9}$$

où i représente la valeur du bit à l'instant k. et $R_{k_1}^{k_2}$ la séquence de symboles reçus de l'instant k_1 à l'instant k_2. Il est à noter que ces définitions sont différentes de celles proposées dans [7], du fait que α et β dépendent toutes deux de la valeur du bit d'information d_k. Cette modification s'avérera très importante pour la simplification de l'expression finale du LRV, $L(d_k)$.

En utilisant la règle de Bayes, la probabilité conjointe dans (3.5) peut s'écrire

$$\lambda_k^i(m) = \frac{Pr(d_k = i, S_k = m, R_1^k, R_{k+1}^N)}{Pr(R_1^N)}, \tag{3.10}$$

où l'on a décomposé la séquence R_1^N en (R_1^k, R_{k+1}^N). L'équation (3.10) peut encore être développée sous la forme

$$\lambda_k^i(m) = \frac{Pr(d_k = i, S_k = m, R_1^k)\, Pr(R_{k+1}^N \mid d_k = i, S_k = m, R_1^k)}{Pr(R_1^N)}. \tag{3.11}$$

Sachant que les événements qui se produisent après l'instant k ne sont pas influencés par les observations précédant cet instant, puisque d_k et S_k sont connus, (3.11) peut être simplifiée en

$$\lambda_k^i(m) = \frac{Pr(d_k = i, S_k = m, R_1^k)\, Pr(R_{k+1}^N \mid d_k = i, S_k = m)}{Pr(R_1^N)}. \tag{3.12}$$

En considérant les équations (3.8) et (3.9), (3.12) devient

$$\lambda_k^i(m) = \frac{\alpha_k^i(m)\,\beta_k^i(m)}{Pr(R_1^N)}. \tag{3.13}$$

L'avantage de cette expression de la probabilité conjointe est que le dénominateur ne dépend pas de la valeur du bit d'information (l'indice i). Utilisant maintenant (3.13) dans (3.7), nous obtenons

$$L(d_k) = ln \frac{\sum_m \alpha_k^1(m)\,\beta_k^1(m)}{\sum_m \alpha_k^0(m)\,\beta_k^0(m)}. \tag{3.14}$$

Il s'agit maintenant de calculer à la prochaine étape les probabilités α_k^i et β_k^i. Nous démontrons dans les paragraphes suivant que les α_k^i peuvent être calculés de façon récursive croissante (c.à.d en utilisant les valeurs antérieures), de même que les β_k^i sont calculés de façon récursive décroissante (c.à.d en utilisant les valeurs futures, d'indice temporel supérieur). Le lecteur voulant éviter les détails de calcul, peut passer directement aux étapes de l'algorithme à la section 3.2.6.

3.2.3 Évaluation de α

L'équation (3.8) peut être réécrite sous la forme

$$\alpha_k^i(m) = Pr(d_k = i, S_k = m, R_1^{k-1}, R_k), \tag{3.15}$$

où l'on a décomposé la séquence reçue R_1^k en la séquence correspondant aux symboles reçus jusqu'à l'instant *k-1*, plus le symbole reçu à l'instant *k*. Cette probabilité peut encore être exprimée sous forme d'une somme sur toutes les transitions possibles de l'instant *k-1* à l'instant *k*, où l'on passe de l'état $S_{k-1}=n$ à $S_k=m$:

$$\begin{aligned}
\alpha_k^i(m) &= \sum_n \sum_{j=0}^{1} Pr(d_k = i, d_{k-1} = j, S_k = m, S_{k-1} = n, R_1^{k-1}, R_k) \\
&= \sum_n \sum_{j=0}^{1} Pr(d_{k-1} = j, S_{k-1} = n, R_1^{k-1}) \\
&\quad \times Pr(d_k = i, S_k = m, R_k \mid d_{k-1} = j, S_{k-1} = n, R_1^{k-1}) \\
&= \sum_n \sum_{j=0}^{1} Pr(d_{k-1} = j, S_{k-1} = n, R_1^{k-1}) \\
&\quad \times Pr(d_k = i, S_k = m, R_k \mid d_{k-1} = j, S_{k-1} = n) .
\end{aligned} \tag{3.16}$$

L'équation (3.16) a été obtenue en utilisant la règle de Bayes, et en tenant compte du fait que la séquence R_1^{k-1} est inutile au calcul puisque d_{k-1} et S_{k-1} définissent sans ambiguïté le chemin à l'instant *k-1*.

Nous définissons à présent la fonction

$$\gamma_{i,j}(R_k, m, n) = Pr(d_k = i, S_k = m, R_k | d_{k-1} = j, S_{k-1} = n) , \quad (3.17)$$

qui sera par la suite simplifiée. Notons simplement que cette fonction sera déterminée par les probabilités de transition du canal ainsi que par celles du treillis du codeur. Nous pouvons cependant déjà remarquer que

$$Pr(d_{k-1} = j, S_{k-1} = n, R_1^{k-1}) = \alpha_{k-1}^j(n) . \quad (3.18)$$

En combinant (3.16), (3.17) et (3.18), nous obtenons l'équation récursive suivante

$$\alpha_k^i(m) = \sum_n \sum_{j=0}^{1} \alpha_{k-1}^j(n) \gamma_{i,j}(R_k, m, n). \quad (3.19)$$

En supposant que l'état initial du codeur est égal à zéro, tous les paramètres $\alpha_0^i(m)$, pour tout état m non nul et tout $i=0, 1$, doivent être initialisés à zéro. Comme nous le verrons plus loin, seul $\alpha_0^i(0)$ sera initialisé à une valeur non nulle.

3.2.4 Evaluation de β

De la même façon, après la réception de tout le bloc de données, les probabilités $\beta_k^i(m)$ peuvent être calculées de façon récursive à partir des probabilités $\beta_{k+1}^i(m)$. La relation (3.9) peut s'écrire

$$\begin{aligned}
\beta_k^i(m) &= Pr(R_{k+1}, R_{k+2}^N | d_k = i, S_k = m) \\
&= \sum_n \sum_{j=0}^{1} Pr(d_{k+1} = j, S_{k+1} = n, R_{k+1}, R_{k+2}^N | d_k = i, S_k = m) \\
&= \sum_n \sum_{j=0}^{1} Pr(R_{k+2}^N | d_{k+1} = j, S_{k+1} = n, R_{k+1}, d_k = i, S_k = m) \\
&\quad \times Pr(d_{k+1} = j, S_{k+1} = n, R_{k+1} | d_k = i, S_k = m) . \quad (3.20)
\end{aligned}$$

Dans le premier terme du produit sous la double somme, la présence du bit d'information et de l'état du codeur à l'instant k n'est pas nécessaire, tous deux étant déjà repré-

sentés à l'instant $k+1$. De même que R_{k+1} est inutile puisque le chemin à l'instant $k+1$ est parfaitement déterminé. L'équation (3.20) devient alors

$$\beta_k^i(m) = \sum_n \sum_{j=0}^{1} Pr(R_{k+2}^N \mid d_{k+1} = j, S_{k+1} = n)$$

$$\times \quad Pr(d_{k+1} = j, S_{k+1} = n, R_{k+1} \mid d_k = i, S_k = m). \tag{3.21}$$

En remplaçant dans (3.17) l'indice temporel k par $k+1$, et en permutant les indices i et j ainsi que les états m et n, nous obtenons

$$\gamma_{j,i}(R_{k+1}, n, m) = Pr(d_{k+1} = j, S_{k+1} = n, R_{k+1} \mid d_k = i, S_k = m). \tag{3.22}$$

On peut alors remarquer comme précédemment que

$$Pr(R_{k+2}^N \mid d_{k+1} = j, S_{k+1} = n) = \beta_{k+1}^j(n). \tag{3.23}$$

Combinant (3.21), (3.22) et (3.23), nous obtenons l'équation récursive

$$\beta_k^i(m) = \sum_n \sum_{j=0}^{1} \beta_{k+1}^j(n)\, \gamma_{j,i}(R_{k+1}, n, m). \tag{3.24}$$

L'initialisation des $\beta_N^i(m)$ sera effectuée plus loin.

3.2.5 Évaluation de γ et représentation graphique

La probabilité $\gamma_{i,j}(R_k, m, n)$ peut être calculée à partir des probabilités de transitions du canal gaussien sans mémoire, et des probabilités de transition du treillis du codeur. Utilisant encore une fois la règle de Bayes nous avons

$$\gamma_{i,j}(R_k, m, n) = Pr(d_k = i, S_k = m, R_k \mid d_{k-1} = j, S_{k-1} = n)$$

$$= Pr(R_k \mid d_k = i, S_k = m, d_{k-1} = j, S_{k-1} = n)$$

$$\times \quad Pr(d_k = i \mid S_k = m, d_{k-1} = j, S_{k-1} = n)$$

$$\times \quad Pr(S_k = m \mid d_{k-1} = j, S_{k-1} = n)$$

Sachant que la connaissance du bit d'information ainsi que l'état du codeur à un instant donné définit parfaitement le chemin dans le treillis à cet instant, et que les bits d'information sont équiprobables, nous pouvons écrire

$$\gamma_{i,j}(R_k, m, n)$$

$$= Pr(R_k | d_k = i, S_k = m)\, Pr(d_k = i)\, Pr(S_k = m | d_{k-1} = j, S_{k-1} = n)$$

$$= Pr(R_k | d_k = i, S_k = m)\ (1/2)\, Pr(S_k = m | d_{k-1} = j, S_{k-1} = n).$$

Si de $S_{k-1} = n$, sachant que $d_{k-1} = j$, on allait à $S_k = m$, alors $Pr(S_k = m | d_{k-1} = j, S_{k-1} = n) = 1$ et l'équation ci-dessus devient

$$\gamma_{i,j}(R_k, m, n) = \frac{1}{2} Pr(R_k | d_k = i, S_k = m), \tag{3.25}$$

autrement, $Pr(S_k = m | d_{k-1} = j, S_{k-1} = n) = 0$ et $\gamma_{i,j}(R_k, m, n) = 0$. C'est ainsi que disparaît la sommation sur tous les n dans (3.19), et le seul n survivant est celui auquel on aboutirait si, partant de l'état m on revenait à l'instant précédant suivant la branche $d_{k-1} = j$. Cet état sera désigné dans la suite par $S_p^j(m)$, et l'équation (3.19) devient alors

$$\alpha_k^i(m) = \sum_{j=0}^{1} \alpha_{k-1}^j(S_p^j(m))\ (1/2)\, Pr(R_k | d_k = i, S_k = m)$$

$$= \frac{1}{2} Pr(R_k | d_k = i, S_k = m) \sum_{j=0}^{1} \alpha_{k-1}^j(S_p^j(m)). \tag{3.26}$$

Pour plus de clarté, l'équation (3.26) est illustrée par la figure 3.1. Sur cette figure, $\alpha_k^i(m)$ et $\frac{1}{2} Pr(R_k | d_k = i, S_k = m)$ peuvent être considérés respectivement comme étant la métrique d'état et la métrique de transition.

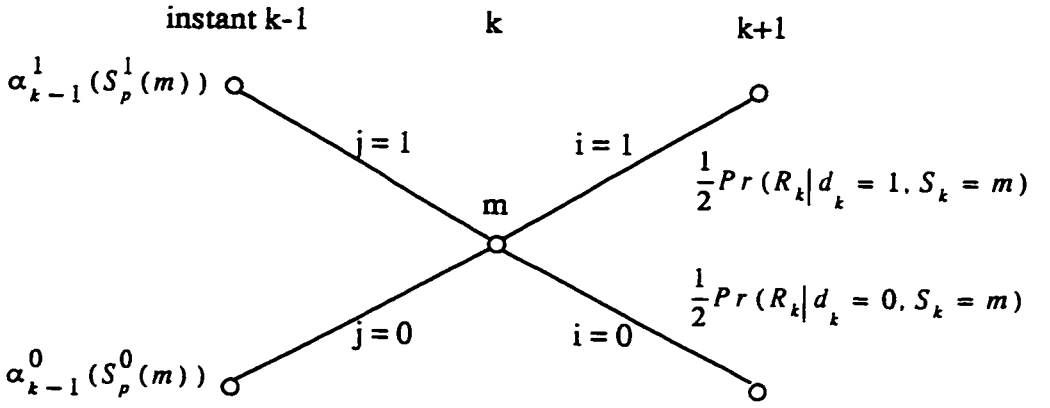


Figure 3.1 Représentation graphique de l'équation (3.26)

Revenant à la probabilité $\gamma_{j,i}(R_{k+1}, n, m)$ définie dans (3.22), nous pouvons la transformer de la même façon:

$$\begin{aligned}\gamma_{j,i}(R_{k+1}, n, m) &= Pr(d_{k+1} = j, S_{k+1} = n, R_{k+1}|d_k = i, S_k = m)\\ &= Pr(R_{k+1}|d_{k+1} = j, S_{k+1} = n, d_k = i, S_k = m)\\ &\quad \text{x}\ Pr(d_{k+1} = j|S_{k+1} = n, d_k = i, S_k = m)\\ &\quad \text{x}\ Pr(S_{k+1} = n|d_k = i, S_k = m)\end{aligned}$$

Comme nous avons procédé plus haut, $\gamma_{j,i}(R_{k+1}, n, m)$ peut être décomposée sous la forme suivante

$$Pr(R_{k+1}|d_{k+1} = j, S_{k+1} = n)\, Pr(d_{k+1} = j)\, Pr(S_{k+1} = n|d_k = i, S_k = m)$$

$$= Pr(R_{k+1}|d_{k+1} = j, S_{k+1} = n)\ (1/2)\, Pr(S_{k+1} = n|d_k = i, S_k = m)\ .$$

Si l'on va maintenant de l'état $S_k = m$ à l'état $S_{k+1} = n$, sachant que $d_k = i$, alors $Pr(S_{k+1} = n|d_k = i, S_k = m) = 1$ et l'équation ci-dessus devient

$$\gamma_{j,i}(R_{k+1}, n, m) = \frac{1}{2}Pr(R_{k+1}|d_{k+1} = j, S_{k+1} = n), \qquad (3.27)$$

sinon, $Pr(S_{k+1} = n | d_k = i, S_k = m) = 0$ et $\gamma_{j,i}(R_k, n, m) = 0$. La sommation sur tous les n disparaît dans (3.24), et le seul n survivant sera représenté par l'état $S_s^i(m)$, auquel on arriverait en partant de l'état m vers l'état suivant en suivant la branche sur laquelle $d_k = i$. L'équation (3.24) devient alors

$$\beta_k^i(m) = \frac{1}{2}\sum_{j=0}^{1} \beta_{k+1}^j(S_s^i(m))\, Pr(R_{k+1} | d_{k+1} = j, S_{k+1} = S_s^i(m)). \qquad (3.28)$$

Une représentation graphique de cette équation est donnée par la figure 3.2, où l'on a définit $\delta_{k+1}^j(S_s^i(m)) = \frac{1}{2} Pr(R_{k+1} | d_{k+1} = j, S_{k+1} = S_s^i(m))$.

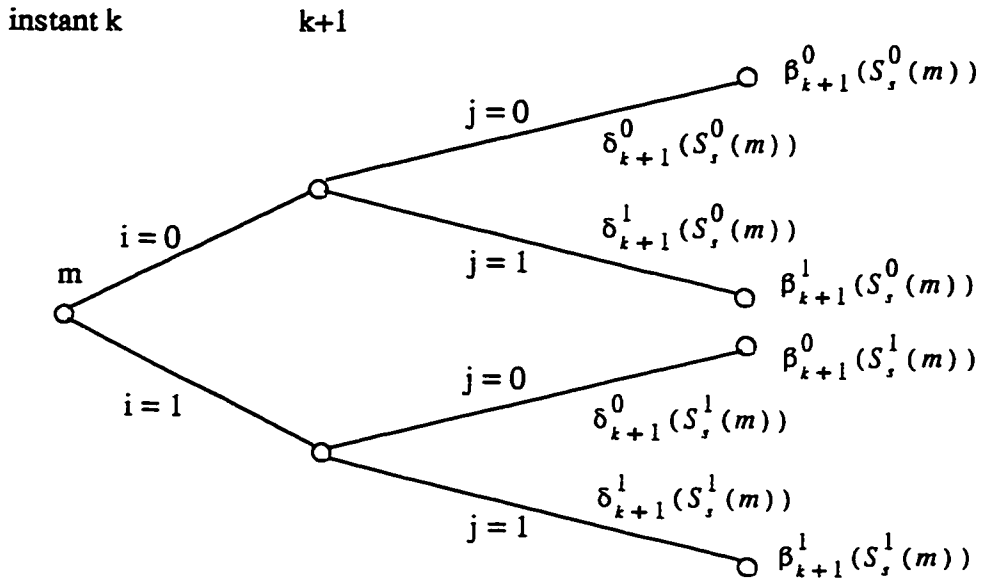


Figure 3.2 Représentation graphique de (3.28)

A partir des nouvelles expressions de $\alpha_k^i(m)$ et $\beta_k^i(m)$ dans (3.26) et (3.28), nous pouvons maintenant recalculer le LRV. Mais avant cela, on se propose d'exprimer

$$\delta_k^i(m) = Pr(R_k | d_k = i, S_k = m)$$

en fonction des paramètres du canal. Pour ce faire, on considère un canal gaussien à bruit blanc additif, de moyenne nulle et de variance σ^2 . Rappelons aussi que les sym-

boles x_k et y_k reçus par le décodeur turbo sont associés respectivement aux sorties X_k=d_k et Y_k du codeur, selon les relations $x_k = (2d_k - 1) + p_k$,

$$y_k = (2Y_k - 1) + q_k,$$

et que p_k et q_k sont deux variables aléatoires gaussiennes indépendantes, de moyenne nulle et de variance σ^2. Par conséquent, x_k et y_k sont aussi deux variables aléatoires indépendantes de moyennes respectives $(2d_k - 1)$ et $(2Y_k - 1)$ et de variance σ^2. On peut alors exprimer la densité $Pr(x_k)$ comme suit:

$$Pr(x_k) = Pr(x = x_k) = \lim_{\Delta x \to 0} \int_{(x_k - \frac{\Delta x}{2})}^{(x_k - \frac{\Delta x}{2})} \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{1}{2\sigma^2}(u - (2d_k - 1))^2\right) du$$

$$= \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{1}{2\sigma^2}(x_k - (2d_k - 1))^2\right) dx,$$

De la même façon, on peut mettre la densité $Pr(y_k)$ sous la forme

$$Pr(y_k) = \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{1}{2\sigma^2}(y_k - (2Y_k - 1))^2\right) dy,$$

Ainsi, les probabilités δ_k^i peuvent être exprimées en fonction des densités de probabilité comme suit:

$$\delta_k^i(m) = Pr(x_k | d_k = i, S_k = m)\, Pr(y_k | d_k = i, S_k = m)$$

$$= \frac{dx}{\sigma\sqrt{2\pi}} exp\left(-\frac{1}{2\sigma^2}(x_k - (2d_k - 1))^2\right) \frac{dy}{\sigma\sqrt{2\pi}} exp\left(-\frac{1}{2\sigma^2}(y_k - (2Y_k - 1))^2\right)$$

$$= \frac{1}{2\pi\sigma^2} exp\left(-\frac{x_k^2 + y_k^2 + 2}{2\sigma^2}\right) exp\left(\frac{2(x_k d_k + y_k Y_k) - x_k - y_k}{\sigma^2}\right) dxdy$$

$$= \frac{1}{2\pi\sigma^2} exp\left(-\frac{x_k^2 + y_k^2 + 2}{2\sigma^2}\right) exp\left(\frac{-x_k - y_k}{\sigma^2}\right) exp\left(\frac{2(x_k d_k + y_k Y_k)}{\sigma^2}\right) dxdy$$

$$= K_k exp\left(\frac{x_k d_k + y_k Y_k}{\sigma^2/2}\right)$$

$$= K_k exp\left(\frac{x_k i + y_k Y_k(i,m)}{\sigma^2/2}\right) \tag{3.29}$$

où K_k est une constante. Dans (3.29), nous avons exprimé Y_k sous la forme $Y_k(i,m)$ afin de souligner le fait que Y_k est fonction du bit d'information ainsi que de l'état du codeur. De la même façon, des simplifications peuvent être apportées à la probabilité suivante

$$Pr(R_{k+1} | d_{k+1} = j, S_{k+1} = S_s^i(m))$$

$$= K_k exp\left(\frac{x_{k+1} j + y_{k+1} Y_{k+1}(j, S_s^i(m))}{\sigma^2/2}\right). \tag{3.30}$$

Substituant (3.29) et (3.30) dans (3.26) et (3.28), nous obtenons

$$\alpha_k^i(m) = C_k^{\alpha} exp\left(\frac{x_k i + y_k Y_k(i,m)}{\sigma^2/2}\right) \sum_{j=0}^{1} \alpha_{k-1}^i(S_p^j(m)) \tag{3.31}$$

$$\beta_k^i(m) = C_k^{\beta} \sum_{j=0}^{1} \beta_{k+1}^j(S_s^i(m)) \, exp\left(\frac{x_{k+1} j + y_{k+1} Y_{k+1}(j, S_s^i(m))}{\sigma^2/2}\right) \tag{3.32}$$

où C_k^{α} et C_k^{β} sont deux constantes. Comme ces dernières disparaîtront lors de la réévaluation du LRV dans (3.14), les équations (3.31) et (3.32) peuvent être reformulées comme suit

$$\alpha_k^i(m) = \delta_k^i(m) \sum_{j=0}^{1} \alpha_{k-1}^j(S_p^j(m)) \tag{3.33}$$

$$\beta_k^i(m) = \sum_{j=0}^{1} \beta_{k+1}^j(S_s^i(m)) \, \delta_{k+1}^j(S_s^i(m)) \tag{3.34}$$

Le logarithme du rapport de vraisemblance peut alors être calculé:

$$L(d_k) = ln \frac{\sum_m \alpha_k^1(m)\,\beta_k^1(m)}{\sum_m \alpha_k^0(m)\,\beta_k^0(m)} \tag{3.35}$$

3.2.6 Les étapes de l'algorithme

Rappelons que les symboles x_k et y_k correspondent respectivement aux valeurs bruitées des symboles d'information et de parité reçues par le décodeur, et que $Y_k(i, m)$ est le symbole de parité que génèrerait l'un des codeurs élémentaires s'il recevait le bit d'information $d_k=i$ quand son état est $S_k=m$. La variable $S_s^i(m)$ correspond à l'état du codeur auquel on arriverait, si partant de l'état m, on transmettait le bit i. Quant à $S_p^i(m)$, il correspond à l'état du codeur qui mènerait à l'état m si le bit j était transmis.

La mise en oeuvre de l'algorithme que nous venons de décrire peut alors se faire en quatre étapes.

- **Etape 1**: Initialiser les probabilités suivantes pour $i = 0, 1$:

  $\alpha_0^i(0) = 1$ et $\alpha_0^i(m) = 0$, pour tout $m \neq 0$.

  $\beta_N^i(0) = 1$ et $\beta_N^i(m) = 0$, pour tout $m \neq 0$.

- **Etape 2**: Pour tout $R_k=(x_k, y_k)$ reçu, calculer et sauvegarder les variables

  $$\delta_k^i(m) = exp\left(\frac{x_k i + y_k Y_k(i, m)}{\sigma^2/2}\right) \tag{3.36}$$

- **Etape 3**: Pour toutes les observations R_k, pour $i=0, 1$, et pour tous les états m, calculer les probabilités suivantes

  $$\beta_k^i(m) = \sum_{j=0}^{1} \beta_{k+1}^j(S_s^i(m))\,\delta_{k+1}^j(m) \tag{3.37}$$

• **Etape 4:** Pour toutes les observations R_k, $i=0, 1$, et tous les états m, calculer $\alpha_k^i(m)$ puis évaluer le LRV:

$$\alpha_k^i(m) = \delta_k^i(m) \sum_{j=0}^{1} \alpha_{k-1}^i(S_p^j(m)) \tag{3.38}$$

$$L(d_k) = ln \frac{\sum_m \alpha_k^1(m)\,\beta_k^1(m)}{\sum_m \alpha_k^0(m)\,\beta_k^0(m)} \tag{3.39}$$

A partir de l'équation (3.36), nous constatons que toutes les probabilités calculées dans l'algorithme MAP dépendent de σ^2. Le décodage suppose donc une parfaite connaissance du canal par le récepteur.

Afin de démontrer à quel point les formules utilisées dans l'algorithme proposé ont été simplifiées, nous présentons ci-dessous les équations obtenues dans [7]:

$$\alpha_k^i(m) = \frac{\sum_n \sum_{j=0}^{1} \alpha_{k-1}^j(n)\,\gamma_i(R_k, n, m)}{\sum_m \sum_n \sum_{i=0}^{1}\sum_{j=0}^{1} \alpha_{k-1}^j(n)\,\gamma_i(R_k, n, m)}$$

$$\beta_k(m) = \frac{\sum_n \sum_{j=0}^{1} \beta_{k+1}(n)\,\gamma_i(R_{k+1}, m, n)}{\sum_m \sum_n \sum_{i=0}^{1}\sum_{j=0}^{1} \alpha_{k-1}^j(n)\,\gamma_i(R_k, n, m)}$$

Il est alors évident que la seule modification dans la définition des variables α et β a permis de simplifier les calculs de façon substantielle, menant à des simulations plus rapides ainsi qu'à une éventuelle réalisation matérielle plus simple.

Malgré les simplifications apportées à l'algorithme présenté, celui-ci nécessite toujours beaucoup d'espace mémoire ainsi que des calculs excessifs. Certes, le remplacement de certains calculs répétitifs, tels que ceux des variables $S_p^i(m)$ et $S_s^i(m)$, par des opérations de lecture dans des tableaux allège quelque peu l'algorithme; toutefois, le nombre d'opérations complexes telles que les divisions et multiplications réelles, ou l'évaluation de logarithmes et d'exponentielles, reste encore trop élevé. D'autres simplifications sont donc nécessaires afin de permettre l'implémentation matérielle de l'algorithme.

3.3 Simplification de l'algorithme MAP

3.3.1 L'algorithme Log-MAP

Afin de réduire la complexité du décodage, on se propose d'éliminer les opérations de multiplication effectuées par l'algorithme MAP, en prenant le logarithme de tout l'algorithme. Utilisée pour la première fois pour des canaux à interférences inter-symboles, cette technique a été appliquée par la suite aux canaux codés [6,20,31,35]. Ainsi, toutes les multiplications de l'algorithme sont transformées en additions; ces dernières étant plus rapides à effectuer en simulation, et surtout plus simples à implémenter.

Pour ce faire, nous définissons une nouvelle fonction, E, de la façon suivante:

$$xEy = -ln\left(e^{-x} + e^{-y}\right) \tag{3.40}$$

$$\underset{m}{E} f(m) = -ln \sum_{m} exp\left(-f(m)\right) \tag{3.41}$$

Posons

$$A_k^i = -ln\,\alpha_k^i(m) \tag{3.42}$$

$$B_k^i = -ln\,\beta_k^i(m) \tag{3.43}$$

L'équation (3.35) peut alors s'écrire comme suit:

$$L(d_k) = ln\sum_m \alpha_k^1(m)\,\beta_k^1(m) - ln\sum_m \alpha_k^0(m)\,\beta_k^0(m)$$

$$= ln\sum_m exp\,(-A_k^1(m) - B_k^1(m)) - ln\sum_m exp\,(-A_k^0(m) - B_k^0(m))$$

$$= \mathop{\mathrm{E}}_m(A_k^0(m) + B_k^0(m)) - \mathop{\mathrm{E}}_m(A_k^1(m) + B_k^1(m)), \qquad (3.44)$$

et les équations (3.33) et (3.34) deviennent respectivement

$$A_k^i(m) = D_k^i(m) + \mathop{\mathrm{E}}_{j=0}^{I} A_{k-1}^j(S_p^j(m)) \qquad (3.45)$$

$$B_k^i(m) = \mathop{\mathrm{E}}_{j=0}^{I} B_{k+1}^j(S_s^i(m)) + D_{k+1}^j(S_s^i(m)) \qquad (3.46)$$

où $$D_k^i(m) = \frac{2}{\sigma^2}(x_k i + y_k Y_k(i,m)) \qquad (3.47)$$

Utilisant ces nouvelles définitions, les étapes de l'algorithme deviennent:

- **Etape 1**: Initialisation: pour $i = 0, 1$

  $A_0^i(0) = 0$ et $A_0^i(m) = -\infty$, pour tout $m \neq 0$.

  $B_N^i(0) = 0$ et $B_N^i(m) = -\infty$, pour tout $m \neq 0$.

- **Etape 2**: Pour tout R_k reçu, calculer et sauvegarder pour $i=0, 1$ et $m=0,..,$M-1

  $$D_k^i(m) = \frac{2}{\sigma^2}(x_k i + y_k Y_k(i,m)) \qquad (3.48)$$

- **Etape 3**: Pour toutes les observations R_k, pour $i=0,1$, et pour tous les états m, calculer

  $$B_k^i(m) = \mathop{\mathrm{E}}_{j=0}^{I}(B_{k+1}^j(S_s^i(m)) + D_{k+1}^j(S_s^i(m))) \qquad (3.49)$$

• **Etape 4:** Pour toutes les observations R_k, $i=0, 1$, et tous les états m, calculer $A_k^i(m)$ puis évaluer le LRV:

$$A_k^i(m) = D_k^i(m) + \underset{j=0}{\overset{1}{\mathrm{E}}} A_{k-1}^j(S_p^j(m)) \tag{3.50}$$

$$L(d_k) = \underset{m}{\mathrm{E}}(A_k^0(m) + B_k^0(m)) - \underset{m}{\mathrm{E}}(A_k^1(m) + B_k^1(m)) \tag{3.51}$$

Au niveau de l'étape 2, nous pouvons remarquer que puisque i et $Y_k(i, m)$ sont toutes deux des valeurs binaires, alors le nombre de paires $(i, Y_k(i, m))$, et donc le nombre de $D_k^i(m)$ calculés, est égal à quatre quelque soit la longueur de contrainte du codeur utilisé.

En passant au domaine logarithmique, toutes les multiplications et divisions de l'algorithme ont été remplacées par des additions et des évaluations de la fonction E. Si nous réexaminons maintenant de plus près la fonction E, nous constatons qu'elle peut être réécrite comme suit:

$$\begin{aligned} x\,E\,y &= -\ln(e^{-x}(1 + e^{x-y})) \\ &= x - \ln(1 + e^{x-y}) \\ &= y - \ln(1 + e^{y-x}) \\ &= \min(x, y) - \ln(1 + e^{-|y-x|}). \end{aligned} \tag{3.52}$$

Par ailleurs, soit h la fonction définie par $h(z) = \ln(1 + e^{-z})$. L'équation (3.50) par exemple devient

$$\begin{aligned} A_k^i(m) = D_k^i(m) + \; & \min(A_{k-1}^0(S_p^0(m)), A_{k-1}^1(S_p^1(m))) \\ & - h(abs(A_{k-1}^0(S_p^0(m)) - A_{k-1}^1(S_p^1(m)))) \end{aligned}$$

où *abs* représente la valeur absolue. La fonction h décroît très rapidement vers 0 (*e.g:* $h(10) = 4{,}5.10^{-5}$, $h(20) = 2{,}0.10^{-9}$). Un nombre fini de valeurs pré-calculées suffit

donc pour couvrir sans erreur importante toute la plage sur laquelle $h(z)$ est non nulle. Ainsi, l'évaluation de la fonction E peut se ramener à une comparaison de deux valeurs et une lecture dans la mémoire. La complexité de l'algorithme log-MAP est estimée à quatre fois celle de l'algorithme de Viterbi [31].
Une simplification supplémentaire, liée à la fonction h, conduit à une version sous-optimale de l'algorithme MAP, comme nous la montrons ci-après.

3.3.2 L'algorithme MAP sous optimal

Posons $h(z) = 0$. Les équations (3.44), (3.45) et (3.46) deviennent alors

$$L(d_k) = \min_m (A_k^0(m) + B_k^0(m)) - \min_m (A_k^1(m) + B_k^1(m)) \qquad (3.53)$$

$$A_k^i(m) = D_k^i(m) + \min (A_{k-1}^0(S_p^0(m)), A_{k-1}^1(S_p^1(m))) \qquad (3.54)$$

$$B_k^i(m) = \min_j (B_{k+1}^j(S_s^i(m)) + D_{k+1}^j(S_s^i(m))) \qquad (3.55)$$

Les performances obtenues avec l'algorithme MAP sous-optimal sont comparables à celles de l'algorithme de Viterbi avec quantification dure [35], avec la différence, bien sûr, liée à la génération d'une information de fiabilité. Les équations (3.54) et (3.55) rappellent d'ailleurs la fameuse opération *ajouter-comparer-sélectionner* utilisée dans l'algorithme de Viterbi. Cependant, contrairement à ce dernier, deux boucles récursives (croissante et décroissante) restent nécessaires pour l'algorithme MAP sous-optimal.

3.4 Comparaison des algorithmes MAP et SOVA

L'algorithme SOVA (*Soft Output Viterbi Algorithm*) est, comme son nom l'indique, une version modifiée de l'algorithme de Viterbi, qui permet la génération d'une information mesurant le degré de fiabilité de la décision prise lors du décodage des symbo-

les reçus. Le principal avantage de SOVA par rapport à MAP est qu'il ne nécessite qu'une seule boucle récursive pour le calcul des probabilités d'erreur, de même qu'il ne requiert aucun stockage des séquences reçues. L'implémentation de SOVA serait donc beaucoup plus simple que celle de MAP.

Une comparaison des algorithmes MAP et SOVA a été menée dans [31]. La conclusion à laquelle les auteurs sont parvenus est que les probabilités générées par SOVA étaient basées sur une approximation intuitive, plutôt biaisée, qui conduisait pour la plupart des bits décodés à une sous-estimation de la probabilité d'avoir pris une bonne décision. Il a été constaté également que l'estimation de ces probabilités était d'autant plus biaisée que le milieu était bruité, donc pour des TEB élevés.

Une autre remarque intéressante porte sur le fait que la variation du niveau des sorties générées par MAP est beaucoup plus importante que celle observée sur les sorties de SOVA. Cette plus grande "sensibilité" de l'algorithme MAP permet d'obtenir de meilleurs rapports de vraisemblance, ce qui est très avantageux dans un processus itératif.

SOVA constitue ainsi une autre alternative pour le décodage turbo. De complexité plus faible, cet algorithme présente des performances légèrement inférieures à celles de l'algorithme MAP. Les propriétés statistiques des sorties qu'il génère sont cependant moins bonnes que dans le cas de MAP, ce qui fait que l'écart entre les deux algorithme se creuse assez rapidement lorsqu'ils sont utilisés dans un système récursif, surtout pour de faibles RSB. Une comparaison quantitative des performances des deux algorithmes est faite au paragraphe suivant.

3.5 Simulations et résultats

Pour nos simulations, nous avons utilisé l'algorithme log-MAP, en prenant pour domaine de définition de la fonction h l'intervalle [0, 16] quantifié en 2^{16} niveaux (65536 valeurs). Ces valeurs ne sont calculées qu'une fois, puis stockées dans un

tableau. Lors des simulations, nous avons pu constater que les métriques α et β étaient sujettes à des fluctuations importantes, pouvant les rendre parfois trop petites ou trop grandes. Afin de palier ce problème, nous avons procédé à une normalisation de ces probabilités en divisant les valeurs obtenues par une valeur moyenne [31]. Par exemple, pour la métrique α ,

$$\alpha_{moy} = exp\,((\log\alpha_{min} + \log\alpha_{max})/2), \tag{3.56}$$

ce qui se ramène dans le domaine logarithmique à une soustraction de la valeur moyenne

$$A_{moy} = (A_{min} + A_{max})/2. \tag{3.57}$$

Cette normalisation n'affecte en rien le résultat final, puisqu'une division est effectuée pour le calcul du rapport de vraisemblance dans l'équation (3.39).

La modification des définitions des probabilités $\alpha_k^i(m)$ et $\beta_k^i(m)$ dans les équations (3.8) et (3.9) respectivement, ajoutée au passage au domaine logarithmique, avec toutes les simplifications que cela implique, ont permis d'accélérer considérablement les simulations par rapport à la version de l'algorithme proposée dans [7].

Dans le tableau 3.1 nous avons rapporté un exemple de résultats obtenus par simulation du code convolutionnel systématique et récursif à 4 états, de taux R=1/2, décodé par les algorithmes MAP et SOVA.

Tableau 3.1 TEB pour R=1/2, K=3, G=(1, 5/7), canal gaussien.

E_b/N_0 (dB)	0	1	2	3	4	5
MAP	$8{,}0.10^{-2}$	$3{,}1.10^{-2}$	$9{,}1.10^{-3}$	$1{,}6.10^{-3}$	$5{,}4.10^{-4}$	$7{,}6.10^{-5}$
SOVA	$8{,}3.10^{-2}$	$3{,}3.10^{-2}$	$9{,}1.10^{-3}$	$1{,}7.10^{-3}$	$5{,}4.10^{-4}$	$7{,}7.10^{-5}$

Nous constatons alors que, tout en restant très faible, l'écart entre les performances des deux algorithmes est d'autant plus important que le RSB est faible. Cette différence, aussi petite soit-elle, est cependant importante lorsqu'il s'agit de décoder de façon itérative.

3.6 Conclusion

Nous avons présenté dans ce chapitre la dérivation de l'algorithme MAP (*Maximum A Posteriori*); algorithme qui répond bien aux critères requis pour le décodage turbo. Il permet en effet de minimiser la probabilité d'erreur par bit, et de générer pour chaque bit décodé une information de fiabilité, mesurant la probabilité d'avoir pris ou non la bonne décision lors du décodage.

Le noyau de l'algorithme MAP est constitué par deux boucles récursives, croissante et décroissante. Le décodage se faisant par bloc, il ne peut commencer, à cause de la boucle décroissante, qu'après la réception complète d'un bloc. Une latence, égale au temps nécessaire pour le décodage d'un bloc, est ainsi introduite.

Deux simplifications majeures ont été apportées à l'algorithme MAP. La première, par S. Pietrobon [31], a porté sur la modification de la définition des métriques d'état et de transition. Cette simplification a permis d'alléger les calculs de façon considérable par rapport à la version originale de l'algorithme. La deuxième simplification quant à elle, a consisté à exprimer l'algorithme dans le domaine logarithmique, remplaçant de la sorte des opérations complexes, telles que les multiplications réelles et l'évaluation d'exponentielles, par des opérations d'addition plus simples. Les performances de la version log-MAP résultant de ces deux modifications sont quasi identiques à celles de la version proposée par Berrou *et al.* dans [7].

La comparaison des algorithmes SOVA et MAP a démontré que malgré la plus grande complexité de ce dernier, évaluée à environ quatre fois celle de l'algorithme de

Viterbi, l'algorithme MAP reste plus attrayant pour l'utilisation dans un processus de décodage itératif. Et ce, grâce essentiellement à la plus grande variation du niveau des sorties qu'il génère, résultant en un gain global de près de 1dB par rapport à SOVA.

Ainsi donc, la principal avantage que présente l'algorithme MAP consiste en une évaluation précise des LRV qu'il génère, dont les valeurs varient de façon avantageuse pour le décodage itératif. Quant à ses inconvénients, ils sont au nombre de trois. D'abord, une complexité qui reste relativement élevée malgré les simplifications apportées. Ensuite, l'introduction d'un délai égal au temps de décodage d'un bloc, qui peut éliminer certains domaines d'application quand la dimension des blocs est assez grande. Le dernier inconvénient enfin, consiste en la nécessité d'évaluer la variance du bruit dans le canal.

Le choix de l'algorithme de décodage constitue un élément déterminant dans l'obtention d'une certaine gamme de performances des codes turbo. Nous avons quant à nous opté pour un haut niveau de performances, au prix d'une plus grande complexité. Dans la suite, nous utiliserons donc exclusivement l'algorithme MAP, et plus particulièrement la version log-MAP.

Chapitre 4

DÉCODAGE ITÉRATIF PAR L'ALGORITHME MAP: LE DÉCODEUR TURBO

4.1 Introduction

Après la description de l'algorithme MAP au chapitre 3, où nous avons démontré les avantages qu'il présente, notamment dans un système itératif, nous présentons dans ce chapitre l'adaptation de cet algorithme au décodage turbo.

Présenté pour la première fois dans [7], le décodeur turbo est un système de décodage modulaire. Il appartient de ce fait à une catégorie de décodeurs dont la principale caractéristique est que le décodage se fait de façon séquentielle, en utilisant à chaque étape un module de décodage correspondant à un code "partiel" donné. L'objectif de cette procédure est de remplacer le décodage à maximum de vraisemblance, dont la complexité croît exponentiellement avec la longueur des codes, par un décodage modulaire plus simple. Dans cette même catégorie, nous retrouvons les codes produit [14], les codes concaténés (en série) [15], et les codes multi-niveaux [21].

La principale innovation apportée par les codes turbo par rapport à leurs prédécesseurs est que, contrairement à ces derniers, de l'information est échangée entre les deux modules de décodage. Ceci a été rendu possible grâce au fait que les deux codeurs constituant le codeur turbo codent deux fois la même séquence d'information.

Dans ce chapitre, nous commençons par présenter le principe du décodage itératif. Nous procédons ensuite à l'examen des sorties générées par l'algorithme MAP, afin de les adapter au fonctionnement du décodeur turbo. Nous démontrons aussi l'impor-

tance de la contribution de l'entrelaceur dans l'obtention des performances exceptionnelles des codes turbo. Nous fournissons enfin un exemple des performances obtenues par le décodage turbo utilisant l'algorithme MAP. Une analyse plus détaillée des performances en fonction des différents paramètres du système sera faite au chapitre 5.

4.2 Principe du décodage itératif

Lors de la description de la structure du système au chapitre 2, nous avons montré que les symboles fournis par le canal au décodeur turbo sont de trois types: les symboles systématiques, communs aux deux codeurs, les symboles de parité générés par le premier codeur, et les symboles de parité générés par le deuxième codeur; tous ces symboles correspondant au même ensemble de bits d'information.

Le principe du décodage itératif des codes turbo est illustré par la figure 4.1.

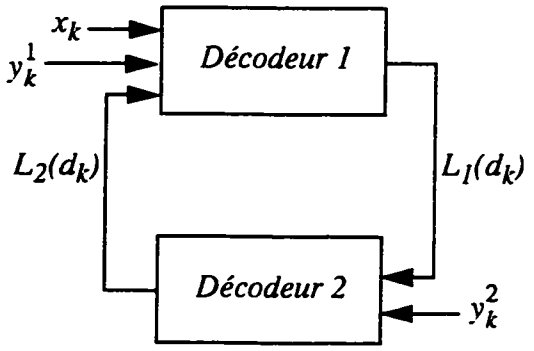


Figure 4.1 Principe du décodage itératif

La technique de décodage utilisée par le décodeur turbo est assez similaire à celle des codes produit. En effet, à partir des séquences $\{x_k\}$ et $\{y_k^1\}$, Décodeur 1 décode le bloc reçu "dans une dimension", celle de Codeur 1, dont il reçoit les symboles de parité. Le bloc est alors passé à Décodeur 2 qui, à son tour, le décode dans la deuxième dimension, celle de Codeur 2. Des erreurs qui n'ont pas été corrigées par

Décodeur 1, pourraient ainsi l'être grâce à Décodeur 2. Après le passage par celui-ci, des salves d'erreurs qui n'auraient pu être éliminées par Décodeur 1 au premier passage, pourraient l'être à présent, puisque Décodeur 2 peut en avoir corrigé une partie.

Sur le schéma de la figure 4.1, et dans un dessein de simplification, nous avons omis de représenter l'entrelaceur séparant la sortie de Décodeur 1 de l'entrée de Décodeur 2; de même que le délaceur de l'autre côté de la boucle. Grâce à la présence de ces deux éléments, les salves d'erreurs qui n'ont pu être corrigées par l'un des deux décodeurs sont dispersées, augmentant ainsi les chances qu'elles le soient par l'autre décodeur. La réitération de cette procédure de décodage permet alors d'améliorer considérablement les performances obtenues. Ainsi, en plus de permettre le réarrangement des séquences échangées entre les deux décodeurs afin qu'elles concordent avec les séquences reçues du canal, l'entrelaceur permet aussi de disperser les paquets de bruit, facilitant de la sorte leur correction.

Considérons à présent l'information échangée entre les deux décodeurs, et notons par L_1 et L_2 les LRV générés respectivement par Décodeur 1 et Décodeur 2. Supposons que les LRV générés par l'un des décodeurs MAP soient entièrement transmis vers l'autre. A la première itération, Décodeur 1 utiliserait les séquences $\{x_k\}$ et $\{y_k^1\}$ pour calculer $\{L_1(d_k)\}$. Cette dernière séquence serait donc fonction de $\{x_k\}$ et de $\{y_k^1\}$. De même, $\{L_2(d_k)\}$ serait fonction de $\{y_k^2\}$ et de $\{L_1(d_k)\}$, donc aussi de $\{x_k\}$ et $\{y_k^1\}$. A la deuxième itération, $\{L_2(d_k)\}$ viendrait s'ajouter aux entrées de Décodeur 1, résultant en une corrélation entre celles-ci. Par conséquent, avant la transmission des LRV générés par un décodeur vers l'entrée de l'autre (à travers l'entrelaceur ou le délaceur), nous devrons en éliminer toute information qui serait corrélée avec les entrées du décodeur qui les reçoit.

Seule une partie des LRV, appelée *information extrinsèque*, sera transmise d'un décodeur à l'autre. Au paragraphe suivant, nous démontrons comment extraire cette information de $L(d_k)$, ainsi que le rôle que joue encore une fois l'entrelaceur dans cette opération.

4.3 Le décodeur turbo

4.3.1 Information extrinsèque

Considérons le décodeur MAP de la figure 4.2. Ses entrées sont constituées par les séquences $\{x_k\}$ et $\{y_k\}$, et à sa sortie, les LRV $\{L(d_k)\}$ sont générés.

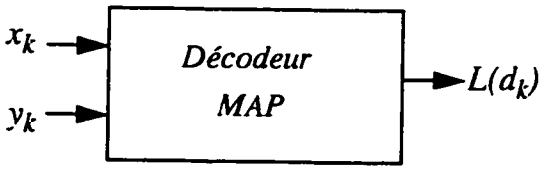


Figure 4.2 Schéma de principe du décodeur MAP

Ces LRV sont calculés à partir des probabilités à posteriori

$$L(d_k) = ln\frac{Pr(d_k = 1|R_1, ..., R_N)}{Pr(d_k = -1|R_1, ..., R_N)} \quad . \tag{4.1}$$

Si les observations $(R_1,..., R_N)$ sont indépendantes, (4.1) peut s'écrire sous la forme

$$L(d_k) = ln\frac{Pr(d_k = 1|x_k)}{Pr(d_k = -1|x_k)}$$

$$+ ln\frac{Pr(d_k = 1|x_1, ..., x_{k-1}, x_{k+1}, ..., x_N, y_1, ..., y_N)}{Pr(d_k = 1|x_1, ..., x_{k-1}, x_{k+1}, ..., x_N, y_1, ..., y_N)} \tag{4.2}$$

Notons par W_k le deuxième terme de l'addition dans l'équation (4.2). Celle-ci devient alors

$$L(d_k) = ln\frac{Pr(d_k = 1|x_k)}{Pr(d_k = -1|x_k)} + W_k$$

$$= ln\frac{Pr(x_k|d_k = 1)}{Pr(x_k|d_k = -1)} + ln\frac{Pr(d_k = 1)}{Pr(d_k = -1)} + W_k \tag{4.3}$$

Comme le bit d'information d_k prend les valeurs +1 et -1 avec égales probabilités, alors

$$ln\frac{Pr(d_k = 1)}{Pr(d_k = -1)} = 0$$

et (4.3) devient

$$L(d_k) = ln\frac{Pr(x_k| d_k = 1)}{Pr(x_k| d_k = -1)} + W_k. \qquad (4.4)$$

Par ailleurs, nous avons démontré au chapitre 3 que

$$Pr(x_k) = \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{1}{2\sigma^2}(x_k - (2d_k - 1))^2\right) dx, \qquad (4.5)$$

avec $\sigma^2 = \left(2R\frac{E_b}{N_0}\right)^{-1}$

d'où $Pr(x_k| d_k = i) = \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{1}{2\sigma^2}(x_k - (2i - 1))^2\right) dx, \quad i=0, 1 \qquad (4.6)$

et l'équation (4.4) devient

$$L(d_k) = \frac{2}{\sigma^2}x_k + W_k. \qquad (4.7)$$

La sortie du décodeur MAP a été ainsi décomposée en une somme de deux termes. Le premier terme est proportionnel à x_k, tandis que le deuxième, W_k, est calculé à partir des deux séquences d'entrée, et constitue en quelque sorte la part de décision du décodeur. W_k est appelée information extrinsèque. Cette information est généralement du même signe que x_k [7] (nous l'avons vérifié par simulation), permettant de ce fait de "renforcer" le symbole systématique x_k en augmentant sa puissance apparente.

4.3.2 L'algorithme itératif

Après avoir vu comment la sortie d'un décodeur MAP pouvait être décomposée en une somme, dont l'un des termes n'est autre que l'information extrinsèque, nous détenons maintenant tous les éléments nécessaires à la compréhension du fonctionnement du décodeur (turbo) itératif. Il s'agira essentiellement de transmettre correctement l'information générée par l'un des décodeurs vers l'entrée de l'autre. Cet échange d'information entre les décodeurs, qui constitue la clé du fonctionnement du décodeur turbo, est souvent passé sous silence dans la plupart des publications. Nous essayerons dans ce paragraphe de fournir une description détaillée de cette procédure.

Plaçons-nous pour commencer au niveau du premier décodeur (Décodeur 1), et considérons la première itération. Les séquences disponibles à l'entrée sont alors $2/\sigma^2\{x_k\}$ et $2/\sigma^2\{y_k\}$. La multiplication par le facteur $2/\sigma^2$ remplace en fait la division par $\sigma^2/2$ dans l'équation (3.36). D'après l'équation (4.7), la sortie générée par Décodeur 1 serait alors

$$L_1(d_k) = \frac{2}{\sigma^2}x_k + W_k^1 \tag{4.8}$$

où W_k^1 est l'information extrinsèque calculée par Décodeur 1. $L_1(d_k)$ étant calculé uniquement à partir des symboles reçus du canal, est entièrement passé à Décodeur 2 tel que représenté sur la figure 4.3. W_n^1 est alors utilisée comme information à priori, ce qui permet à Décodeur 2 de fonctionner en quelque sorte à un RSB plus élevé, et de pouvoir par conséquent corriger plus d'erreurs.

Figure 4.3 Schéma de principe de Décodeur 2, à la première itération du décodage

Sur la figure 4.3, nous avons remplacé l'indice temporel k par l'indice n, puisque les séquences reçues par Décodeur 2 passent d'abord par un entrelaceur. Si nous remplaçons maintenant x_k dans l'équation (4.7) par $(x_n + (\sigma^2/2)\, W_n^1)$, nous obtenons les LRV générés par Décodeur 2:

$$L_2(d_n) = \frac{2}{\sigma^2} x_n + W_n^1 + W_n^2. \tag{4.9}$$

Au terme de cette première itération, $\{L_2(d_n)\}$ peut être comparée à un seuil nul afin de fournir une estimation $\{\hat{d}_n\}$ de la séquence de bits d'information. Mais cela ne sera sûrement pas effectué avant que la procédure ne soit encore réitérée un certain nombre de fois. Il s'agit donc à présent de fournir à Décodeur 1 une information à priori pour chaque bit à décoder, afin qu'il puisse améliorer sa performance.

Dans l'équation (4.9), nous constatons que les deux premiers termes de la somme constituant $L_2(d_n)$ ont été en fait générés par Décodeur 1. Leur retransmission vers celui-ci provoquerait donc une corrélation des entrées, et l'équation (4.2) ne serait plus vérifiée. Par conséquent, seule l'information extrinsèque W_n^2, générée par Décodeur 2, serait, après avoir été délacée, passée à Décodeur 1. Ce dernier l'utiliserait alors comme information à priori, en l'additionnant aux symboles d'information reçus du canal. A partir de la deuxième itération, les LRV générés par Décodeur 1 seraient alors de la forme

$$L_1(d_k) = \frac{2}{\sigma^2} x_k + W_k^1 + W_k^2. \tag{4.10}$$

De ces valeurs, il faudrait soustraire l'information provenant de Décodeur 2, en l'occurrence W_k^2, avant de retransmettre vers ce dernier, via l'entrelaceur, la nouvelle séquence d'information à priori.

Sur la figure 4.4, nous avons représenté l'étage correspondant à une itération du décodage turbo. Le décodeur turbo serait alors formé par une cascade de ces étages, dont le nombre serait égal au nombre d'itérations. Les sorties générées à la fin d'une

itération, exceptée la séquence $\{\hat{d}_n\}$, sont directement utilisées comme entrées à l'itération suivante. L'estimation des bits d'information $\{\hat{d}_n\}$ est effectuée à la fin de la dernière itération. Les délais D_{MAP} et D_{iter} correspondent respectivement au temps nécessaire au décodeur MAP pour le décodage d'un bloc, et à la durée totale d'une itération. Le coefficient A est égal à $2/\sigma^2$.

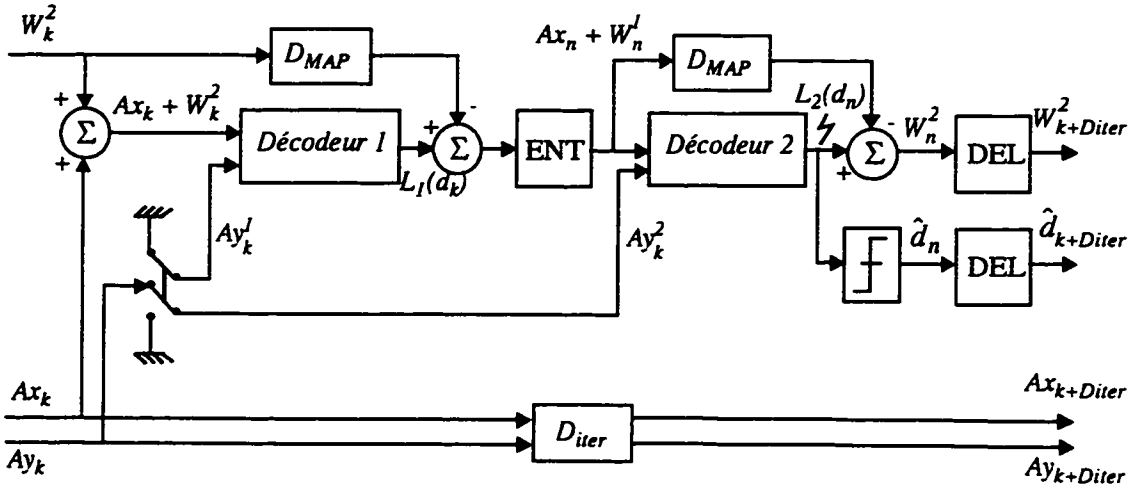


Figure 4.4 Schéma de principe du décodeur (turbo) itératif

A cause de la latence D_{MAP} introduite par Décodeur 1, les entrées de Décodeur 2 risquent de ne plus être synchronisées, du moins à la première itération. Nous pouvons donc comprendre maintenant la nécessité du module de retard placé à l'entrée de Codeur 2, et qui introduirait un délai égal à D_{MAP}.

4.3.3 Décorrélation des entrées du décodeur MAP

Jusque là, nous avons présumé qu'en ne transférant que l'information extrinsèque d'un décodeur à l'autre, nous ne provoquions aucune corrélation au niveau des entrées du décodeur récepteur. Néanmoins, en réexaminant l'équation (4.2), nous constatons que W_n^2 est en fait fonction de tous les x_i, excepté x_n, de tous les W_n^1, et

de tous les y_n^2. Egalement, W_k^1 est fonction de tous les x_i, excepté x_k, de tous les W_k^2, et de tous les y_k^1. Par conséquent, même la seule transmission de l'information extrinsèque peut entraîner une faible corrélation. Nous allons cependant démontrer que, grâce encore une fois à l'entrelacement, cette faible corrélation peut devenir négligeable. Une démonstration analytique s'étant avérée très complexe, nous avons procédé par simulation.

Pour ce faire, nous avons considéré la sortie $L(d_k)$ d'un décodeur MAP, calculée pour un bit d_i donné à partir des séquences d'entrée $\{x_k\}$ et $\{y_k\}$. Nous avons ensuite évalué l'influence de chaque paire (x_k, y_k) de symboles reçus sur le calcul de $L(d_k)$. Nous avons alors recalculé le LRV en changeant à chaque fois le signe des symboles correspondant à l'un des bits du bloc transmis. Sur la figure 4.4, nous avons représenté les fluctuations relatives des LRV $L(d_0)$, $L(d_{200})$ et $L(d_{399})$ pour un bloc de 400 bits. Les mesures ont été effectuées pour un code à 16 états (K=5) et à un RSB égal à 0,5dB, donc pour une transmission fortement bruitée.

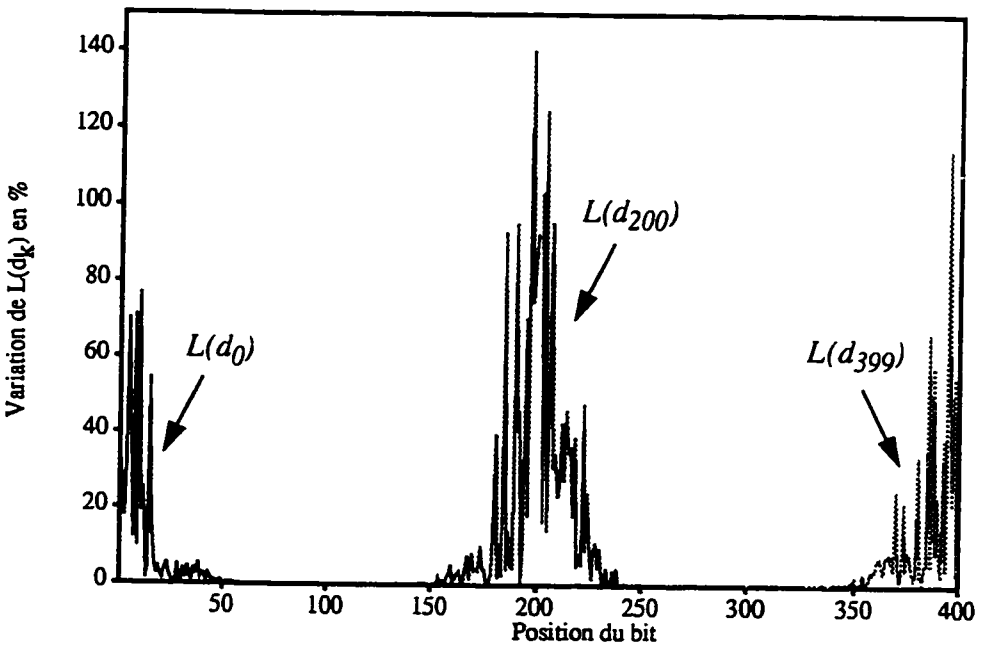


Figure 4.5 Influence des divers symboles reçus sur le calcul de $L(d_k)$

Tel que nous l'avions soupçonné, seuls les bits à l'intérieur d'une fenêtre centrée sur un bit d_i donné, influencent le calcul de $L(d_i)$. Nous avons répété ces simulations en variant les longueurs de blocs, les positions des bits dans un bloc et les RSB. Les résultats obtenus étaient quasi inchangés. Le seul paramètre qui semble influencer ces résultats, est la longueur du code utilisé. En effet, la largeur de la fenêtre contenant les symboles influents est de l'ordre de 20 fois la longueur de contrainte du code (100 bits pour K=5).

La figure 4.5 montre alors le rôle de décorrélation que joue l'entrelaceur lors de la transmission de l'information extrinsèque d'un décodeur à l'autre. Cette illustration correspond à un code de longueur de contrainte K=5.

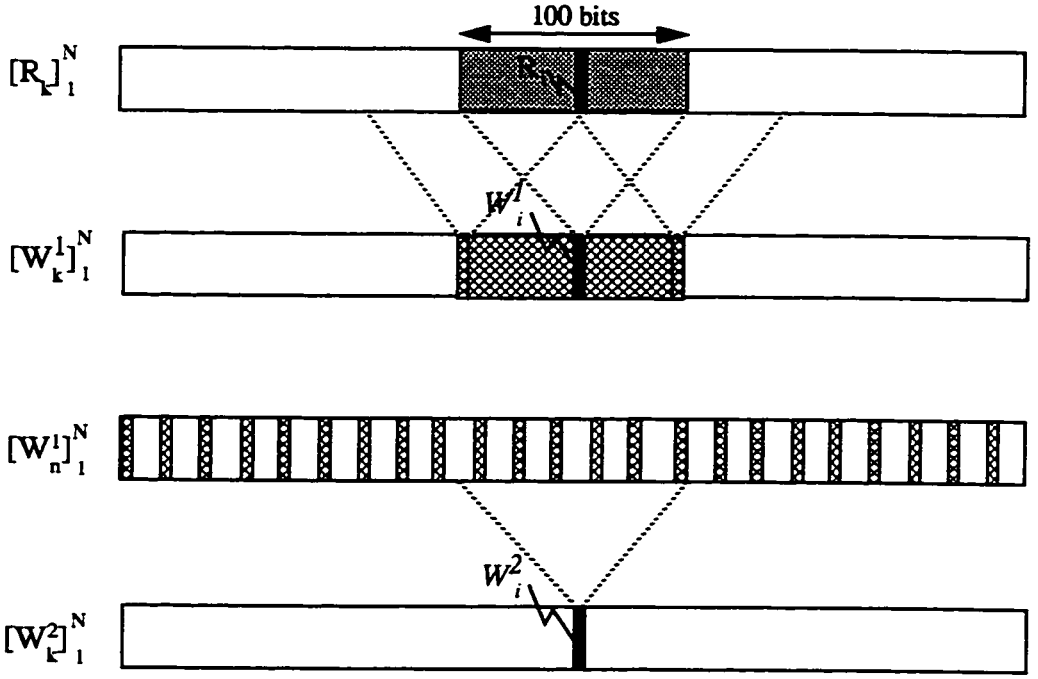


Figure 4.6 Décorrélation des entrées des décodeurs par l'entrelaceur

Soit $[R_k]_1^N$ la séquence reçue du canal par Décodeur 1, où $R_k=(x_k, y_k)$. On s'intéresse au décodage du bit d_i. Comme nous l'avons démontré plus haut, le calcul de W_i^1

serait influencé, dans le cas K=5, par la séquence $[R_k]_{i-50}^{i+50}$. Les informations extrinsèques générées par Décodeur 1 et influencées par R_i seraient alors $[W_k^1]_{i-50}^{i+50}$. Après entrelacement, ces dernières valeurs se trouveraient réparties sur tout le bloc, de sorte que W_i^2 ne serait que très peu corrélée avec R_i, et pourrait ainsi être utilisée comme information à priori à la prochaine itération.

Ainsi, il est clair que l'entrelaceur détient un autre rôle dans le fonctionnement du décodeur turbo. Cette fonction consiste à décorréler les entrées des décodeurs MAP, permettant ainsi de leur apporter à chaque itération une nouvelle information supplémentaire, ce qui explique en partie l'amélioration progressive des performances obtenues par décodage turbo. La question qui peut se poser à présent cependant est: quel type d'entrelaceur utiliser? La réponse à cette question sera apportée au chapitre 5. Nous présentons cependant pour commencer les différents types d'entrelaceurs qui peuvent être utilisés.

4.4 Les différents types d'entrelaceurs

La figure 4.7, où nous avons tracé le TEB en fonction du nombre d'itérations, nous donne une idée de l'allure qu'auraient les performances d'un code turbo sans entrelacement.

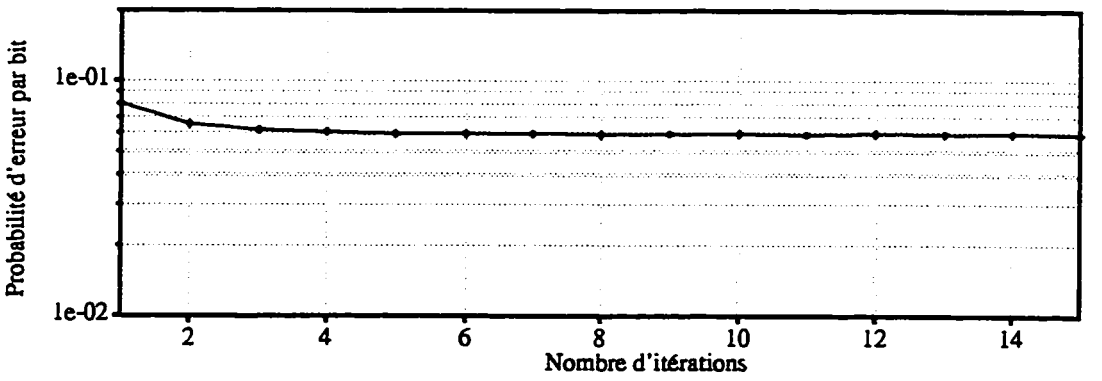


Figure 4.7 Allure des performances des codes turbo sans entrelacement

Ces résultats correspondent à la transmission de blocs de 3600 bits sur un canal gaussien à E_b/N_0=1,5dB. Les codes utilisés ont une longueur de contrainte K=3 avec les générateurs G=(1, 5/7). Comme nous pouvions le prévoir, nous constatons ainsi qu'en l'absence de tout entrelacement, l'augmentation du nombre d'itérations n'apporte aucune amélioration aux performances.

Les différents types d'entrelaceurs peuvent être subdivisés en deux classes: les entrelaceurs déterministes et les entrelaceurs pseudo-aléatoires.

4.4.1 Les entrelaceurs déterministes

A. L'entrelaceur bloc

Comme leur nom l'indique, les entrelaceurs déterministes constituent une catégorie d'entrelaceurs où, connaissant la position d'un bit dans un bloc, on peut calculer sa nouvelle position après entrelacement du bloc. L'entrelaceur déterministe le plus simple est l'entrelaceur bloc. Il s'agit simplement dans ce cas d'écrire dans l'ordre, ligne par ligne, les bits d'un bloc dans une matrice (idéalement carrée), et de les lire ensuite colonne par colonne. La figure 4.8 est une illustration de l'entrelacement de type bloc d'une séquence de 25 bits.

Si la matrice qui constitue l'entrelaceur est carrée de dimension (n x n), alors deux bits consécutifs dans un bloc seront séparés après entrelacement de n-1 bits. Ainsi, dès qu'un paquet d'erreurs est de longueur supérieure à n, la séparation des bits erronés ne sera jamais totale. Supposons par exemple que les bits 3 à 14 (soulignés sur la figure 4.8) aient été bruités lors de la transmission. Nous constatons alors qu'un entrelaceur bloc ne permet de disperser que partiellement cette salve d'erreurs, qui d'une longueur de 12 bits est passée à une longueur maximale de 3 bits, avec des salves de longueur 2 bits et des erreurs isolées. Par conséquent, nous pouvons déjà prévoir que l'augmentation du nombre d'itérations cessera rapidement d'apporter une amélioration aux performances des codes turbo.

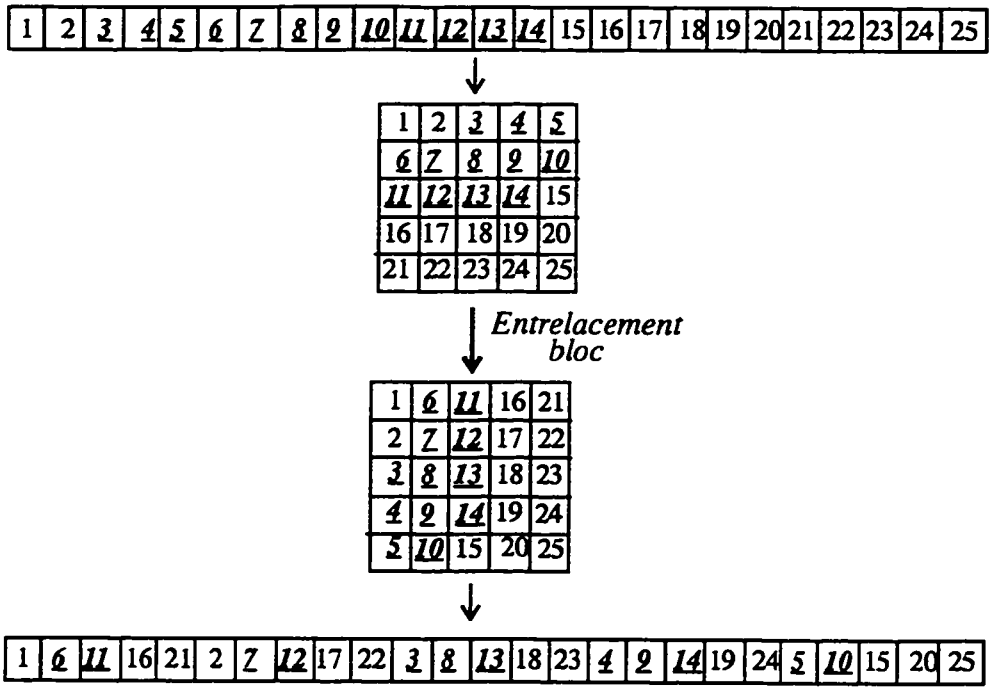


Figure 4.8 Principe de fonctionnement de l'entrelaceur bloc

B. L'entrelaceur hélicoïdal

L'entrelaceur hélicoïdal est un autre entrelaceur déterministe. Il fonctionne selon le même principe que l'entrelaceur bloc, à la différence que la lecture dans la matrice se fait non plus par colonne, mais suivant la diagonale. Les dimensions de la matrice sont de type n x $(n+1)$. La figure 4.9 illustre ce type d'entrelacement.

Comme l'entrelaceur bloc, nous voyons que l'entrelaceur hélicoïdal n'est pas non plus très efficace pour la dispersion des paquets d'erreur (bits 3 à 11, soulignés sur la figure 4.9). Il ne devrait donc pas donner de meilleurs résultats.

Sur la figure 4.11 nous avons joint aux performances obtenues sans entrelacement, celles obtenues avec les entrelaceurs bloc et hélicoïdal. Nous constatons alors d'abord, que les performances obtenues avec les deux types d'entrelaceurs détermi-

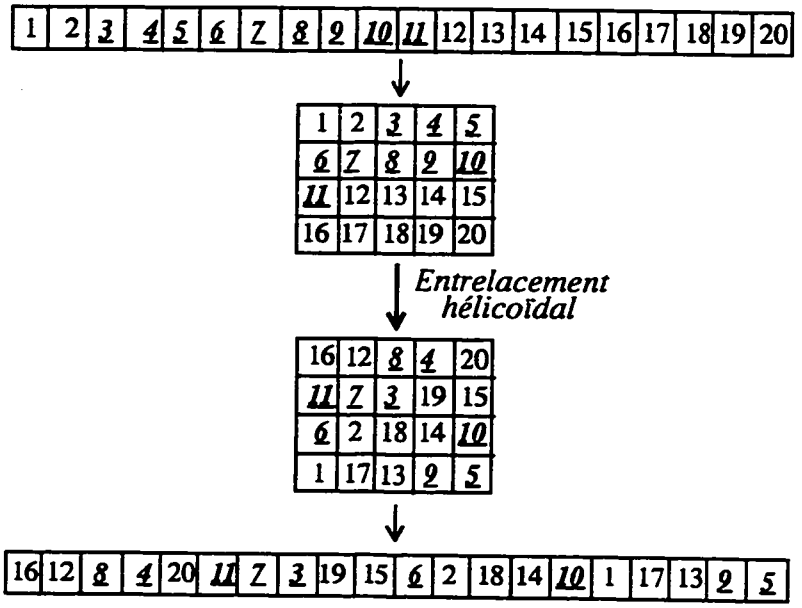


Figure 4.9 Principe de l'entrelacement hélicoïdal

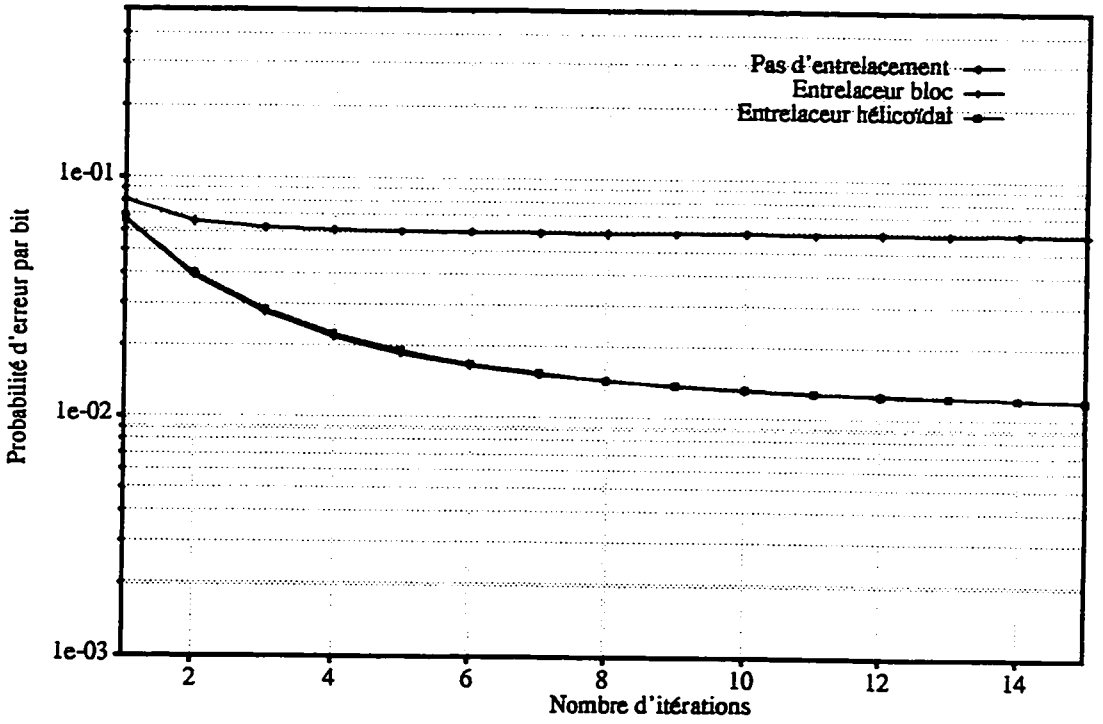


Figure 4.10 Performances obtenues avec les entrelaceurs bloc et hélicoïdal

nistes sont très semblables, même identiques dans le cas de notre exemple. Une amélioration est certes observée par rapport au cas sans entrelacement, toutefois à partir de la cinquième itération environ, toute itération supplémentaire devient inutile.

4.4.2 Les entrelaceurs pseudo-aléatoires

L'entrelacement aléatoire d'un bloc de longueur N est effectué de la façon suivante: pour chaque bit dans le bloc, un nombre aléatoire entre 1 et N est généré. Ce nombre correspond à la nouvelle position du bit dans le bloc entrelacé. Par conséquent, contrairement aux entrelaceurs déterministes, les positions des bits après entrelacement aléatoire ne peuvent être prévues à partir des positions avant entrelacement. Une table de correspondance doit donc être utilisée afin de permettre le délacement ultérieur des blocs.

Sur la figure 4.11, nous pouvons constater l'amélioration progressive des performances obtenues avec l'entrelacement aléatoire, ainsi que le TEB dix fois plus faible que celui atteint avec les entrelaceurs bloc et hélicoïdal après 15 itérations.

La différence entre les performances obtenues avec différents entrelaceurs, utilisant différentes séquences pseudo-aléatoires, est généralement très faible. Nous avons toutefois testé plusieurs entrelaceurs afin de nous assurer de ne pas tomber sur une trop mauvaise séquence, telle que celle qui transformerait simplement le bloc à entrelacer en lui-même.

Les entrelaceurs pseudo-aléatoires semblent ainsi être beaucoup plus avantageux à utiliser pour les codes turbo. Néanmoins, comme nous le verrons au chapitre 5, les entrelaceurs blocs peuvent constituer dans certains cas, notamment pour de faibles longueurs de blocs, une solution intéressante de par sa simplicité.

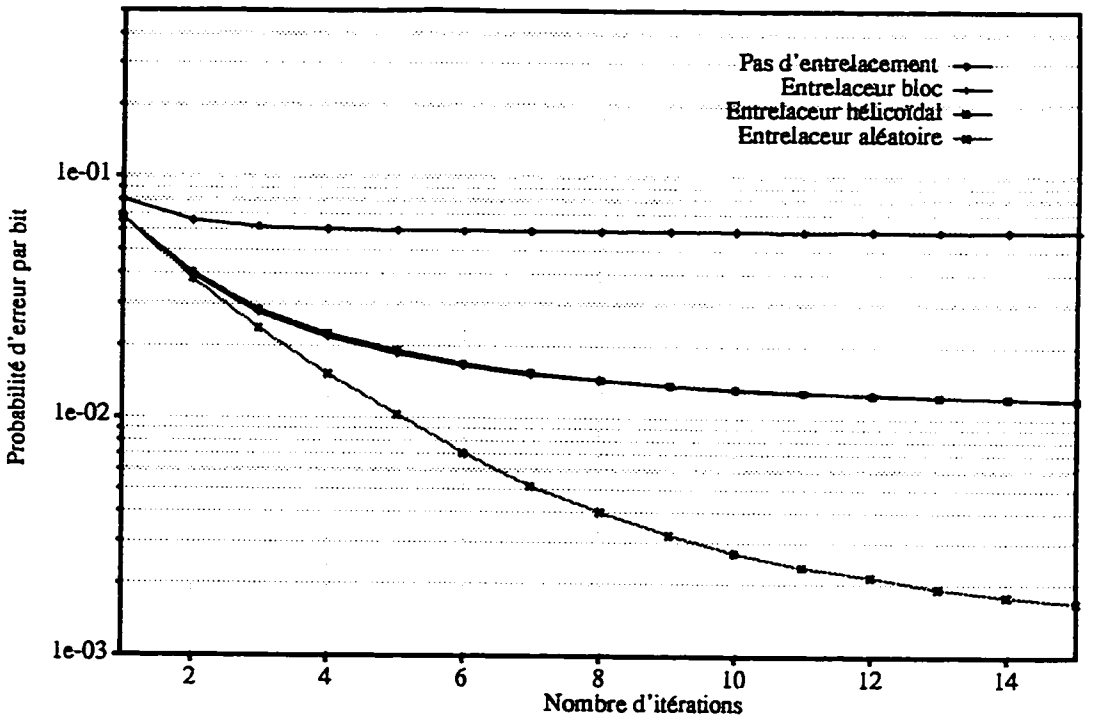


Figure 4.11 Amélioration des performances apportée par l'entrelacement aléatoire

4.5 Simulations

Nous avons rapporté sur la figure 4.12 un exemple des résultats obtenus par simulation des codes turbo. Le codeur est constitué par la concaténation de deux codes convolutionnels identiques, systématiques et récursifs, de taux R=1/2, de longueur de contrainte K=3, et avec les générateurs G=(1, 5/7). Le taux de codage global est aussi égal à 1/2. L'entrelaceur utilisé est de type pseudo-aléatoire, de longueur 3600 bits, et le décodage est effectué en 12 itérations.

Nous donnons ci-dessous, sans les commenter, un certain nombre de remarques que l'on peut faire à partir de la figure 4.12. Une analyse de ces points sera effectuée au chapitre 5.

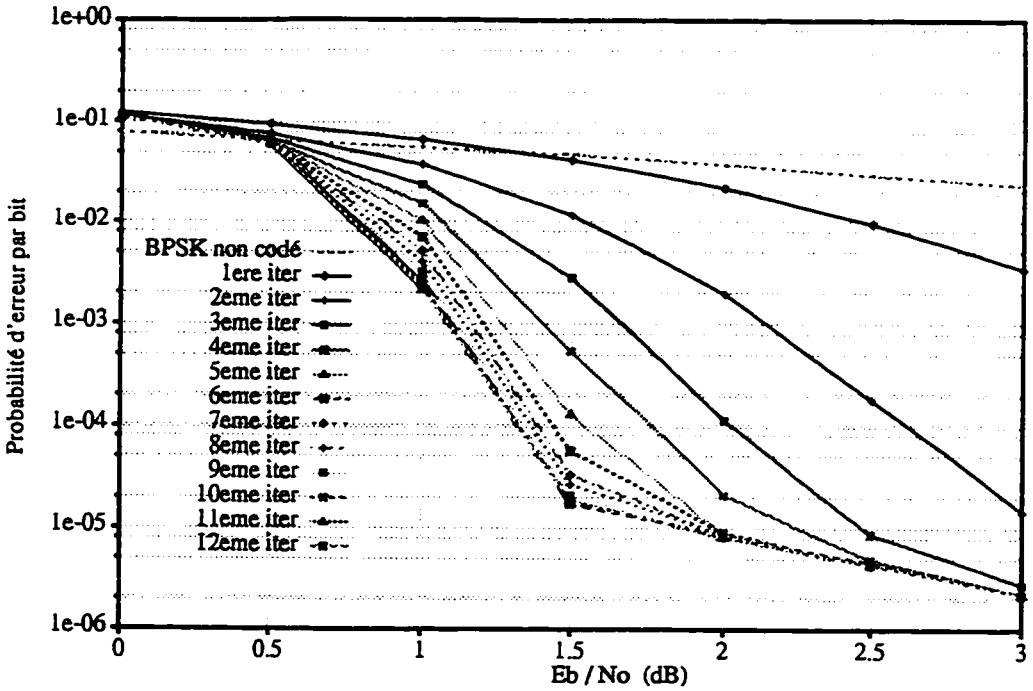


Figure 4.12 Performances du code turbo pour K=3, G=(1, 7/5); R_{global}=1/2; entrelaceur pseudo-aléatoire de 3600 bits.

1. A mesure que le nombre d'itérations augmente, les performances du système s'améliorent de façon progressive. Toutefois, au bout d'un certain nombre d'itérations, l'algorithme converge.

2. Le nombre d'itérations avant convergence des performances est fonction du RSB. Dans le cas de notre exemple, 3 itérations sont suffisantes à 3dB, tandis qu'à 1dB, une dizaine d'itérations peuvent être nécessaires.

3. Les différents résultats que nous avons obtenus par simulation ainsi que ceux trouvés dans la littérature, montrent que, pour un RSB donné, le nombre d'itérations avant convergence est aussi fonction du nombre de bits par bloc: plus les blocs sont grands, et plus le nombre d'itérations augmente pour les faibles RSB. Pour des blocs de 65536 bits, 18 itérations ont été effectuées à 0,7dB [7], tandis qu'à cette même valeur du RSB, comme le montre la figure 4.12, 10 itérations sont largement suffisantes pour des blocs de 3600 bits.

4. A partir d'un certain E_b/N_0, la pente des courbes de performances des codes turbo chute subitement et semble atteindre un seuil lié au propriétés du code utilisé et de l'entrelaceur.

Comme le nombre d'itérations requises dépend, et du RSB, et de la longueur des blocs, il serait avantageux, afin d'optimiser le fonctionnement du décodeur turbo, d'interrompre le processus itératif dès que le gain apporté par des itérations supplémentaires devient minime. Pour ce faire, deux solutions ont été proposées par les auteurs de [34] et [20]. La première technique consiste à évaluer la variance de la sortie de Décodeur 2, et d'arrêter les itérations dès que cette variance atteint une borne inférieure fixée, qui dépend du RSB. La deuxième technique, proposée par les auteurs de [20], repose sur l'évaluation de l'entropie conjointe des sorties de Décodeur 2 correspondant à deux itérations successives. Plus élaborée que la précédente, cette technique donne de très faibles écarts de performances par rapport à un nombre illimité d'itérations.

4.6 Conclusion

Grâce à l'extraction de l'information extrinsèque des LRV générés par l'algorithme MAP et à l'utilisation de l'entrelaceur, l'échange d'information entre les décodeurs MAP a pu se faire sans corrélation, permettant ainsi d'améliorer les performances obtenues d'itération en itération.

Par ailleurs, le gain apporté par l'entrelacement dépend de l'entrelaceur utilisé. L'analyse des principes de fonctionnement des entrelaceurs déterministe et pseudo-aléatoire, a montré que ce dernier devrait donner de meilleures performances grâce à une meilleure répartition des paquets de bruit sur tout le bloc. Il apparait alors évident que l'augmentation de la taille des blocs en conjonction avec un choix judicieux de l'entrelaceur, ne peut être que bénéfique pour l'amélioration des performances des codes turbo.

Le comportement des codes turbo semble présenter une caractéristique assez singulière, dans la mesure où à partir d'un certain RSB, l'amélioration des performances subit un palier.

Une analyse plus détaillée des performances sera faite au chapitre suivant, où nous regarderons en particulier l'influence des codes utilisés et de l'entrelaceur sur les résultats obtenus.

Chapitre 5

ANALYSE DES PERFORMANCES DES CODES TURBO

5.1. Introduction

Passée la vague de scepticisme qui accueillit la découverte des codes turbo avec leurs performances exceptionnellement proches de la capacité, cette nouvelle classe de codes a été reconnue dans la communauté des théoriciens de l'information et du codage comme étant le développement le plus important apporté dans le domaine depuis plusieurs années. Plusieurs équipes de recherche de par le monde ont pu en effet reproduire les résultats originaux présentés par Berrou *et al.* [7], et des améliorations ont vite commencé à être proposées.

Les premiers travaux sur les codes turbo étaient presque tous expérimentaux, basés sur des résultats de simulation, sans qu'il n'y ait réellement d'études théoriques. La raison à cela est qu'une analyse théorique classique de la concaténation parallèle de codes s'est avérée très complexe. Le "mystère" des performances des codes turbo est ainsi resté presque entier pendant quelque temps. Néanmoins, les simulations ont rapidement montré qu'en plus de l'algorithme utilisé pour le décodage turbo, les éléments constituant le codeur, notamment les codes élémentaires et l'entrelaceur, avaient également une grande influence sur les résultats obtenus.

Les premières simulations des codes turbo utilisaient l'algorithme MAP [7]. Elles ont été effectuées sur un canal gaussien, avec la concaténation de deux codeurs convolutionnels identiques, récursifs et systématiques, à 16 états (K=5), et avec les générateurs G=(1, 21/37). Un TEB=10^{-5} a été alors obtenu pour un taux de codage global R=1/2, à E_b/N_0=0,7dB, soit à 0,7dB de la capacité. Il est à noter cependant que l'entrelaceur pseudo-aléatoire utilisé avait une longueur de quelque 65536 bits, et que le

décodage se faisait en 18 itérations.

Les mêmes codes ont été repris par Pietrobon et Barbulescu (1994) [31], mais cette fois en utilisant un entrelaceur bloc de 400 bits, et l'algorithme log-MAP. Un TEB=10^{-4} a été atteint à E_b/N_0=2dB après 8 itérations.

Jung *et al.* ont considéré les codes turbo pour une éventuelle transmission de la voix dans des systèmes de radio mobile, où les blocs sont de longueur 192 bits [23]. Toujours avec les mêmes caractéristiques du système proposée dans [7], 800 entrelaceurs pseudo-aléatoires ont été testés, et les meilleures performances ont été obtenues pour un TEB=$1{,}2.10^{-3}$ à E_b/N_0=2dB. Dans [24], les mêmes auteurs ont conclu que l'effet du choix de l'entrelaceur sur les performances des codes turbo était négligeable dans le cas de la transmission de petits blocs.

Une comparaison des codes turbo aux codes convolutionnels, basée sur leurs complexités respectives, a été effectuée dans une récente publication, pour des blocs de longueur inférieure à 300 bits [26]. Les auteurs soutiennent alors que si la complexité est considérée, les codes convolutionnels donneraient de meilleures performances. Nous reviendrons sur ce point à la fin du chapitre, pour montrer que ces conclusions ont été émises un peu trop hâtivement.

Depuis la confirmation de la capacité de correction d'erreur des codes turbo, un très grand nombre d'études, portant sur les différentes parties du système, ont été effectuées, mais encore la plupart du temps sur la base de résultats obtenus par simulation. Une question clé restait alors sans réponse: comment la concaténation de deux codes très simples pouvait-elle donner d'aussi bonnes performances?

Les premiers à avoir apporté une explication théorique au comportement des codes turbo furent Benedetto et Montorsi (1996) [4,5]. Les auteurs ont alors proposé un modèle des codes turbo, à travers lequel ils ont établi une borne supérieure sur la probabilité d'erreur en supposant le décodage à maximum de vraisemblance. Il fut alors

prouvé analytiquement que si les codeurs élémentaires constituant le codeur turbo sont récursifs et systématiques, alors le TEB est inversement proportionnel à la longueur de l'entrelaceur.

Une autre analyse intéressante des propriétés des codes turbo a été présentée par Perez *et al.* [29]. Une interprétation des performances basée sur l'étude du spectre de distances est proposée, apportant une réponse à plusieurs questions jusque là inexpliquées. Il y est par exemple démontré que le redressement des courbes de performances des codes turbo est essentiellement dû à leur faible distance libre.

Nous présentons dans ce chapitre une analyse des performances des codes turbo en fonction des paramètres des codeurs élémentaires et de l'entrelaceur. Cette analyse est essentiellement basée sur les résultats de simulation que nous avons obtenus, en faisant toutefois référence à certains résultats théoriques présentés en particulier dans [4,5,29]. Nous nous intéressons essentiellement au codage par blocs de petites à moyennes dimensions (moins de 1000 bits), pour lequel nous montrons les différences par rapport à l'utilisation de blocs de longueurs beaucoup plus grandes.

5.2 Les codes convolutionnels systématiques, récursifs et non récursifs

Une comparaison des codes convolutionnels récursifs et systématiques (CRS) aux codes convolutionnels non systématiques, basée sur leurs spectres de distances et leurs TEB, a été menée dans [8]. Les résultats obtenus démontraient alors que les deux types de codes présentaient des performances très similaires, avec un léger avantage cependant pour les codes CRS à de faibles RSB.

Dans [4], les auteurs ont comparé quant à eux les performances des codes CRS à celles des codes convolutionnels systématiques non récursifs. Les résultats obtenus étaient encore une fois très similaires. Toutefois, un point intéressant est à noter. Le

poids de Hamming minimal (w_{min}) d'une séquence d'entrée pouvant générer un événement erreur, est égal à 2 pour les codes récursifs (de taux $R=1/V$), et à 1 pour les codes non récursifs. Rappelons qu'un événement erreur correspond à un chemin dans le treillis qui diverge du chemin nul, pour y retourner après un nombre fini de transitions, avec une métrique supérieure à celle du chemin nul. L'explication à cela est simple: dans le cas des codes non récursifs, la transmission d'une queue de $K-1$ zéros après la transmission d'un bit à 1, garantit le retour à l'état nul. Par contre, dans le cas des codes récursifs, une fois que l'on a quitté l'état 0 par la transmission d'un bit à 1, une queue composée d'au moins un bit non nul est requise pour le retour. Au moins deux bits non nuls sont donc nécessaires pour provoquer un événement erreur.

C'est précisément cette propriété que Benedetto et Montorsi ont utilisé, pour démontrer que les codes élémentaires utilisés dans la concaténation formant le codeur turbo *doivent être récursifs* [5]. Leur démonstration est basée sur le fait que si $w_{min}=2$, alors le TEB des codes turbo est inversement proportionnel à la longueur des blocs. Tandis que si $w_{min}=1$, alors l'augmentation de la longueur de l'entrelaceur n'apporte aucune amélioration aux performances. C'est ainsi que les codeurs convolutionnels utilisés pour le codage turbo ont toujours été choisis récursifs et systématiques. Pour toutes les simulations effectuées dans le cadre de ce mémoire, nous avons utilisé des codeurs élémentaires récursifs et systématiques de taux $R=1/2$.

Il est à noter cependant que la concaténation parallèle de deux codeurs CRS à travers un entrelaceur n'est pas sans entraîner une certaine complication. En effet, à cause de la présence de l'entrelaceur, les états dans lesquels se retrouvent les deux codeurs, après le codage d'une séquence d'information, sont différents avec une très grande probabilité. Or, la réinitialisation d'un codeur CRS nécessite l'utilisation d'une queue non nulle qui dépend de l'état final. Par conséquent, la probabilité de réinitialiser simultanément les deux codeurs à l'aide d'une seule queue est très faible.

5.3 Réinitialisation des codes turbo

Lors de la description de l'algorithme MAP, utilisé pour le décodage turbo, nous avons supposé que l'état initial du codeur et son état final, après le codage de chaque bloc, sont tous deux égaux à zéro. Ceci doit donc être le cas pour les deux codeurs élémentaires. Le problème qui se pose alors est que les états des deux codeurs, après le codage d'un bloc sont différents, et que par conséquent, les séquences nécessaires à leur réinitialisation sont aussi différentes. De plus, la présence de l'entrelaceur, surtout si celui-ci est pseudo-aléatoire, fait que, en supposant que l'on transmette la queue nécessaire à la réinitialisation du premier codeur, les chances que l'on retrouve, à la sortie de l'entrelaceur, la bonne séquence pour la remise à zéro du deuxième codeur, sont infimes.

Dans certaines publications traitant du sujet [23,29,34], les auteurs suggèrent de ne remettre à zéro qu'un seul codeur, laissant *ouvert* le treillis de l'autre. Cela n'engendrerait selon eux, qu'une légère baisse des performances. Sur la figure 5.1, nous avons représenté les performances obtenues pour des longueurs de blocs de 100 à 900 bits, avec et sans remise à zéro du deuxième codeur.

Ces résultats sont obtenus après 6 itérations, avec l'utilisation de codes élémentaires à 16 états, pour un taux de codage global R=1/2. Nous constatons alors que l'influence de la remise à zéro des deux codeurs plutôt que d'un seul, est d'autant plus importante que la taille des blocs est faible. Ainsi obtient-on par exemple un gain d'environ 0,7dB à un TEB=10^{-3} pour des blocs de 100 bits. Ce gain a toutefois tendance à diminuer à mesure que l'on augmente la taille des blocs, pour atteindre 0,1dB à un TEB=10^{-6} pour des blocs de 900 bits.

Comme l'intérêt de notre recherche porte essentiellement sur les blocs de petites à moyennes dimensions (moins de 1000 bits par bloc), nous avons opté, dans toutes nos simulations, pour une remise à zéro des deux codeurs élémentaires. Cette opération

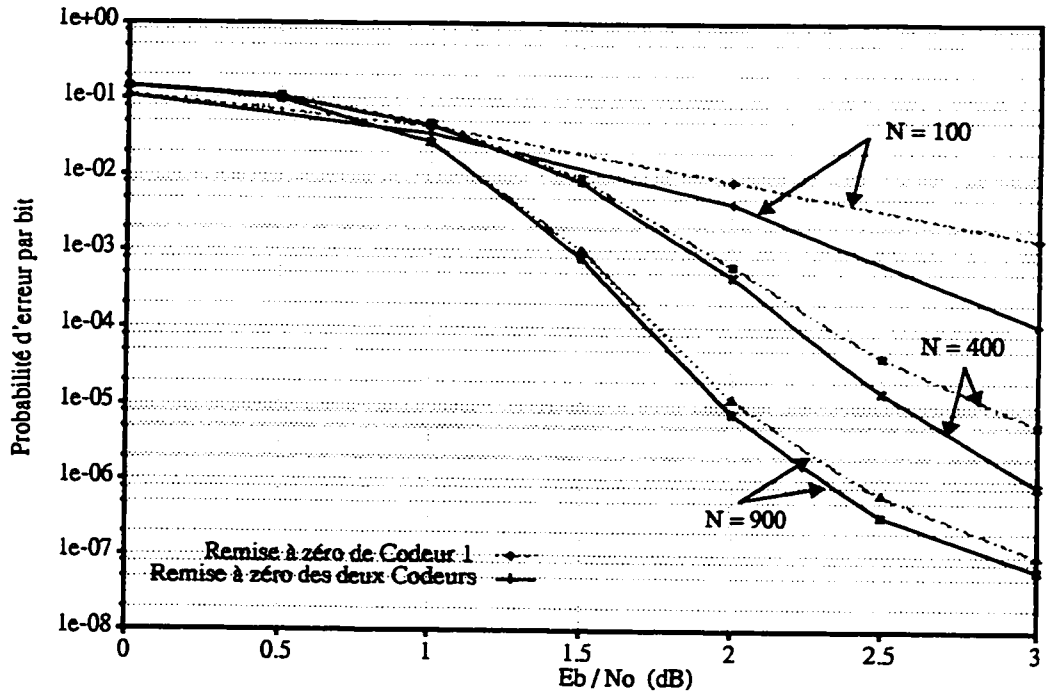


Figure 5.1 Influence de la réinitialisation des deux codeurs élémentaires avec K=5, G=(1, 21/37), 6 itérations

est effectuée par la transmission d'une queue de K-1 bits pour chacun des codeurs élémentaires. Il est à noter cependant que les résultats présentés sur la figure 5.1 et correspondant à un taux de codage global R=1/2, ont été obtenus par la transmission de deux queues perforées, tout comme les séquences de parité générées par les deux codeurs. Cela équivaut donc à la transmission d'une seule queue non perforée, et le taux de codage global se verrait ainsi inchangé: $R = L/(L + (K - 1))V$.

Dans [2], les auteurs proposent un modèle d'entrelaceur qui permet, après la transmission d'une séquence d'information, de terminer les treillis des deux codeurs dans le même état. Dans ce cas, une seule queue permet de remettre à zéro les deux codeurs simultanément. L'inconvénient de cet entrelaceur cependant, est qu'en plus du fait

que sa longueur dépend de la longueur de contrainte des codes élémentaires, il est basé sur un entrelaceur hélicoïdal, et donne donc comme nous le démontrons plus loin, de moins bonnes performances qu'un entrelaceur pseudo-aléatoire dès que la longueur des blocs dépasse 200 bits.

Dans une publication récente, S. Pietrobon propose une méthode pour l'implémentation de l'algorithme MAP, permettant d'effectuer un décodage continu des codes turbo [30]. Cette méthode est basée sur une technique de fenêtre glissante, qui présente deux avantages principaux. D'une part, la taille des blocs lors du décodage a pu être diminuée sans dégradation importante des performances, et par suite les délais dûs à la réitération du processus de décodage ont également été réduits. D'autre part, il n'est plus nécessaire de réinitialiser les codeurs puisque le codage se fait de façon continue. Cette solution présente toutefois l'inconvénient de nécessiter beaucoup plus de calculs que dans le cas d'un codage par bloc, ainsi que des exigences plus rigoureuses en termes de synchronisation.

5.4 Optimisation des codes élémentaires

Lors de la première présentation des codes turbo [7], excepté l'argumentation du fait que les codes élémentaires étaient choisis récursifs et systématiques, aucun critère n'a été mentionné pour le choix des paramètres de ces codes, en l'occurrence leur longueur de contrainte et les générateurs utilisés. Il était simplement signalé que les meilleurs résultats ont été obtenus avec des codes à 16 états, utilisant les générateurs G=(1, 21/37). Cette omission trouve cependant sa justification, comme nous le démontrerons, dans le fait que les entrelaceurs choisis étaient de longueur énorme (65536 bits), et par suite les performances n'étaient examinées qu'à de très faibles RSB.

Si nous observons le comportement des performances des codes turbo sur une plage de RSB suffisamment grande, nous constatons qu'à partir d'un E_b/N_0 donné, les cour-

bes d'erreur semblent tendre vers une asymptote d'inclinaison très faible; ceci étant observé indépendamment du choix des paramètres du système. Les deux exemples représentés sur les figures 5.2 et 5.3, illustrent bien ce comportement singulier, et l'on peut alors constater que le taux d'erreur par bit connaît une sorte de saturation à partir d'un RSB autour de 2dB.

L'analyse de cette particularité des codes turbo a été effectuée par Perez *et al.* dans [29]. Les auteurs ont alors démontré que pour des RSB modérés à élevés, les probabilités d'erreur par bit obtenues avec les codes turbo approchent l'asymptote décrite par l'équation (5.1).

$$P_b \approx \frac{N_{libre}\tilde{w}_{libre}}{N} Q\left(\sqrt{d_{libre}\frac{2RE_b}{N_0}}\right) \tag{5.1}$$

où d_{libre} = distance libre du code, égale au poids de Hamming minimal pour tous les mots de codes;

N_{libre} = multiplicité des mots de codes à la distance libre;

$\tilde{w}_{libre}$ = poids moyen des séquences d'information occasionnant des mots de codes à la distance libre;

N = longueur des blocs d'information;

R = taux de codage global;

E_b = énergie par bit d'information;

N_0 = densité spectrale de puissance du bruit.

L'expression décrite par l'équation (5.1) a été obtenue par analogie à la borne union calculée pour la probabilité d'erreur par bit des codes convolutionnels, décodés par un algorithme à maximum de vraisemblance sur un canal gaussien. Une approximation a été ensuite effectuée sous l'hypothèse que, pour des valeurs relativement élevées de E_b/N_0, le terme relatif à la distance libre domine la borne union sur la probabilité d'erreur par bit.

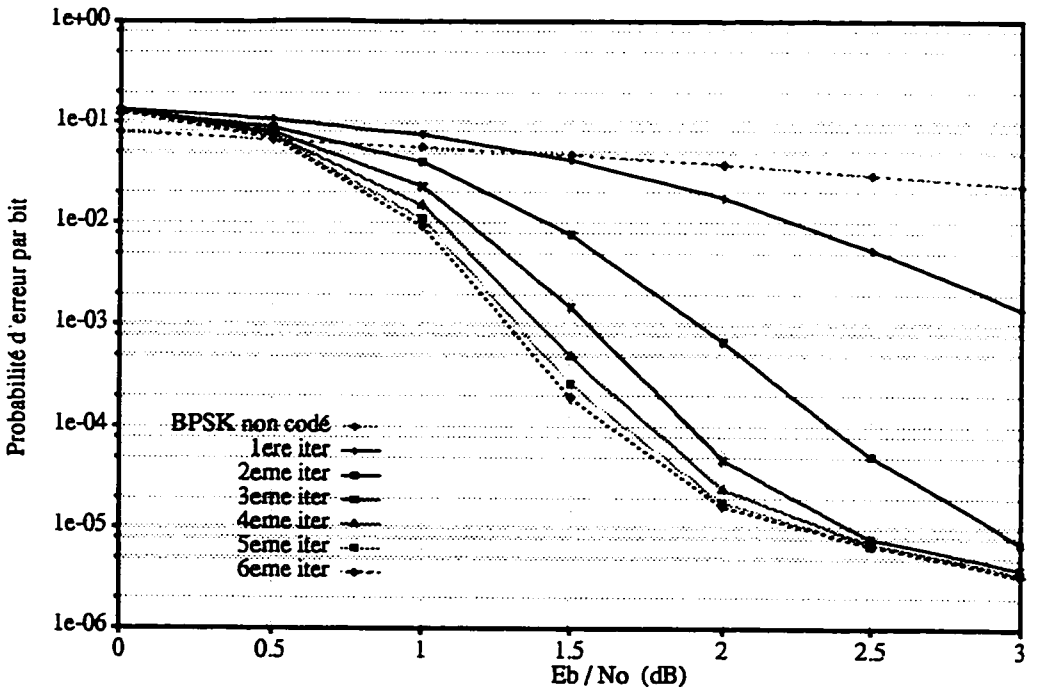


Figure 5.2 Performances d'erreur pour K=5, G=(1, 21/37), R_{global}=1/2, N=900

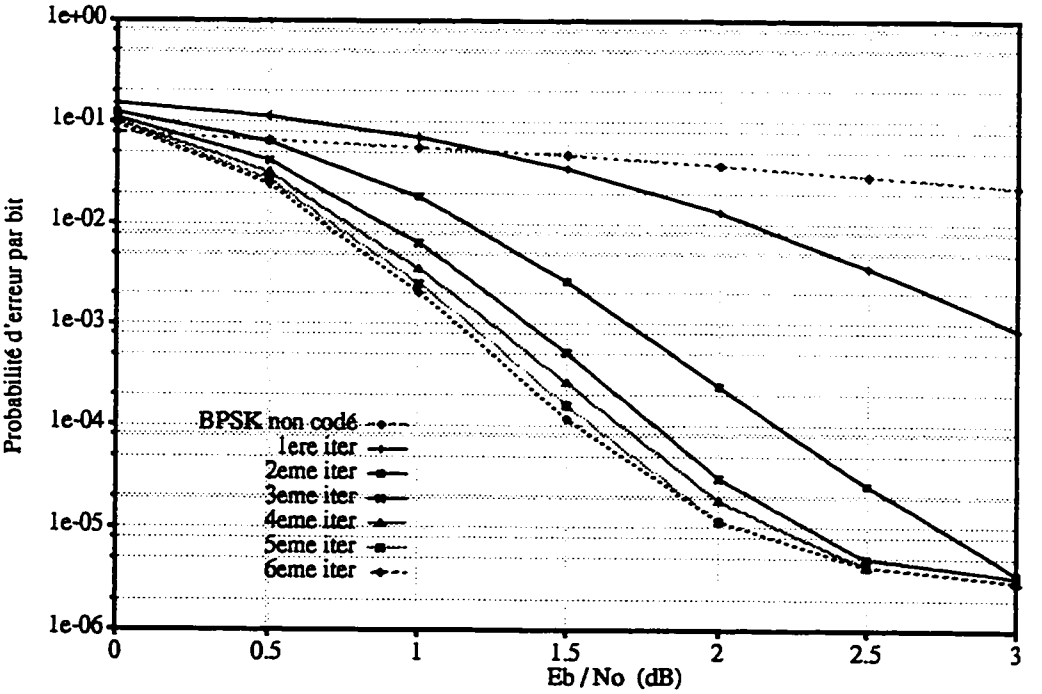


Figure 5.3 Performances d'erreur pour K=5, G=(1, 21/37), R_{global}=1/3, N=400

En admettant le fait que si un entrelaceur pseudo-aléatoire est utilisé, alors $(N_{libre}\tilde{w}_{libre})/N << 1$, nous constatons que la pente de l'asymptote est dominée par la distance libre du code. Nous pouvons alors en déduire que la saturation observée sur les courbes de performance à partir d'un E_b/N_0 donné, est due au fait que la distance libre des codes turbo est relativement très faible. Par conséquent, la sélection des codes élémentaires de sorte à maximiser la distance libre du code turbo qu'ils constituent, peut s'avérer importante surtout si le système doit fonctionner à des RSB relativement élevés (supérieurs à 2,5dB pour des blocs de moins de 1000 bits).

La détermination de la distance libre d'un code turbo est assez complexe; un algorithme pour le faire a été proposé dans [36]. Toutefois, il a été démontré que le choix des générateurs des codes élémentaires permet de faire varier ce paramètre de façon plus ou moins importante.

5.4.1 Choix des générateurs des codes élémentaires

Afin de maximiser la distance libre des codes turbo, une procédure permettant d'obtenir les meilleurs générateurs pour les codes élémentaires a été proposée dans [5]. La méthode proposée suggère de choisir un polynôme premier de degré *K-1* comme polynôme générateur correspondant à la boucle de contre-réaction des codeurs CRS. Le deuxième générateur est alors recherché de sorte que la distance libre du code concaténé soit maximale. Si plusieurs polynômes donnent la même distance, il faut alors examiner la répartition des poids ainsi que les TEB obtenus à différents RSB. Il a été constaté en effet, qu'un écart pouvant aller jusqu'à 3dB peut être observé en comparant, à des RSB assez élevés, les performances de deux codes ayant la même distance libre.

Il est à noter cependant que toutes ces conclusions sont basées essentiellement sur l'évaluation théorique de bornes supérieures sur les probabilités d'erreur, ainsi que sur le modèle de l'entrelaceur uniforme proposé par les auteurs. Ce dernier est défini

comme étant une permutation choisie au hasard parmi les $\binom{N}{k}$ permutations possibles de k bits à 1 dans un bloc de N bits.

Un appui au choix d'un polynôme premier pour le générateur de contre-réaction a été apporté dans [29]. L'argument fourni est basé encore une fois sur l'hypothèse de l'utilisation d'un entrelaceur pseudo-aléatoire, choisi au hasard parmi toutes les combinaisons possibles, dont la longueur tend vers l'infini.

Dans le but de vérifier l'importance du choix des générateurs sur les performances obtenues, surtout pour de faibles longueurs de blocs, nous avons testé lors de nos simulations plusieurs codes, pour différentes longueurs de contraintes. Ces codes ont été choisis un peu au hasard, parmi les plus utilisés dans la littérature. Sur les figures 5.4 à 5.7, nous avons représenté deux exemples de résultats obtenus par simulation de deux codes, résultant de la concaténation de codeurs à 16 états, et utilisant respectivement les générateurs G=(1, 21/37) et G=(1, 27/31).

A partir de ces figures, nous pouvons constater les points suivants:

1. Le passage du code G=(1, 21/37) au code G=(1, 27/31) a permis, aussi bien pour N=400 que pour N=900, d'apporter une nette amélioration aux TEB obtenus à partir de E_b/N_0=2,5dB. Cette amélioration est due de façon évidente à l'augmentation de la pente de l'asymptote sur cette plage de RSB, et si nous nous fions à l'analyse théorique [5], cela serait expliqué par la supériorité de la distance libre du code utilisant le générateur (1, 27/31) par rapport à celle du code utilisant le générateur G=(1, 21/37).
2. En observant les figures 5.6 et 5.7, nous constatons que les performances connaissent dans les deux cas une saturation. Toutefois, avec le générateur (1, 27/31), cette saturation se produit à un TEB plus faible et un RSB plus élevé.
3. Le gain apporté par l'utilisation du générateur G=(1, 27/31), *pour des valeurs du RSB entre 2 et 3dB*, est bien plus important dans la cas de N=900 que dans celui de N=400 bits. Ceci devient tout à fait compréhensible si l'on considère que la

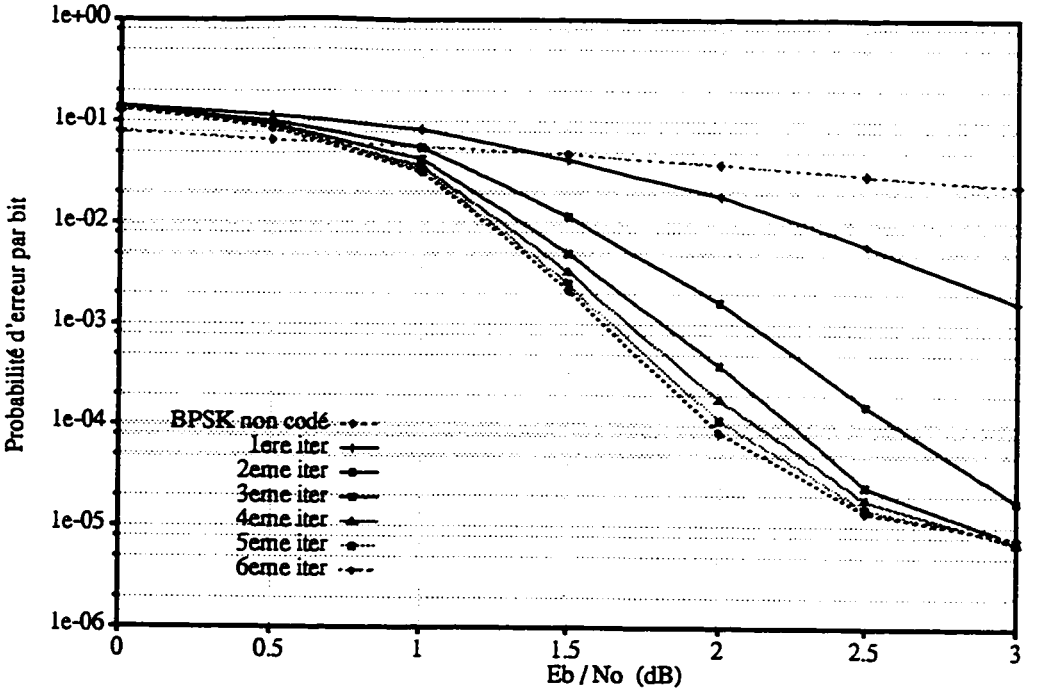


Figure 5.4 Performances d'erreur du code turbo de taux R=1/2, avec N=400, Entrelaceur aléatoire, K=5, G=(1, 21/37)

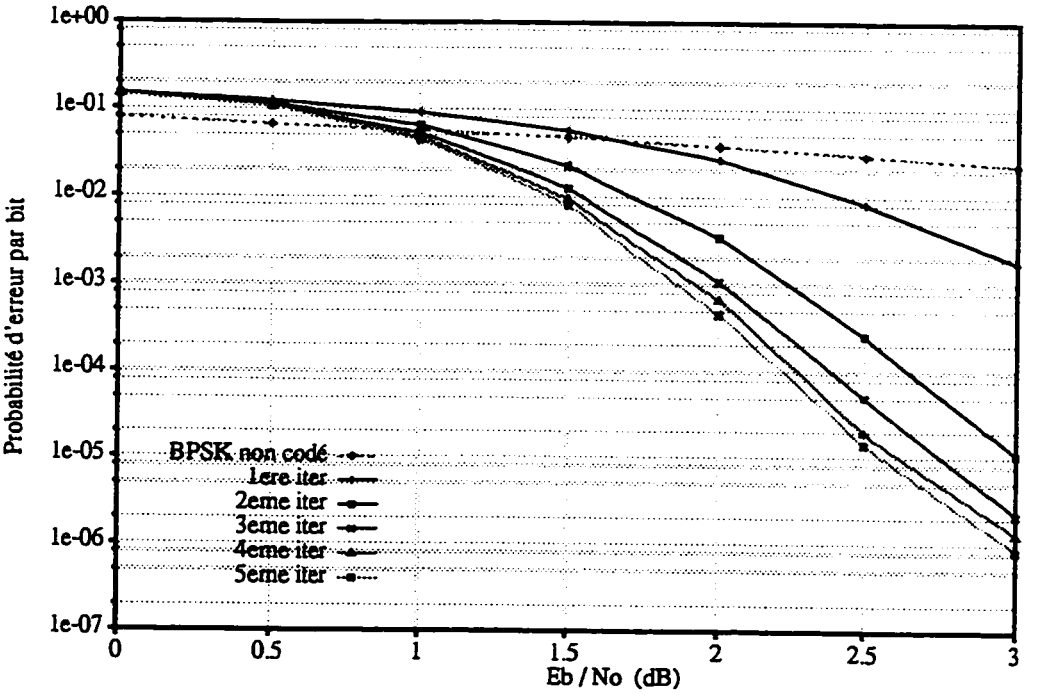


Figure 5.5 Performances d'erreur du code turbo de taux R=1/2, avec N=400, Entrelaceur aléatoire, K=5, G=(1, 27/31)

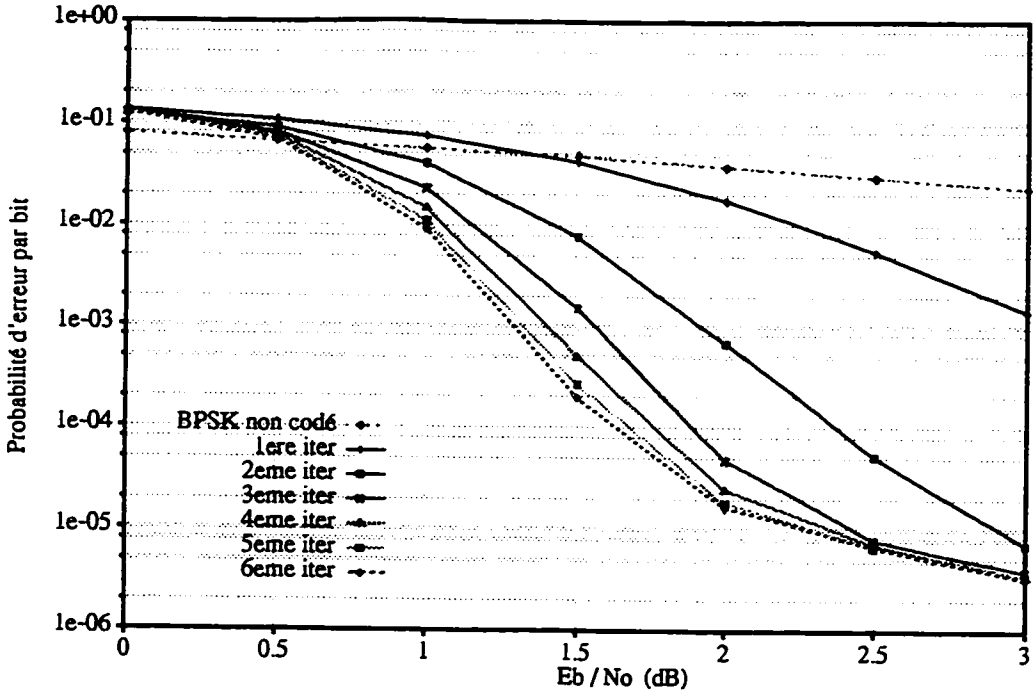


Figure 5.6 Performances d'erreur du code turbo de taux R=1/2, avec N=900, Entrelaceur aléatoire, K=5, G=(1, 21/37)

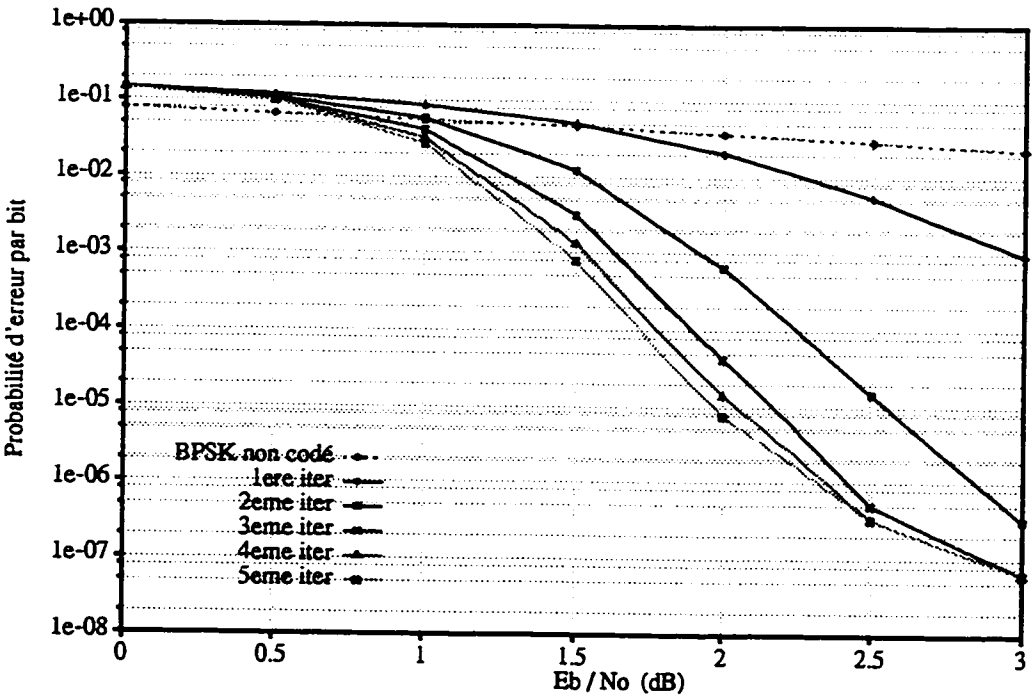


Figure 5.7 Performances d'erreur du code turbo de taux R=1/2, avec N=900, Entrelaceur aléatoire, K=5, G=(1, 27/31)

saturation des performances commence à se produire pour un code donné à un certain $(TEB)_{sat}$, indépendamment de la longueur des blocs. $(TEB)_{sat}$, qui est de l'ordre de 10^{-6} pour le code utilisant le générateur (1, 21/37), est presque atteint à 3dB pour des blocs de 400 bits. Tandis que pour le code utilisant le générateur G=(1, 27/31), les performances pour des blocs de 400 bits, n'ont pas encore atteint la saturation, qui se produit à environ $(TEB)_{sat} = 10^{-8}$.

4. Si nous examinons les performances des deux codes à 1,5dB, donc avant que les courbes ne commencent à se redresser, nous constatons que le TEB atteint après un certain nombre d'itérations est plus faible avec G=(1, 21/37). La différence est encore une fois d'autant plus visible que N est grand. Cela impliquerait donc que le comportement d'un code est différent selon que l'on se place avant ou après le RSB à partir duquel les performances commencent à saturer. Et cela justifierait également le fait que le code utilisant le générateur G=(1, 37/21) ait donné les meilleures performances lorsque celles-ci ont été observé à E_b/N_0=0,7dB pour des blocs de 65536 bits [7].

Somme toute, il ne fait aucun doute que le choix des générateurs des codes élémentaires est très important pour l'obtention des meilleures performances après concaténation. Toutefois selon nous, ce choix dépend d'un trop grand nombre de paramètres pour que l'on puisse établir une méthode rigoureuse de recherche des meilleurs codes. Cette recherche serait alors beaucoup plus empirique, et l'optimisation du choix des codes élémentaires dépendrait étroitement de la longueur des blocs, ainsi que de la plage de RSB où le codeur turbo est appelé à fonctionner.

5.4.2 Longueur des codes élémentaires

Sur la figure 5.8 nous avons rapporté les résultats obtenus par simulation de différents codes, en variant la longueur de contrainte des codes élémentaires.

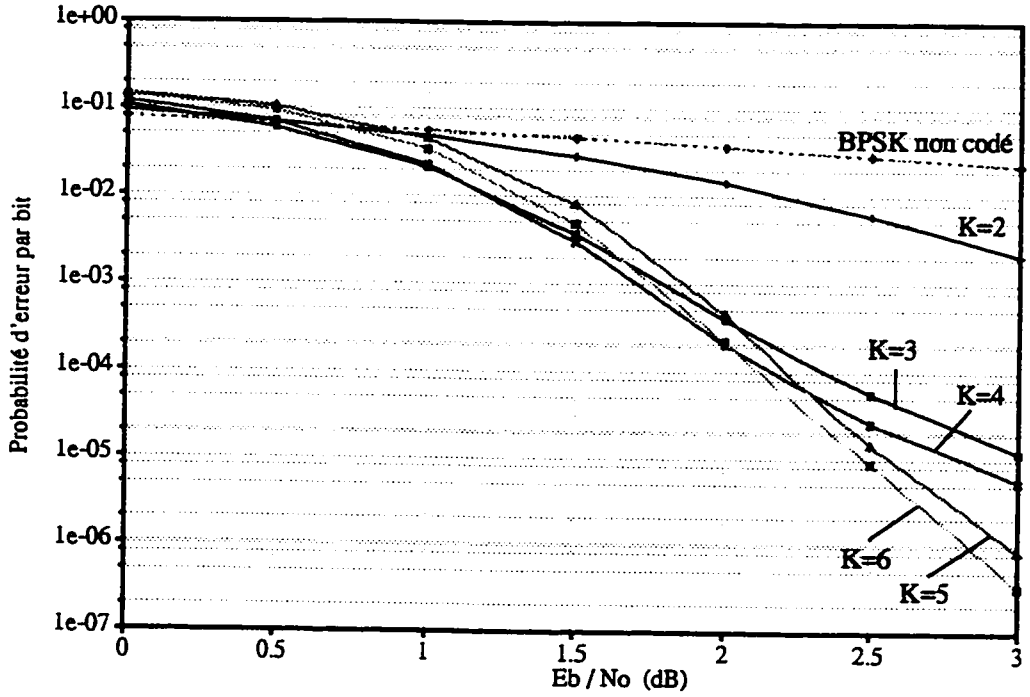


Figure 5.8 Performances d'erreur des codes turbo de taux R=1/2, avec N=400, Entrelaceur aléatoire, 6 itérations

Dans le cas de la concaténation de codeurs dont la longueur de contrainte est K=2, il est clair que le code résultant est trop faible pour que les itérations puissent apporter une améliorations aux performances. Quant aux autres codes (K=3 à 6), la comparaison des performances peut être faite en deux parties.

Si nous observons les performances au delà de 2,5dB, il est alors clair que l'augmentation de la longueur de contrainte des codes élémentaires permet d'améliorer les performances des codes turbo. Ainsi par exemple, un gain de 0,5dB est obtenu à 10^{-5} en passant de K=3 à K=6. Il est à noter cependant que ce gain se fait au prix d'une augmentation de la complexité; mais il serait également utile de signaler que le délai supplémentaire engendré par l'accroissement de la complexité est relativement très faible lorsque comparé à celui qu'entraîne l'augmentation de la longueur des blocs.

Si l'on se place maintenant à TEB=10^{-3}, il en va alors tout autrement: K=4 donne dans ce cas les meilleures performances, suivi dans l'ordre par K=6, K=3 et K=5. Il est alors vraisemblable que les codes élémentaires avec K=5 aient pu donner les meilleures performances à E_b/N_0=0,7dB [7]. Nous ne sommes pas en mesure de fournir une explication précise à ce comportement, mais soupçonnons fortement que le choix des générateurs y est pour beaucoup. En effet, comme nous l'avons montré au paragraphe précédent, un code qui donne les meilleures performances à un RSB relativement élevé (dépendamment de la longueur des blocs), ne donne pas nécessairement les meilleurs résultats si le RSB est plus faible, surtout si les deux points de fonctionnement se trouvent de part et d'autre du coude formé par les courbes de performances. Ainsi, si les générateurs des différents codes étaient choisis de façon à optimiser leur fonctionnement pour de très faibles RSB, le gain apporté par une augmentation de la longueur de contrainte pourrait alors redevenir régulier.

Contrairement aux codes convolutionnels, l'augmentation de la distance libre des codes turbo ne garantit pas l'amélioration des performances d'erreur. Le choix des générateurs des codes élémentaires utilisés se fait plutôt de sorte à obtenir les meilleurs spectres.

5.5 Choix de l'entrelaceur

Suite à l'analyse du fonctionnement du décodeur turbo au chapitre 4, nul doute ne subsiste quant à l'importance du rôle que joue l'entrelaceur dans l'amélioration progressive des performances d'itération en itération. Rappelons que l'entrelacement permet, d'un côté de disperser les paquets d'erreurs sur tout le bloc, et d'un autre côté de décorréler les entrées des deux décodeurs MAP constituant le décodeur turbo. Nous avons également présenté au chapitre 4 les deux types principaux d'entrelaceurs, notamment l'entrelaceur bloc, ou plus généralement les entrelaceurs déterministes, et l'entrelaceur pseudo-aléatoire.

Nous nous proposons dans ce paragraphe d'évaluer l'importance du choix des paramètres de l'entrelaceur utilisé, en l'occurrence son type (bloc ou pseudo-aléatoire) et sa longueur, en vue de l'obtention des meilleures performances des codes turbo.

5.5.1 Longueur des blocs

L'équation (5.2), décrit l'expression asymptotique de la borne supérieure sur la probabilité d'erreur par bit des codes turbo, proposée par Benedetto et Montorsi [5]. Cette expression suppose que le décodage est à maximum de vraisemblance, que les codes élémentaires sont récursifs et systématiques, et que l'entrelaceur utilisé est uniforme.

$$P_b \leq \sum_{k=1}^{\lfloor N/2 \rfloor} 2k \binom{2k}{k} \frac{1}{N} \frac{\left(H^{2+2z_{min}}\right)^k}{\left(1-H^{2z_{min}-2}\right)^{2k}} \Bigg|_{H=exp(-\frac{RE_b}{N_0})} , \qquad (5.2)$$

où z_{min} est le poids minimum des bits de parité dans les événements erreur générés par des séquences d'information de poids w=2.

D'après l'équation (5.2), et sous toutes les hypothèses citées plus haut, il apparait alors que le gain apporté par l'entrelaceur est inversement proportionnel à la longueur N des blocs. Les propriétés du code sont représentées par le taux de codage R et le paramètre z_{min}.

L'influence de la longueur de l'entrelaceur apparait de la même façon dans l'équation (5.1), dont nous rappelons l'expression:

$$P_b \approx \frac{N_{libre}\tilde{w}_{libre}}{N} Q\left(\sqrt{d_{libre}\frac{2RE_b}{N_0}}\right)$$

Là encore, les codes élémentaires sont supposés récursifs et systématiques, et l'entrelaceur est supposé pseudo-aléatoire. L'avantage de cette expression par rapport à (5.2)

est qu'elle illustre mieux la relation entre la longueur des blocs et les propriétés du code.

Ainsi, le gain apporté par l'entrelaceur serait inversement proportionnel à N si et seulement si $(N_{libre}\tilde{w}_{libre}) << N$, ce qui est le cas si l'entrelaceur utilisé est pseudo-aléatoire de longueur très grande. Dans ce cas, il ressort que les performances des codes turbo peuvent être modifiées de deux façons. En effet, en gardant constantes la multiplicité N_{libre} et la dimension N de l'entrelaceur, et en augmentant d_{libre}, la pente de l'asymptote P_b est augmentée, permettant de la sorte d'apporter un gain d'autant plus important que le RSB est élevé. D'un autre côté, si d_{libre} et N_{libre} sont constantes, alors l'augmentation de N permet de baisser le niveau de l'asymptote sans en modifier la pente.

Jusque là, il s'est agit de conclusions issues d'analyses théoriques, basées sur un nombre d'hypothèse simplificatrices. Examinons à présent ce que cela donne en simulation, surtout lorsqu'il s'agit de longueurs de blocs relativement faibles. Sur la figure 5.9 nous avons rapporté les résultats obtenus par simulation de la concaténation de deux codes élémentaires à 4 états (K=3) avec les générateurs G=(1, 5/7). Le décodage est effectué en 6 itérations, pour des longueurs de blocs variant de 100 à 3600 bits, et le taux de codage global est R=1/2.

A partir de la figure 5.9 nous constatons les points suivants:

1. Le gain apporté par l'augmentation de la longueur des blocs est manifeste: par exemple, un gain de 3dB est obtenu à un TEB=10^{-4} en passant de N=196 à N=3600.
2. Il est facilement vérifiable que le TEB atteint pour les différentes valeurs de N n'est pas inversement proportionnel à la longueur des blocs. Ceci trouve sa justification dans le fait que l'hypothèse $(N_{libre}\tilde{w}_{libre}) << N$, n'est pas vérifiée pour des longueurs de blocs de moins de 1000 bits. Un calcul simple permet toutefois

de constater qu'à E_b/N_0=3dB par exemple, le TEB est plutôt inversement proportionnel à N^2.

3. Au paragraphe 5.4, nous avons montré que les codes turbo présentaient cette caractéristique d'atteindre, à partir d'un certain E_b/N_0, une sorte de seuil de saturation qui dépend essentiellement du code utilisé, et plus particulièrement de sa longueur de contrainte et de sa distance libre. Ce qui se passe maintenant si la longueur de l'entrelaceur est augmentée, est que ce seuil est simplement atteint plus rapidement.

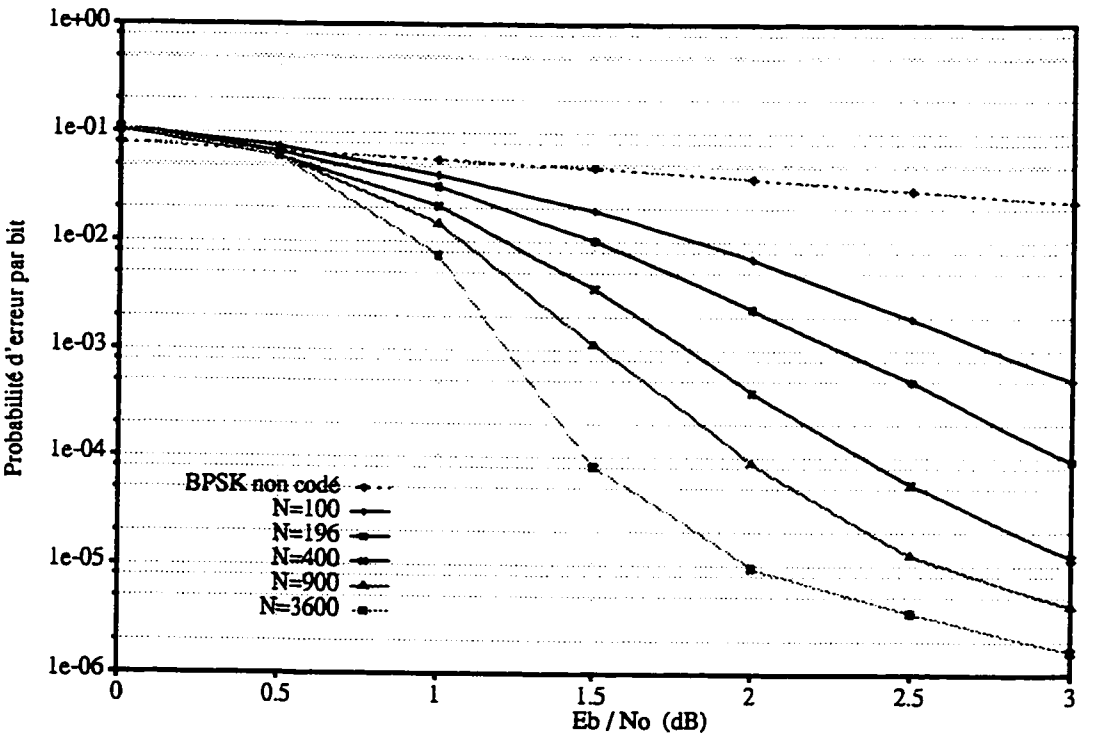


Figure 5.9 Effet de l'augmentation de la longueur de l'entrelaceur aléatoire sur la probabilité d'erreur pour R=1/2, K=3, G=(1, 5/7), 6 itérations

Sur la figure 5.10, nous avons repris à titre illustratif les résultats présentés dans [29] pour des longueurs de blocs de 16384 et 65536 bits, utilisant des codes avec K=5, G=(1, 21/37) et un décodage en 18 itérations. Nous constatons alors pour commencer que les codes K=3, G=(1, 5/7) et K=5, G=(1, 21/37) ont, malgré la

différence de leur longueur de contrainte, des seuils de saturation très voisins (autour de 10^{-6}). L'allure des courbes obtenues nous permet en effet de prévoir, avec une incertitude presque nulle, qu'à E_b/N_0=3dB, les TEB atteints avec N=3600, N=16384 et N=65536 seraient très voisins.

Ce que nous voulions montrer par là est, d'abord que l'étude de l'influence de l'augmentation de la longueur des blocs n'est significative que si les longueurs comparées sont du même ordre de grandeur: il n'est d'aucun intérêt par exemple de comparer les performances obtenues à E_b/N_0=1dB avec des blocs de 1000 bits à celles atteintes avec des blocs de 65536 bits. En effet, alors qu'un code aura atteint sa phase de saturation avec des blocs très longs, les performances du même code commenceraient à peine à s'améliorer si un entrelaceur beaucoup plus petit est utilisé. En d'autres termes, les performances des différents codes doivent être comparées loin de la région de saturation.

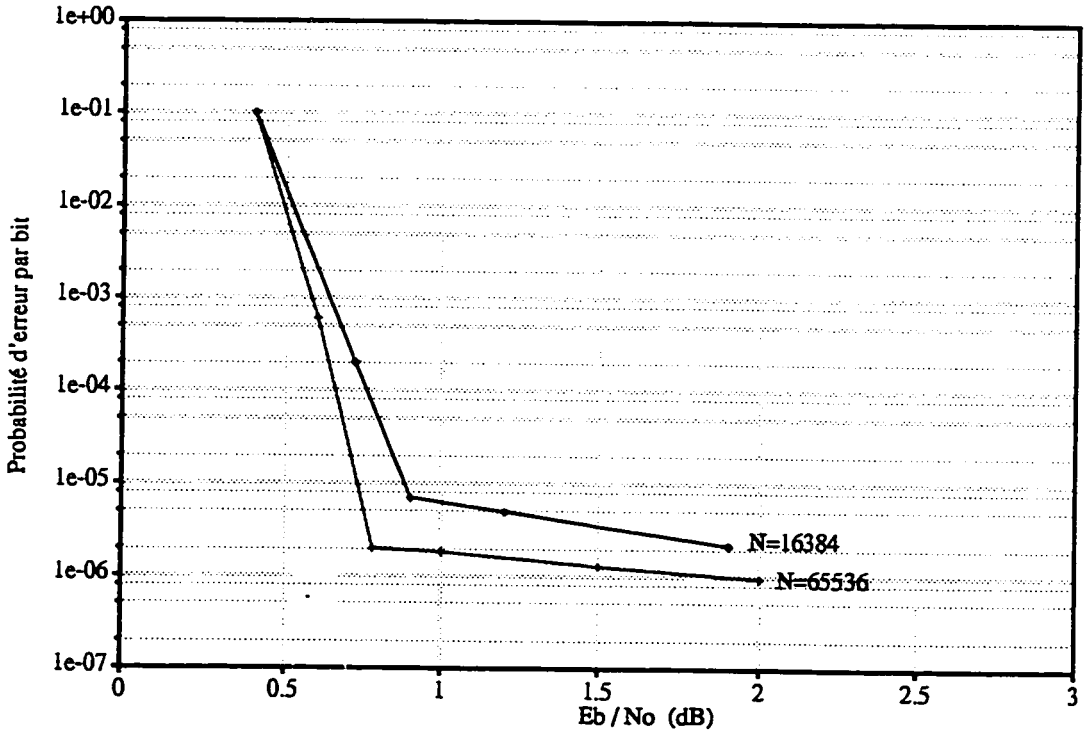


Figure 5.10 Performances d'erreur des codes turbo de taux R=1/2, avec K=5, G=(1, 21/37), Entrelaceur aléatoire, 18 itérations.

D'un autre côté, pour un ordre de grandeur de blocs donné, l'amélioration des performances engendrée par une augmentation de la taille de l'entrelaceur se limite à une certaine plage de RSB. Pour des blocs de moins de 1000 bits par exemple, cette plage se limite, pour des taux de codage R=1/2 et R=1/3 aux valeurs de E_b/N_0 comprises entre 1,5 et 3,0dB. En deçà et au delà de cet intervalle, l'amélioration due à l'augmentation de la taille des blocs devient rapidement négligeable.

Ainsi donc, l'amélioration des performances par l'augmentation de la taille des blocs constitue une autre caractéristique des codes turbo; caractéristique d'autant plus intéressante que le gain apporté n'engendre aucun accroissement de la complexité du décodage. Néanmoins, l'inconvénient majeur de cette approche est l'augmentation proportionnelle du délai dû au décodage itératif. C'est la raison pour laquelle nous avons choisi d'étudier les performances des codes turbo pour des blocs dont la taille ne dépasse pas les 1000 bits.

5.5.2 Entrelaceurs aléatoires ou entrelaceurs blocs

Sur la figure 5.11 nous avons représenté les résultats obtenus avec les mêmes simulations décrites sur la figure 5.9, en substituant toutefois l'entrelaceur pseudo-aléatoire par un entrelaceur bloc. Nous constatons alors que l'augmentation de la taille des blocs cesse très rapidement d'apporter une amélioration aux performances.

En fait, ce résultat n'est guère surprenant si l'on tient compte des principes mêmes de fonctionnement des deux entrelaceurs. En effet, nous avons montrée au chapitre 4 que, contrairement aux entrelaceurs pseudo-aléatoires, la séparation des paquets d'erreur n'était que partielle avec les entrelaceurs blocs; et la différence entre les deux types d'entrelacement est d'autant plus importantes que la dimension des blocs est grande.

Une explication de ce même fait peut encore être donnée, en comparant les spectres des codes turbo utilisant un entrelaceur pseudo-aléatoire à ceux utilisant un entrelaceur bloc [29]. En effet, il a été démontré que lorsqu'un entrelaceur bloc est utilisé, la multiplicité N_{libre} est de l'ordre de N, et ce, quelque soit la longueur des blocs. Par conséquent, le rapport $(N_{libre}\tilde{w}_{libre})/N$ demeure quasi constant, et l'augmentation de N n'apporte alors aucune amélioration aux performances.

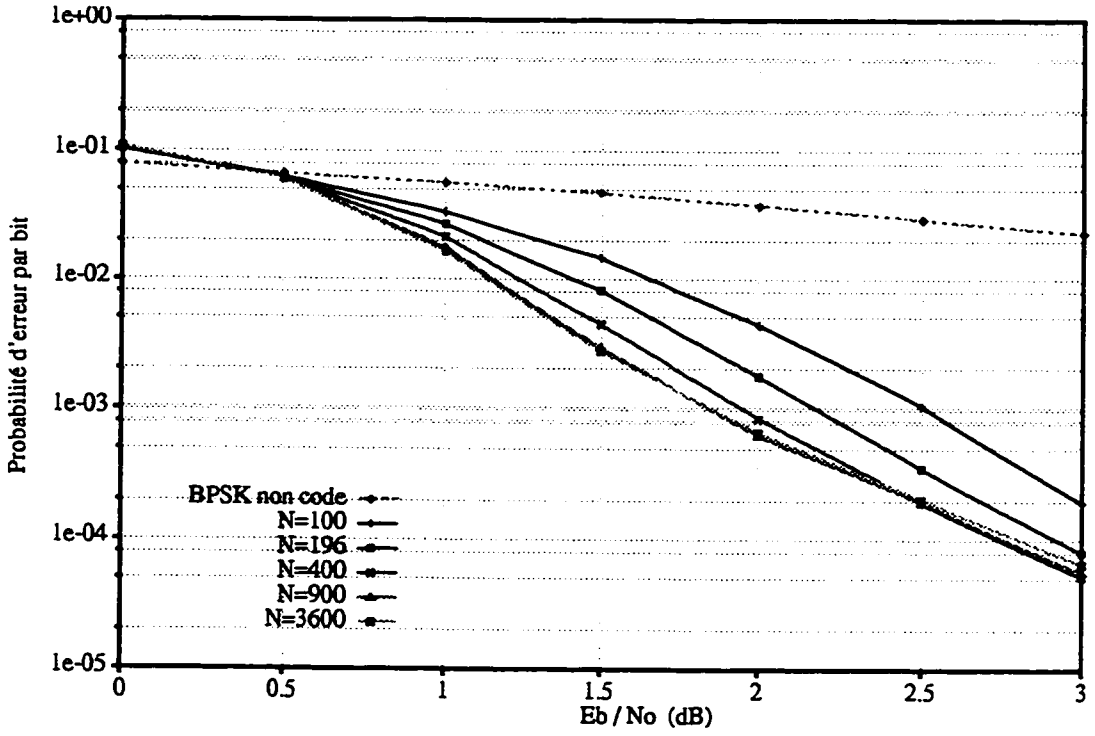


Figure 5.11 Effet de l'augmentation de la longueur de l'entrelaceur bloc sur la probabilité d'erreur pour R=1/2, K=3, G=(1, 5/7), 6 itérations

Toutefois, si N est faible (inférieur à 200 bits), alors la relation $(N_{libre}\tilde{w}_{libre}) << N$ n'est plus vérifiée même si l'entrelaceur utilisé est pseudo-aléatoire. Les deux types d'entrelacement se retrouverait alors à pied d'égalité, avec un avantage de simplicité cependant pour l'entrelacement bloc. Sur les figures 5.12 à 5.15, nous avons représenté les résultats de simulation, pour des blocs de longueur N=100 et N=196, obte-

nus avec un entrelaceur pseudo-aléatoire et un entrelaceur bloc. Les codes utilisés sont de longueur de contrainte K=3 et de générateurs G=(1, 5/7), et le taux de codage global est R=1/2.

Ainsi, pour des blocs de longueur N=196 bits, les performances obtenues avec les entrelaceurs bloc et pseudo-aléatoire sont très similaires, sinon identiques. Plus encore, pour N=100 l'entrelaceur bloc donne de meilleurs résultats. Ce qui se produit en fait est que plus la longueur des blocs est faible, moins l'entrelacement aléatoire est efficace: la probabilité que deux bits voisins le restent après entrelacement est inversement proportionnelle à N. Une étude de l'influence du choix de l'entrelaceur sur les performances des codes turbo utilisant des blocs de 192 bits a été faite dans [24]. Après avoir testé un nombre interminable d'entrelaceurs pseudo-aléatoires, les auteurs sont arrivés à la conclusion qu'un entrelaceur bloc était plus approprié pour la transmission de blocs de faible dimension.

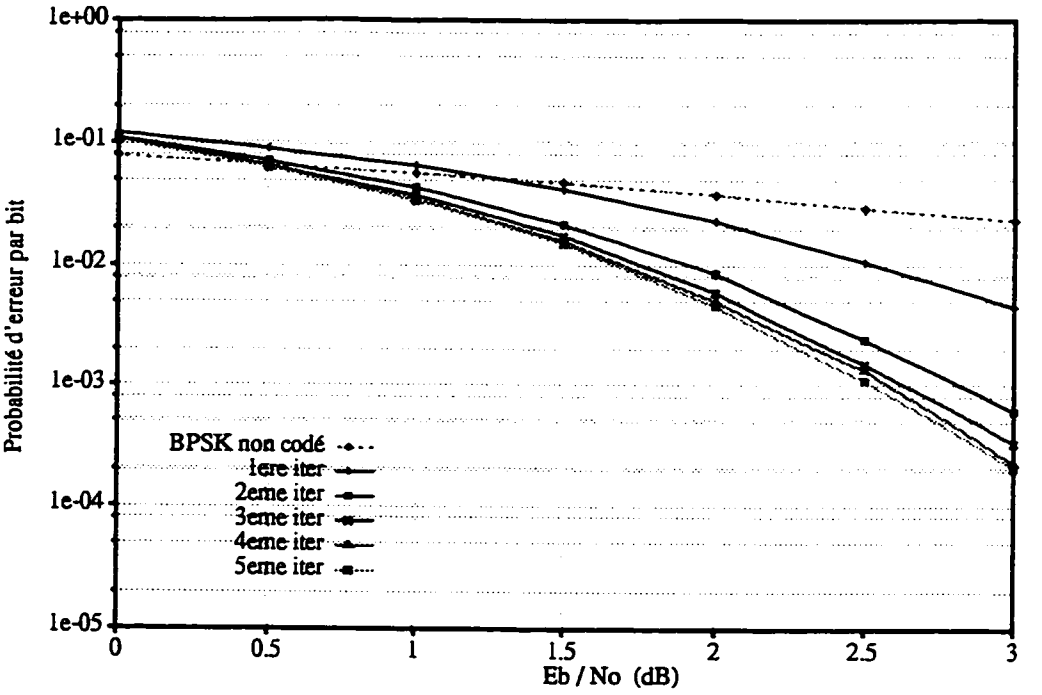


Figure 5.12 Performances d'erreur du code turbo de taux R=1/2, avec K=3, G=(1, 5/7), N=100, Entrelaceur bloc

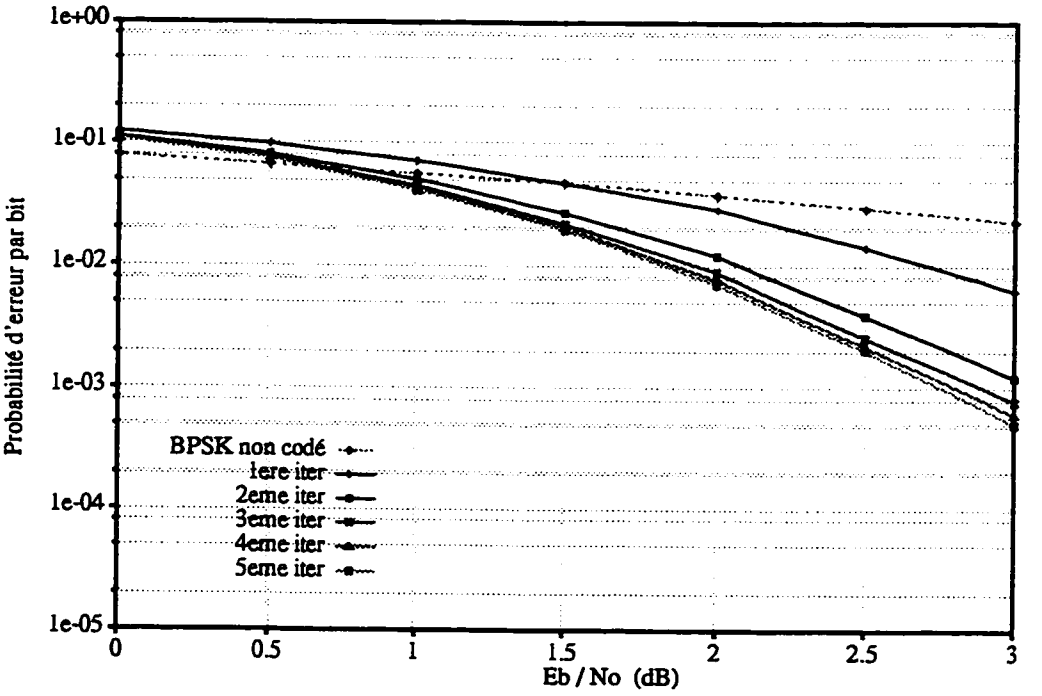


Figure 5.13 Performances d'erreur du code turbo de taux R=1/2, avec K=3, G=(1, 5/7), N=100, Entrelaceur aléatoire

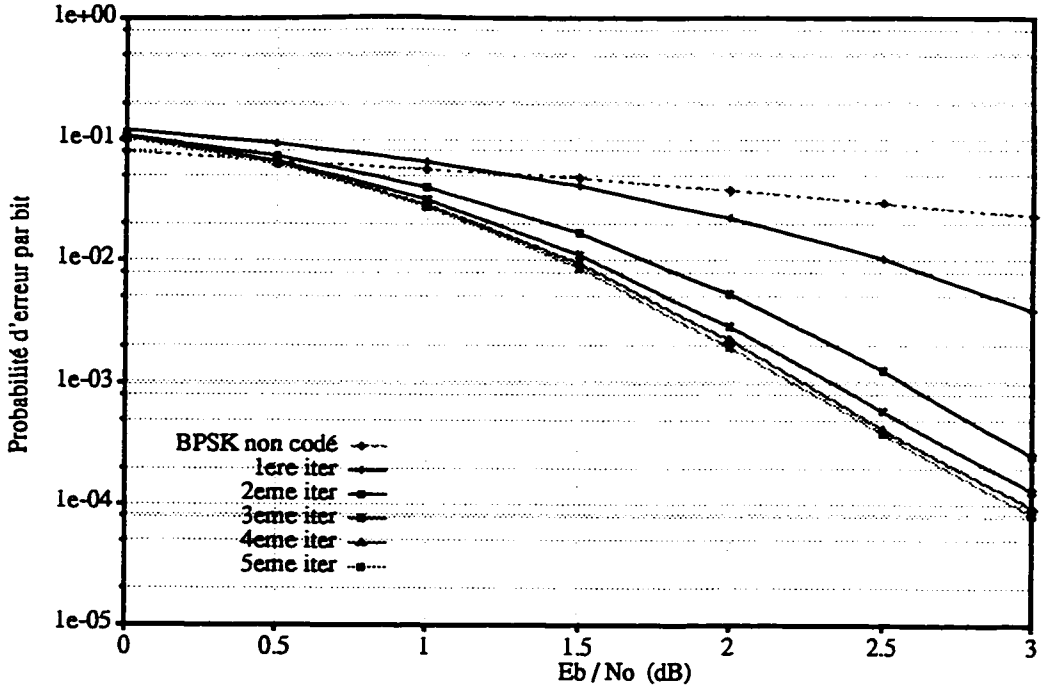


Figure 5.14 Performances d'erreur du code turbo de taux R=1/2, avec K=3, G=(1, 5/7), N=196, Entrelaceur bloc

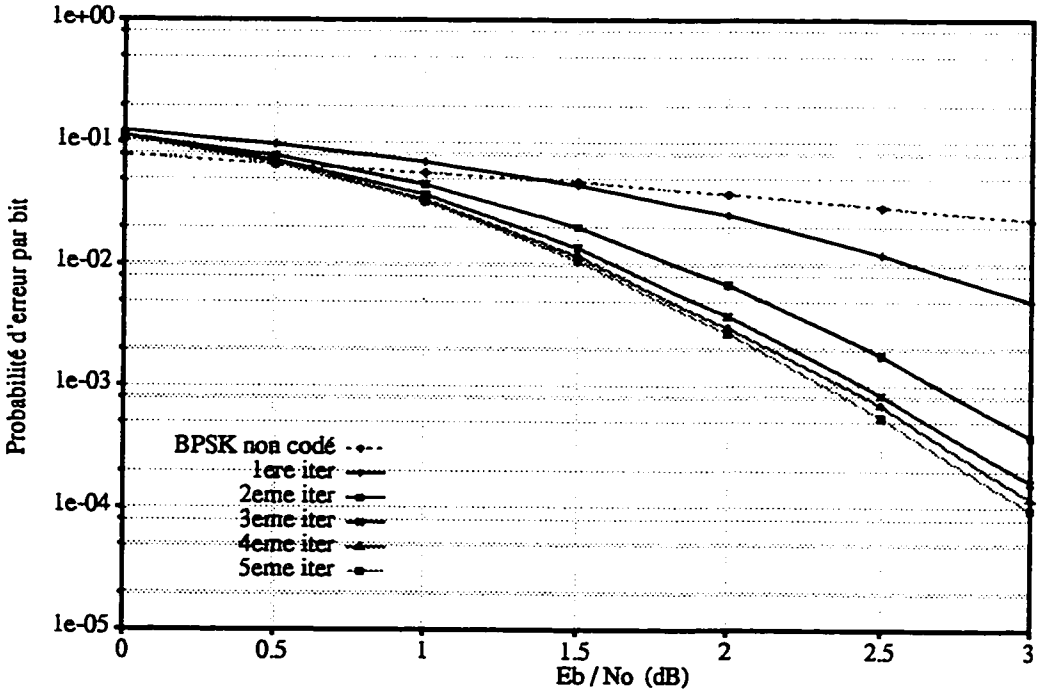


Figure 5.15 Performances d'erreur du code turbo de taux R=1/2, avec K=3, G=(1, 5/7), N=196, Entrelaceur aléatoire

5.6. Autres résultats

L'annexe 1 comporte une série de résultats de simulation des codes turbo pour des taux de codage globaux R=1/2 et R=1/3. Les codeurs élémentaires utilisés ont des longueurs de contraintes K=3 et K=5. Pour le cas K=5, les générateurs G=(1, 21/37) et G=(1, 27/31) ont été utilisés; quant aux codeurs avec K=3, seul G=(1, 5/7) a été employé. La longueur des blocs varie de N=100 à N=900, avec quelques simulations à N=3600 bits pour fins de comparaison. Pour toutes les simulations présentées, le décodage a été effectué en 5 ou 6 itérations, pour des valeurs du RSB allant de 0 à 3dB.

A partir des résultats présentés dans l'annexe 1, nous pouvons noter les points suivants:

1. Pour les longueurs de blocs examinées (N<1000 bits), 5 à 6 itérations pour le décodage s'avèrent suffisantes pour des valeurs du RSB variant de 1,5 à 3dB. D'ailleurs à 3dB, il est rarement nécessaire d'aller au delà de 4 itérations.
2. Pour certaines simulations à R=1/2 et K=5, nous avons représenté les résultats obtenus avec les deux générateurs cités. La différence que nous avions mentionnée au paragraphe 5.4.1 apparait alors clairement (figures A1.1.10 à A1.1.19).
3. Le passage de K=3 à K=5 implique un gain d'autant plus important que la longueur des blocs est grande. Un gain de 0,7dB est ainsi observé à 10^{-5} sur les figures A1.1.7 et A1.1.19 pour des blocs de 900 bits. Ce gain se voit réduit à 0,4dB pour des blocs de 400 bits (figures A1.1.6 et A1.1.17). Toutefois, lorsque la longueur des blocs devient trop petite, c'est le contraire qui se produit: sur les figures A1.1.4 et A1.1.11 par exemple, les performances avec des blocs de 196 bits sont très similaires avec K=3 et K=5. Pour des blocs de 100 bits enfin, des codeurs de longueur de contrainte K=3 donnent un gain d'environ 0,2dB à 10^{-4} par rapport à ceux avec K=5. Cet inversement de situation est dû au fait que, comme nous l'avions montré au chapitre 4, une longueur minimale des blocs, de plus de vingt fois la longueur de contrainte des codeurs élémentaires, est recommandée afin

que la décorrélation des entrées des décodeurs MAP puisse se faire convenablement.

4. Le choix du meilleur type d'entrelacement dépend de la longueur des blocs. L'entrelaceur bloc nécessitant moins de mémoire, constitue ainsi un meilleur choix pour des longueurs de bloc de moins de 200 bits.

5. L'augmentation de la longueur des blocs permet d'améliorer les performances de façon incontestable. Cependant, dépassée une certaine longueur, la comparaison devient inappropriée. Par exemple, pour N=3600, les points de fonctionnement d'intérêt se situent plutôt entre 1 et 2dB; alors qu'à 3dB, la phase de saturation est bien entamée et il devient plus avantageux alors d'employer des blocs plus courts.

6. En comparant les performances obtenues sur la figure A1.1.7 pour K=3, N=900 et A1.1.16 pour K=5, N=400, nous pouvons noter qu'elles sont similaires. Or nous savons que l'augmentation de K implique un accroissement de la complexité, sans toutefois occasionner un délai important; tandis que l'augmentation de N engendre une latence due au décodage itératif sans pour autant qu'il n'y ait augmentation de la complexité. Il est à noter que la complexité dans ce contexte est évaluée en terme de nombre d'opérations, et en considérant que la taille de la mémoire n'est pas un facteur déterminant. Par conséquent, les choix de la longueur de contrainte et de la dimension des blocs dépendent des contraintes auxquelles l'on doit se soumettre.

7. Tous les bits de parité étant transmis dans le cas R=1/3, le gain apporté par la première itération de décodage est plus important que dans le cas R=1/2; si bien que déjà au terme de la deuxième itération, un gain moyen d'environ 0,5 dB est obtenu par rapport à R=1/2.

8. La diminution du taux de codage de R=1/2 à R=1/3 constitue une autre alternative à l'amélioration des performances sans qu'il n'y ait ni accroissement de la complexité, ni apparition d'un délai supplémentaire. A titre d'exemple, des performances très comparables sont obtenues avec les trois combinaisons suivantes:

R=1/3, K=5, N=200 (figure A1.2.10); R=1/2, K=5, N=400 (figure A1.1.18) et R=1/2, K=3, N=900 (figure A1.1.7).

5.7 Comparaison aux codes convolutionnels

Les premiers résultats publiés relativement aux codes turbo étaient tellement exceptionnels qu'aucune comparaison à n'importe quelle autre classe de codes ne pouvaient être faite. Ces résultats paraissaient d'autant plus invraisemblables que les codes utilisés dans la concaténation formant le codeur turbo étaient relativement très petits.

Une interprétation des hautes performances des codes turbo malgré la simplicité des codes élémentaires utilisés est donnée dans [29]. Les auteurs démontrent alors, à travers une comparaison avec les codes convolutionnels, que la supériorité des codes turbo est due au fait que leur spectre des distances est épars et que les performances sont dominées par l'asymptote de la distance libre (équation (5.1)). Seulement, ces résultats n'ont été vérifiées que pour des blocs très longs. La question qui se pose alors est: est-ce que les codes turbo restent avantageux pour des longueurs de blocs plus faibles?

Une comparaison des codes turbo aux codes convolutionnels, basée sur leurs complexités respectives, a été effectuée dans une récente publication, pour des blocs dont les longueurs varient entre 48 et 288 bits [26]. Les auteurs sont alors arrivés à la conclusion que pour des blocs dont la longueur est inférieure à 200 bits, utilisés dans les systèmes de communication mobile, les codes convolutionnels constituent un meilleur choix, si la complexité est considérée. Nous donnons dans la suite une série d'arguments qui nous poussent à contredire cette assertion.

1. Les résultats présentés dans [26], sont basés sur l'évaluation du nombre d'opérations effectuées dans un étage de décodage des codes convolutionnels et des codes turbo. Il est à noter cependant qu'il n'a pas été tenu compte des simplifica-

tions qui peuvent être apportées à l'algorithme MAP, ce qui aurait permis de réduire de façon non négligeable le nombre d'opérations effectuées.

2. Le nombre d'opérations ne dépend, dans le cas des codes convolutionnels, que de la longueur de contrainte; tandis que dans le cas des codes turbo, il est également fonction du nombre d'itération effectuées. Or, la comparaison a été faite sur la base de 10 itérations de décodage, nombre inutilement élevé pour les dimensions de blocs considérées.

3. Les codes élémentaires utilisés ont une longueur de contrainte K=5. Or comme nous l'avons déjà démontré, pour des blocs de moins de 200 bits de tels codeurs donnent de moins bons résultats que leurs congénères de longueur de contrainte plus faible. Par exemple, les performances que nous avons obtenues avec des codeurs à 4 états et un taux de codage global R=1/3, après 6 itérations (figure A1.2.2), sont meilleures que celles obtenues par les auteurs avec K=5 pour le même taux de codage, après 10 itérations. La complexité des codes turbo se verrait alors considérablement diminuée, et les conclusions des auteurs sont alors remises en question.

4. Un entrelaceur aléatoire a été utilisé pour toutes les longueurs de blocs considérées. Encore une fois, ce choix n'est pas approprié pour des blocs de moins de 200 bits. L'utilisation d'un entrelaceur bloc, en particulier pour N de l'ordre de 100 bits, aurait sûrement donné de meilleurs résultats.

Ainsi, les codes turbo restent encore avantageux pour de faibles longueurs de blocs, du moins pour le taux de codage R=1/3. Nous n'irons sans doute pas jusqu'à des blocs contenant aussi peu que 48 bits, mais pour N de l'ordre de 100, la concaténation de deux codeurs à 8 ou à 16 états, avec l'utilisation d'un entrelaceur bloc, peut donner des résultats très satisfaisants. Toutefois, lorsque l'on en vient à des applications en temps réel, telles que les communications mobiles, le délai causé par le décodage itératif doit être considéré, quoique pour des dimensions de blocs aussi faibles, ce délai devrait être tolérable.

5.8 Conclusion

Nous avons montré dans ce chapitre l'importance des rôles que jouent les codes élémentaires et l'entrelaceur, en vue de l'optimisation des performances des codes turbo. En admettant le fait que les codes élémentaires doivent être récursifs et systématiques, quatre paramètres régissent le comportements des codes turbo:

1. *La longueur de contrainte des codes élémentaires.* Son augmentation permet généralement d'obtenir de meilleures performances, mais au prix d'un accroissement de la complexité. Nous estimons cependant non rentable d'aller au delà de K=6. Si la dimension des blocs est inférieure è 200 bits, alors des longueurs de contrainte de 3 ou 4 sont recommandées.
2. *Les générateurs des codes élémentaires.* Leur variation permet d'agir sur la distance libre du code concaténé, qui permet à son tour de diminuer ou d'augmenter la pente du seuil de saturation que connaissent les codes turbo à partir d'un TEB donné. Le choix des générateurs dépend également de la longueur des blocs et du point de fonctionnement du code.
3. *La longueur de l'entrelaceur.* Egale à la longueur des blocs, son augmentation permet d'améliorer très rapidement les performances, mais engendre un délai en conséquence. L'optimisation de la longueur N de l'entrelaceur dépend encore une fois du point de fonctionnement: il est inutile par exemple d'utiliser des blocs de 10.000 bits si un TEB de 10^{-4} à E_b/N_0=2dB est requis. Par ailleurs, le choix de N domine d'autant plus les performances que le RSB est faible. En effet, à partir d'une certaine valeur de E_b/N_0, les performances cessent de s'améliorer, et l'augmentation de N n'apporterait alors qu'un délai supplémentaire.
4. *Le type d'entrelacement.* Il constitue le paramètre le plus simple à fixer: l'entrelaceur aléatoire donne toujours des résultats nettement supérieur, excepté le cas où la dimension des blocs devient très faible (moins de 200 bits environ), auquel cas l'entrelaceur bloc constitue un meilleur choix.

Le problème qui se pose avec les codes turbo est alors, comme nous pouvons le constater, que les paramètres du système sont tellement liés qu'il n'est point évident de dire que tel ou tel autre choix est le meilleur. Tout dépend en fait des critères que l'on se fixe, notamment le point de fonctionnement, les contraintes de complexité, et les limites de délai.

Afin de limiter les délais causés par le décodage itératif, nous avons considéré tout au long de ce mémoire des blocs dont la longueur est inférieure à 1000 bits. Nous avons alors démontré que les codes turbo restent encore avantageux pour cet ordre de grandeur des blocs à de faibles RSB, à condition de choisir convenablement les différents paramètres du système. Nous avons également prouvé par simulation que la remise à zéro des deux codeurs élémentaires s'avère d'autant plus importante que la dimension des blocs est faible.

Au terme de cette analyse, il apparait que la diversité des moyens qui permettent d'agir sur les performances des codes turbo conduit à un décodage flexible et de taux variable. En effet, par la simple variation de la dimension N de l'entrelaceur, les TEB atteints peuvent varier de façon considérable. D'un autre côté, en fixant N et en variant la longueur de contrainte K, les mêmes variations des performances peuvent être observées. Enfin, si N et K sont fixés, la variation du taux de codage peut encore agir sur les performances. C'est d'ailleurs ce dernier point que nous examinons au prochain chapitre, où nous présentons les codes turbo de taux variable.

Chapitre 6
CODES TURBO DE TAUX VARIABLES

6.1 Introduction

La supériorité des performances des codes turbo, du moins pour les taux de codage R=1/2 et R=1/3, ne fait aujourd'hui aucun doute. Il a été par ailleurs démontré dans ce mémoire que même pour de faibles dimensions de blocs, les résultats obtenus avec cette nouvelle classe de codes restent très avantageux, notamment par rapport aux codes convolutionnels.

Nous nous proposons dans ce chapitre d'examiner les performances des codes turbo pour des taux de codage R>1/2. Le but de cette étude est de vérifier si l'efficacité du décodage itératif est diminuée par l'augmentation du taux de codage. Nous nous limiterons cependant à l'examen de certains résultats de simulation, sans envisager l'optimisation des codes turbo de taux élevés.

Ce chapitre commence par la présentation du principe de perforation des codes convolutionnels. Une méthode pour l'augmentation du taux de codage des codes turbo est ensuite proposée. Enfin, l'analyse de quelques résultats, que nous comparons à ceux obtenus avec les codes convolutionnels, est présentée.

6.2 Perforation des codes convolutionnels

Le besoin sans cesse croissant d'augmenter les débits de transmission, tout en gardant un haut niveau de performance ainsi qu'une largeur de bande acceptable, a conduit à la recherche de bons codes de taux élevés $R=b/V$, de même que de techniques appropriées de codage et de décodage. En effet, l'application des algorithmes de décodage

connus à des codes de taux élevés s'avère très difficile à cause de l'accroissement rapide de la complexité à mesure que R augmente.

La perforation des codes convolutionnels a été proposée par Cain *et al.* pour le décodage par l'algorithme de Viterbi [11], et adapté au décodage séquentiel par Haccoun et Bégin [3,17]. Cette technique consiste à perforer la sortie d'un codeur convolutionnel de faible taux de codage, en en éliminant périodiquement certains symboles codés. Le code de taux élevé résultant de la perforation dépend ainsi du code de faible taux, appelé *code d'origine*, ainsi que du nombre et de la position des symboles codés éliminés. Le patron des symboles perforés est appelé *patron de perforation* du code perforé, et est décrit par une matrice appelée *matrice de perforation*.

Considérons la construction d'un code convolutionnel perforé de taux $R=b/V$, à partir d'un code d'origine de taux $R=1/V_0$. Pour chaque bV_0 symboles générés par le code d'origine et correspondant à b bits d'information, $S=(bV_0 - V)$ symboles sont éliminés suivant le patron de perforation choisi. Le taux de codage résultant est alors $R=b/(bV_0 - S)$, égal au taux $R=b/V$ visé. La variation du code d'origine de faible taux et du patron de perforation permet alors d'obtenir un large éventail de codes de taux élevés [17].

Le patron de perforation est exprimé par une matrice de V_0 lignes et b colonnes, dont les éléments prennent les valeurs 0 et 1, correspondant respectivement à la perforation ou non d'un symbole donné, généré par le code d'origine. La variation du nombre et de la valeur des éléments de la matrice de perforation permet alors d'obtenir différents codes avec divers taux de codage.

Le décodage des codes perforés se fait en utilisant les mêmes algorithmes employés pour les codes d'origines à partir desquels ils ont été construits. Ces algorithmes sont alors adaptés à la perforation sans que leur complexité ne soit augmentée.

6.3 Perforation des codes turbo

Afin d'augmenter le taux de codage global R, initialement égal à 1/3, un commutateur a été placé à la sortie du codeur turbo de sorte que pour chaque bit d'information codé, un seul symbole de parité, généré alternativement par l'un ou l'autre des deux codeurs élémentaires, était transmis. Le taux de codage global résultant devint alors R=1/2, tout en conservant le caractère systématique du code.

Cette façon de procéder est en fait équivalente à l'utilisation à la sortie du codeur d'un module de perforation dont le patron correspond à la matrice

$$P = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

La première ligne de la matrice correspond à la sortie systématique du codeur; tous ses éléments sont donc égaux à *1* afin de préserver le caractère systématique du code après perforation. Les lignes 2 et 3 quant à elles, correspondent respectivement aux symboles de parité générés par Codeur 1 et Codeur 2. Le zéro sur chacune de ces lignes indique la position du symbole codé qui ne sera pas transmis. La figure 6.1 illustre le séquencement des sorties du codeur turbo de taux R=1/2.

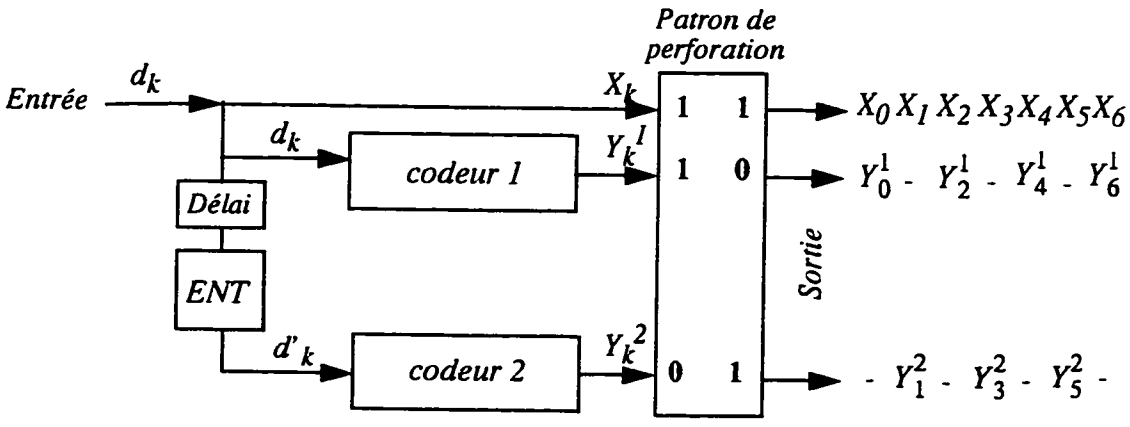


Figure 6.1 Séquencement des sorties du codeur turbo de taux R=1/2

A la réception des séquences codées, les symboles non transmis sont remplacés par des zéros.

Afin d'augmenter le taux de codage à des valeurs supérieures à 1/2, la technique la plus simple serait donc de généraliser la méthode utilisée pour R=1/2, par l'utilisation de différents patrons de perforation. Comme les codes que l'on obtiendrait pour les différentes valeurs de R sont construits à partir du même code d'origine de taux R=1/3, on parle alors de codes de taux variables. Le codeur résultant est représenté sur la figure 6.2.

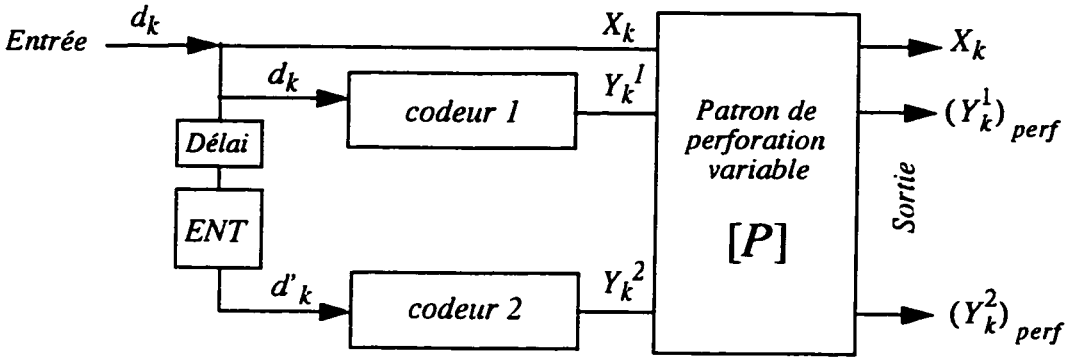


Figure 6.2 Structure du codeur turbo de taux variable

La seule condition que doit remplir la matrice P est que la première ligne soit formée uniquement de *1*, afin que le code résultant demeure systématique. Il est à noter cependant que cette exigence implique que le taux $R=b/V$ doit remplir la condition $b \leq V-2$. En effet, comme les b bits d'information doivent être transmis et que chaque codeur devrait transmettre au moins un symbole codé, alors V est au moins égal à $b+2$. Pour un taux de codage R=6/8 par exemple, obtenu à partir du code turbo de départ de taux R=1/3, un patron de perforation pouvant être utilisé est exprimé par la matrice

$$P = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

6.4 Analyse des résultats de simulation

Dans l'annexe 2 nous avons rapporté une série de résultats obtenus par simulation des codes turbo perforés, construits à partir du même code turbo d'origine de taux R=1/3 résultant de la concaténation de deux codeurs CRS de taux R=1/2 chacun et dont la longueur de contrainte est K=3 ou K=5. Les blocs utilisés sont toujours de longueur N<1000 bits et les taux de codage testés sont R=4/6, R=6/8, R=8/10 et R=14/16. Nous avons également représenté les patrons de perforation utilisés, choisis parmi plusieurs patrons testés, pour avoir donné les meilleures performances. La recherche du meilleur patron n'est cependant pas exhaustive.

Un premier examen des performances obtenues permet de constater une nette dégradation des performances à mesure que R augmente. Cette dégradation était prévisible et théoriquement justifiée par la diminution de la distance libre. Afin de mesurer cette baisse de performance, nous avons représenté sur les figures 6.3 à 6.5 respectivement les résultats obtenus avec les longueurs de blocs N=169, N=400 et N=900 bits pour des taux de codage variant de R=1/2 à R=14/16.

Il apparait alors que pour des TEB entre 10^{-3} et 10^{-5}, une dégradation variant entre 2 et 3dB est obtenue en passant de R=1/2 à R=14/16. Afin de mesurer l'importance de cette dégradation, nous avons représenté sur la figure 6.6 les bornes sur la probabilité d'erreur par bit pour des codes convolutionnels perforés de taux équivalents, construits à partir d'un code d'origine de taux R=1/2 et de longueur de contrainte K=7.

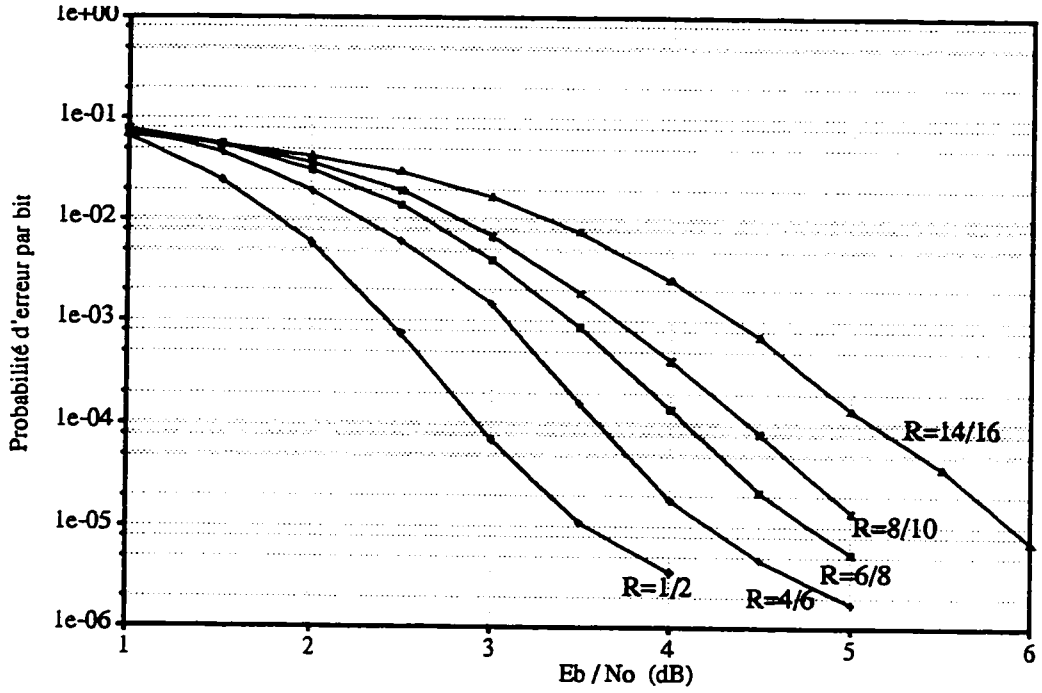


Figure 6.3 Performances d'erreur du code turbo avec K=5, G=(1, 27/31), N=169, entrelaceur aléatoire, 6 itérations

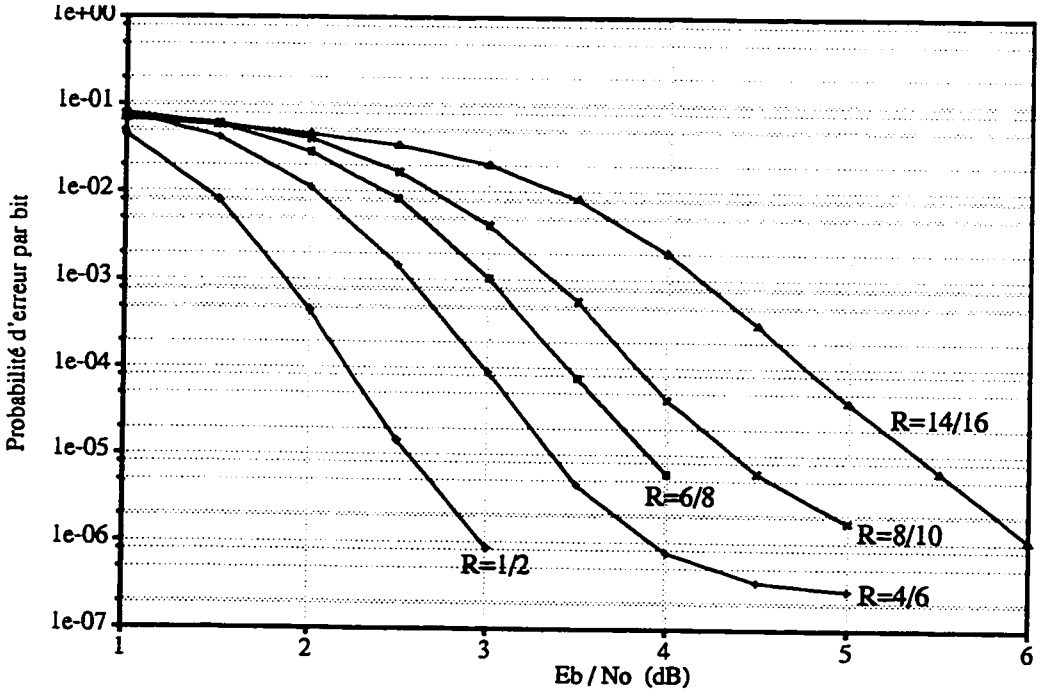


Figure 6.4 Performances d'erreur du code turbo avec K=5, G=(1, 27/31), N=400, entrelaceur aléatoire, 6 itérations

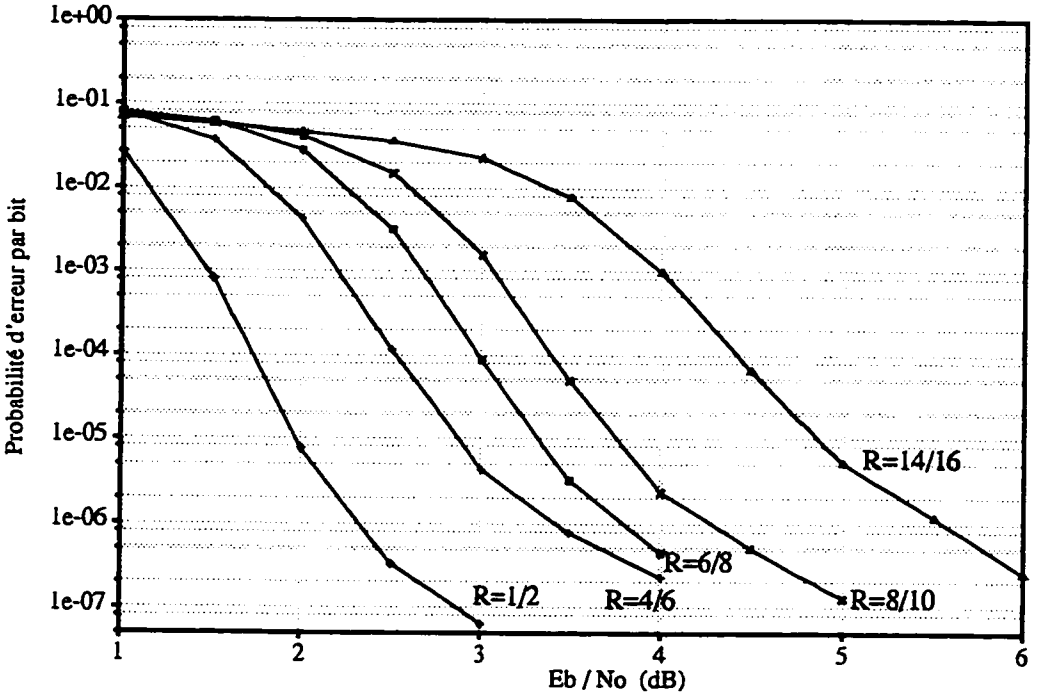


Figure 6.5 Performances d'erreur du code turbo avec K=5, G=(1, 27/31), N=900, entrelaceur aléatoire, 6 itérations

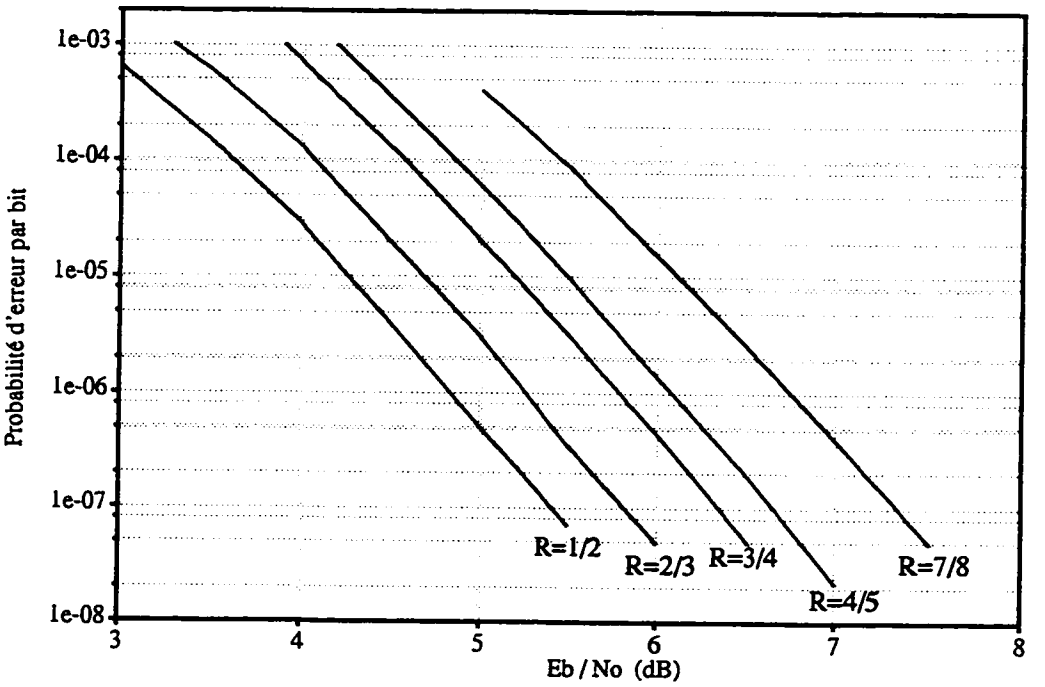


Figure 6.6 Bornes supérieures sur les probabilités d'erreur par bit pour des codes convolutionnels perforés à partir d'un code d'origine de taux R=1/2, sur un canal gaussien non quantifié.

Nous pouvons ainsi noter à partir de la figure 6.6 que la dévaluation des performances des codes convolutionnels, en passant de R=1/2 à R=7/8, est plutôt constante, et vaut environ 2dB. Les codes turbo seraient donc plus vulnérables à l'augmentation du taux de codage. Toutefois, cela ne signifie pas encore qu'ils ont cessé d'être plus avantageux que les codes convolutionnels; cela dépendra du gain obtenu avant perforation.

La question n'a évidemment pas lieu si les blocs considérés sont de longueur très grande, puisque la différence des performances est tellement importante dans ce cas, que même en perdant 2dB de plus que les codes convolutionnels à cause de la perforation, les performances des codes turbo restent nettement supérieures. Toutefois, dans le cas où la dimension des blocs est plutôt faible, nous pouvons déjà prévoir que la longueur minimale N_{min} des blocs pour laquelle les codes turbo restent avantageux, et

qui valait environ 100 bits pour R=1/3, devra probablement être quelque peu augmentée.

Les figures 6.7 et 6.8 illustrent les performances obtenues avec les codes turbo perforés de taux R=4/6 et R=6/8, pour des longueurs de blocs variant de N=81 à N=900 bits. La longueur de contrainte des codes élémentaires est K=5. Nous avons également joint sur ces figures les bornes supérieures sur les probabilités d'erreur par bit pour les codes convolutionnels perforés de taux équivalents (R=2/3 et R=3/4) et de longueur de contrainte K=11. Cette valeur de K a été choisie en nous référant à l'évaluation des complexités des codes turbo et des codes convolutionnels effectuée dans [26], et selon laquelle, si la longueur de contrainte des codes élémentaires est K=5, alors les codes turbo resteraient avantageux après 6 à 8 itérations s'ils donnaient de meilleures performances qu'un code convolutionnel à 1024 états au moins.

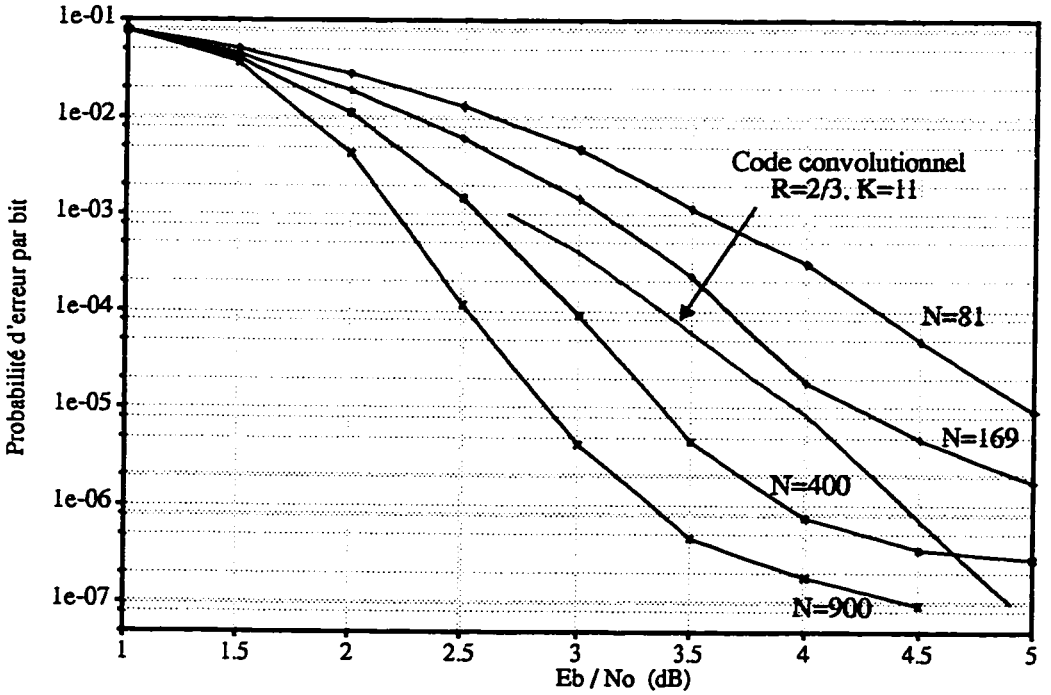


Figure 6.7 Performances d'erreur du code turbo perforé de taux R=4/6, avec K=5, G=(1, 27/31), 6 itérations

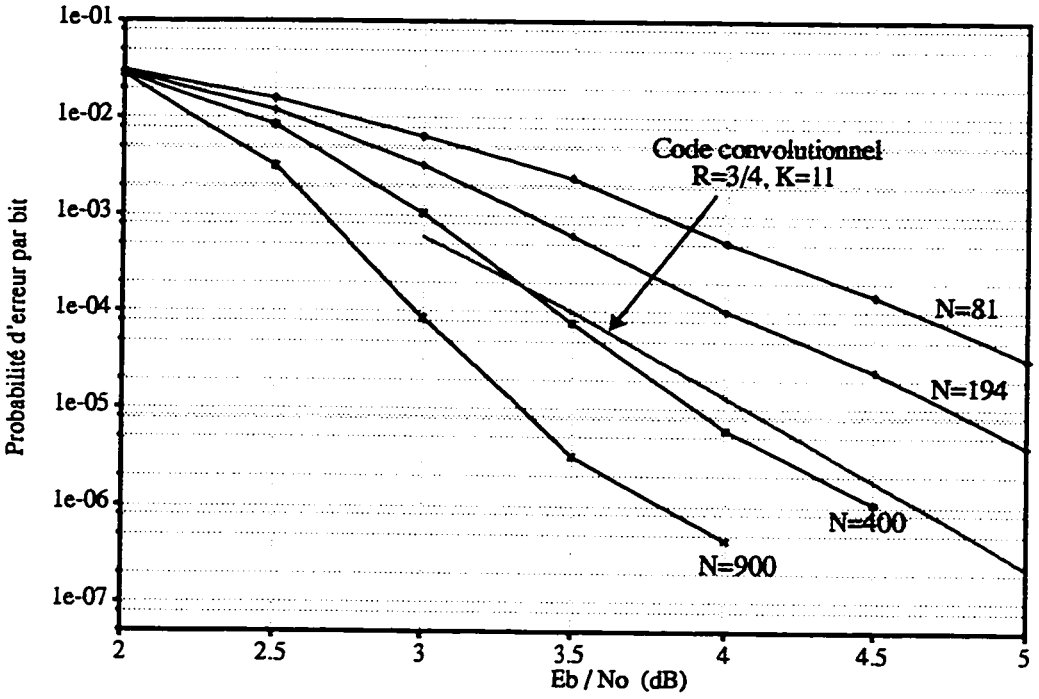


Figure 6.8 Performances d'erreur du code turbo perforé de taux R=6/8, avec K=5, G=(1, 27/31), 6 itérations

D'après la figure 6.7 nous constatons que pour des longueurs de blocs inférieures à 400 bits, les performances que devraient donner les codes convolutionnels de taux R=2/3 pour des TEB entre 10^{-3} et 10^{-5} sont meilleures que celles des codes turbo de taux équivalent R=4/6. Lorsque ce taux est augmenté à R=6/8, alors la longueur minimale des blocs se voit aussi augmentée: des blocs de plus de 400 bits sont dans ce cas nécessaires.

Rappelons cependant que ces comparaisons ne s'appliquent que pour des valeurs du TEB comprises dans une certaine plage, qui dépend de l'ordre de grandeur de la dimension des blocs considérés. En effet, si les TEB sont trop élevés, relativement à la longueur des blocs, alors l'amélioration des performances risque d'être encore trop lente; tandis que si l'on dépasse une certaine valeur du TEB, alors les performances atteignent la phase de saturation. Dans ces deux derniers cas, les codes convolutionnels peuvent alors fournir des performances supérieures à celles des codes turbo.

6.5 Conclusion

Nous avons proposé dans ce chapitre une technique pour la construction de codes turbo de taux élevés $R>1/2$. L'analyse des performances obtenues par simulation a montré que les codes turbo perforés continuent à donner de meilleurs résultats que les codes convolutionnels perforés de taux de codage équivalents; et ce, malgré une plus grande dégradation due à la perforation. Toutefois, le dimension minimale requise pour les blocs, valant 100 bits environ pour $R=1/3$, est d'autant plus grande que R augmente. Un minimum de plus de 400 bits par bloc est par exemple nécessaire pour un taux de codage R=6/8.

Par ailleurs, Il est à noter que les résultats présentés dans ce chapitre n'ont fait l'objet d'aucune optimisation, notamment pour le choix des codes élémentaires, nous fiant plutôt à la règle "*de bons codes d'origine donnent de bons codes perforés*"; et que la comparaison aux codes convolutionnels a été faite par rapport à des bornes supérieures sur les probabilités d'erreur par bit, à défaut de résultats de simulation. La considération de ces deux points pourrait donc conduire à une légère amélioration des performances.

La perforation des codes turbo constitue ainsi un autre moyen permettant de faire varier les performances obtenues. Cette technique peut s'avérer intéressante dans le cas par exemple où le TEB désiré varie en fonction du type d'information à transmettre, ou encore en fonction des bits dans un même bloc. Il suffirait alors de réduire le taux de codage pour avoir des TEB plus faibles.

D'autres techniques pour l'augmentation du taux de codage des codes turbo pourraient être envisagées, telles que par exemple la perforation de codes résultant de la concaténation de plus de deux codeurs élémentaires.

Chapitre 7

CONCLUSIONS ET PERSPECTIVES DE RECHERCHE

Nous avons présenté dans ce mémoire une étude portant sur les codes turbo, nouvelle technique de correction d'erreur, reconnue pour son haut niveau de performance dans des canaux très bruités.

Au chapitre 2 nous avons présenté la concaténation parallèle des codeurs convolutionnels formant le codeur "turbo", et décrit le principe de fonctionnement du décodage itératif effectué par le décodeur turbo, constitué par la concaténation en série de deux décodeurs élémentaires. Nous avons également montré le rôle que joue l'entrelaceur lors du codage et du décodage turbo.

L'algorithme MAP, utilisé pour le décodage turbo, a été introduit au chapitre 3. Adapté aux codes convolutionnels systématiques, cet algorithme a permis d'obtenir d'excellentes performances lorsqu'utilisé dans un système itératif. Il présente cependant trois inconvénients majeurs: d'abord une complexité élevée; ensuite, la nécessité d'une parfaite connaissance du canal par le récepteur; et enfin, la présence d'une latence causée par une boucle de calcul récursif à rebours.

Deux simplifications importantes ont permis de réduire la complexité de l'algorithme de façon considérable, la ramenant à environ quatre fois celle de l'algorithme de Viterbi. Ces simplifications consistaient respectivement à modifier les définitions des métriques d'état et de transition, et à exprimer l'algorithme dans le domaine logarithmique.

La comparaison des algorithmes MAP et SOVA a montré que ce dernier donnaient de moins bonnes performances, mais qu'il était en revanche de complexité plus faible

que celle de l'algorithme MAP.

Les rouages du décodeur turbo ont été examinés au chapitre 4. Nous avons alors montré qu'afin d'adapter l'algorithme MAP au décodage itératif, les rapports de vraisemblance qu'il génère doivent être décomposés en une somme, dont l'un des termes correspond à l'information extrinsèque. Seule cette information est transférée d'un décodeur MAP à l'autre.

Grâce à l'utilisation d'un entrelaceur, l'échange d'information entre les décodeurs MAP constituant le décodeur turbo peut se faire sans corrélation, permettant ainsi d'apporter une amélioration des performances à chaque itération. L'utilisation d'un entrelaceur pseudo-aléatoire permet d'obtenir les meilleurs performances des codes turbo, à condition cependant que la dimension des blocs ne soit pas trop faible, auquel cas un entrelaceur bloc constitue un meilleur choix.

Le principal inconvénient que présente le décodage itératif des codes turbo consiste en un délai proportionnel au nombre d'itérations effectuées; nombre d'autant plus grand que les blocs utilisés sont longs et que les TEB visés sont faibles. Pour des blocs de moins de 1000 bits, un nombre moyen de 6 itérations est généralement suffisant, mais le délai engendré peut, malgré cela, entraîner l'élimination de certains domaines d'application.

L'analyse des résultats de simulation des codes turbo au chapitre 5 a démontré l'importance des choix des codes élémentaires, et du type et de la longueur de l'entrelaceur, pour l'obtention des meilleures performances. Nous avons ainsi montré que l'augmentation de la faible distance libre des codes turbo, obtenue par un choix adéquat des générateurs des codes élémentaires, permettait d'obtenir des TEB plus faibles à des RSB modérés à élevés. Pour de faibles valeurs du RSB cependant, l'amélioration des performances est dominée par l'augmentation de la dimension des blocs.

La comparaison des performances des codes convolutionnels de taux R=1/3 à celles des codes turbo de même taux, utilisant des blocs de faibles dimensions (entre 100 et 200 bits) a permis de constater que même pour ces longueurs de blocs, les codes turbo restent encore avantageux, à condition cependant de choisir convenablement les longueurs de contrainte des codes élémentaires. Nous avons également montré que la réinitialisation des deux codeurs élémentaires était très importante dans ce cas, et qu'un entrelaceur bloc était également recommandé.

Au chapitre 6 nous avons procédé à la perforation des codes turbo, conduisant à des codes de taux variable. Malgré une dégradation des performances qui semble être plus importante que pour les codes convolutionnels de taux équivalents, les codes turbo perforés demeurent très avantageux à des RSB relativement faibles. Toutefois, la dimension minimale requise pour les blocs se voit augmentée à mesure que le taux de codage augmente.

Le décodage itératif de la concaténation parallèle de deux codes convolutionnels récursifs et systématiques à permis de se rendre, pour un TEB=10^{-5}, à 0,7dB de la capacité du canal. Cette performance exceptionnelle n'a pu cependant être obtenue que grâce à l'utilisation de blocs très longs engendrant un délai en conséquence. Néanmoins, la variation des différents paramètres du système, notamment les caractéristiques des codes élémentaires, la longueur et le type de l'entrelaceur, ainsi que le taux de codage global, permet de couvrir un large éventail de performances tout aussi intéressantes.

Plusieurs portes restent ouvertes sur une recherche future, en vue notamment de la diminution de la complexité et du délai que présente le décodage turbo. Il serait ainsi profitable de

1. proposer de nouveaux algorithmes de moindre complexité que l'algorithme MAP, sans pour autant qu'il n'y ait dégradation importante des performances. En parti-

culier, l'algorithme SOVA pourrait être considéré dans le cas des blocs de faibles dimensions;

2. envisager l'utilisation d'autres types de codes dans la concaténation formant le codeur turbo. L'usage de codes en blocs systématiques a été considéré dans [20], où les auteurs soutiennent que les codes en blocs donnent de meilleures performances pour des taux de codage élevés R>2/3. Un développement de ces résultats seraient ainsi intéressant pour les codes turbo perforés;

3. construire de nouveaux types d'entrelaceurs qui permettraient d'obtenir de meilleures performances, notamment pour de faibles longueurs de blocs. Les entrelaceurs blocs permettent d'obtenir des performances comparables, sinon meilleures, que celles obtenues avec les entrelaceurs pseudo-aléatoires, pour de faibles longueurs de blocs. Il serait donc avantageux de proposer d'autres entrelaceurs déterministes qui permettraient d'améliorer encore plus les performances;

4. considérer des techniques de modulation plus élaborées que BPSK, telles que 8 PSK ou 16 QAM. Une étude sur le sujet est publiée dans [27], et un approfondissement dans ce sens nous semble prometteur.

RÉFÉRENCES

1. BAHL, L., COCKE, J., JELINEK, F. et RAVIV, J. (1972). "Optimal decoding of linear codes for minimizing symbol error rate", *IEEE Int. Symp. Inform. Theory*, p. 90, Asilomar, CA.

2. BARBULECU, A. S. et PIETROBON, S. S. (1995). "Terminating the trellis of turbo codes in the same state", *Electronics Letters*, vol. 31-1, pp. 22-23.

3. BÉGIN, G. et HACCOUN, D. (1989). "High-rate punctured convolutional codes: structure properties and construction technique", *IEEE Tans. Comm.*, vol. 37, pp 1381-1385.

4. BENEDETTO, S. et MONTORSI, G. (1996). "Unveiling turbo codes: some results on parallel concatenated coding schemes", *IEEE Trans. Inform. Theory*, vol. 44-5, pp. 409-428.

5. BENEDETTO, S. et MONTORSI, G. (1996). "Design of parallel concatenated convolutional codes", *IEEE Trans. Comm*, vol. 44-5, pp. 591-600.

6. BENEDETTO, S., MONTORSI, G., DIVSALAR, D. et POLLARA, F. (1996). "Soft-output decoding algorithms in iterative decoding of turbo codes", *JPL TDA Progress Report*, vol. 42-124, pp. 63-87.

7. BERROU, C., GLAVIEUX, A. et THITIMAJSHIMA, P. (1993). "Near Shannon limit error correcting coding and decoding: turbo codes", *International Conference on Communications (ICC 93)*, pp. 1064-1070, Genève, Suisse.

8. BERROU, C. et GLAVIEUX, A. (1996). "Near optimum error correcting coding and decoding: turbo codes", *IEEE Trans. Comm.*, vol. 44, pp. 1261-1271.

9. BEZILE, J., (1990). "Algorithmes et architechtures de décodeurs séquentiels", mémoire de maîtrise, Ecole Polytechnique de Montréal.

10. BEZILE, J., (1993). "Décodage sous-optimal des codes convolutionnels et applications", thèse de PhD, Ecole Polytechnique de Montréal.

11. CAIN J. B., CLARK C. et GEIST J. (1979). "Punctured convolutional codes of rate (n - 1)/n and simplified maximum likelihood decoding", *IEEE Trans. Inform. Theory*, vol. IT-25, pp. 97-100.

12. CHANG, R. W. et HANCOCK, J. C. (1996). "On receiver structure for channels having memory", *IEEE Trans. Inform. Theory*, vol. IT-12, pp. 463-468.

13. CHAN, F. et HACCOUN, D. (1991). "Adaptive decoding of convolutional codes", *Proc. Canadian Conf. on Electrical and computer engineering*, 69.4.1-69.4.4, Québec, Canada.

14. ELIAS, P. (1954). "Error-free coding", *IRE Trans. on Inform. theory*, vol. PGIT-4, pp. 29-37.

15. FORNEY, G. D. (1966). "Concatenated codes", *MIT Press*, Cambridge, Mass.

16. FORNEY, G. D. (1970). "Convolutional codes: algebraic stucture", *IEEE Trans. Inform. Theory*, vol. IT-16, pp. 720-738.

17. HACCOUN D. et BEGIN G. (1989). "High-rate puncture convolutional codes for Viterbi and sequential decoding", *IEEE Tran. Comm*, vol 37, pp. 1113-1125.

18. HAGENAUER, J. et HOEHER, P. (1989). "A Viterbi algorithme with soft-decision outputs and its applications", *GLOBECOM 89*, pp. 1680-1686, Dallas, Texas.

19. HAGENAUER, P., ROBERTSON, P. et PAPKE, L. (1994). "Iterative (turbo) decoding of systematic convolutional codes with the MAP and SOVA algorithms", *ITG Tagung*, pp. 21-29, Frankfort, Allemagne.

20. HAGENAUER, J., OFFER, E. et PAPKE, L. (1996). "Iterative decoding of binary bloc and convolutional codes", *IEEE Trans. Inform. Theory*, vol. 42-2, pp. 429-445.

21. IMAI, H. et HIRAKAWA, S. (1977). "A new multilevel coding method using error-correcting codes", *IEEE Trans. on Inform. Theory*, vol. IT-23, pp. 371-377.

22. JUNG, P. (1995). "Novel low complexity decoder for turbo codes", *Electronics Letters*, vol. 31-2, pp. 86-87.

23. JUNG, P. et NABHAN, M. (1994). "Performance evaluation of turbo codes for short frame transmission systems", *Electronics Letters*, vol. 30-4, pp. 111-113.

24. JUNG, P. et NABHAN, M. (1994). "Dependence of the error performance evaluation of turbo codes on the interleaver structure in short frame transmission systems", *Electronics Letters*, vol. 30-2, pp. 287-288.

25. JUNG, P. et NABHAN, M. (1995). "Comprehensive comparison of turbo-code decoders", *IEEE Vehicular Tech. Conf.*, pp. 624-628, Chicago, ILL.

26. KOORAPATY, H., WANG, Y. E. et BALACHANDRAN, K. (1997). "Performance of turbo codes with short frame sizes", *IEEE Vehicular Tech. Conf.*, pp. 329-333, Phoenix, ARIZ.

27. LE GOFFE, S., GLAVIEUX, A. et BERROU, C. (1994). "Turbo codes and high spectral efficiency modulation", *ICC 94*, pp. 645-649.

28. McADAM, P. L., WELCH, L. R. et WEBER, C. L. (1972). "M.A.P. bit decoding of convolutional codes", *IEEE Int. Symp. Inform. Theory*, p. 91, Asilomar, CA.

29. PEREZ, L., SEGHERS, J. et COSTELLO, D. J. (1996). "A distance spectrum interpretation of turbo codes", *IEEE Trans. Inform. Theory*, vol. 42-6, pp. 1698-1709.

30. PIETROBON, S. S. (1996). "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.

31. PIETROBON, S. S. et BARBULESCU, S. A. (1994). "A simplification of the modified Bahl decoding algorithm for systematic convolutional codes", *Int. Symp. Inform. Theory & its applications*, pp. 1073-1077, Sydney, Australie.

32. RIEDEL, S. et SVIRID, Y. V. (1995). "Iterative (turbo) decoding of threshold decodable codes", *European Trans. on Telecomm.* (ITT), accepté pour publication.

33. ROBERTSON, P. (1994). "Illuminating the structure of decoders for parallel concatenated recursive systematic (turbo) codes", *IEEE Globecom conf.*, pp. 1298-1303, San Fransisco, CA.

34. ROBERTSON, P. (1994). "Improving decoder and code structure of parallel concatenated recursive systematic (turbo) codes", *Int. Conf. Universal Personal Commun.*, pp. 183-187, San Diego, CA.

35. ROBERTSON, P., VILLEBRUN, E. et HOEHER, P. (1995). "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain", *ICC 95*, pp. 1009-1013, Seattle, WA.

36. SEGHERS, J. (1995). "On the free distance of turbo codes and related product codes", Rapport de projet de fin d'études, *SS 95-6613*, Swiss Federal Institute of Technology, Zurich, Suisse.

37. Signal Processing Worksystem (SPW), ALTA GROUP of Cadance Design Systems, Inc., 1996.

38. WANG, X. et WICKER, S. B. (1996). "A soft-output decoding algorithm for concatenated systems", *IEEE Trans. Inform. Theory*, vol. 42-2, pp. 543-553.

ANNEXE 1

Résultats de simulation pour les taux de codage R=1/2 et R=1/3

Cet annexe comporte une série de résultats de simulation des codes turbo pour les taux de codage R=1/2 et R=1/3. Les performances sont représentées en termes de la probabilité d'erreur par bit en fonction de E_b/N_0. Les codeurs élémentaires utilisés ont des longueurs de contraintes K=3 et K=5. Pour le cas K=5, les générateurs G=(1, 21/37) et G=(1, 27/31) ont été utilisés; quant aux codeurs avec K=3, seul G=(1, 5/7) a été employé. La longueur des blocs varie de N=100 à N=900, avec quelques simulations avec N=3600 bits pour fins de comparaison. Pour toutes les simulations présentées, le décodage est effectué en 5 où 6 itérations, pour des valeurs de E_b/N_0 variant entre 0 à 3dB.

A1.1 Résultats de simulation pour un taux de codage global

R=1/2

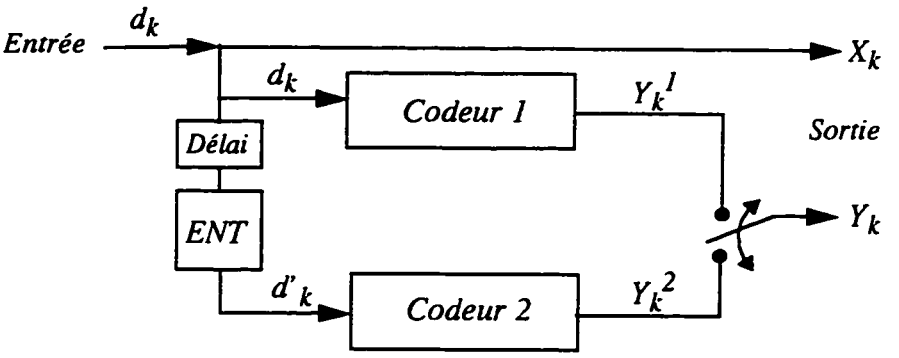


Figure A1.1.1 Structure du codeur turbo

A1.1.1 Codeurs élémentaires K=3, G=(1, 5/7)

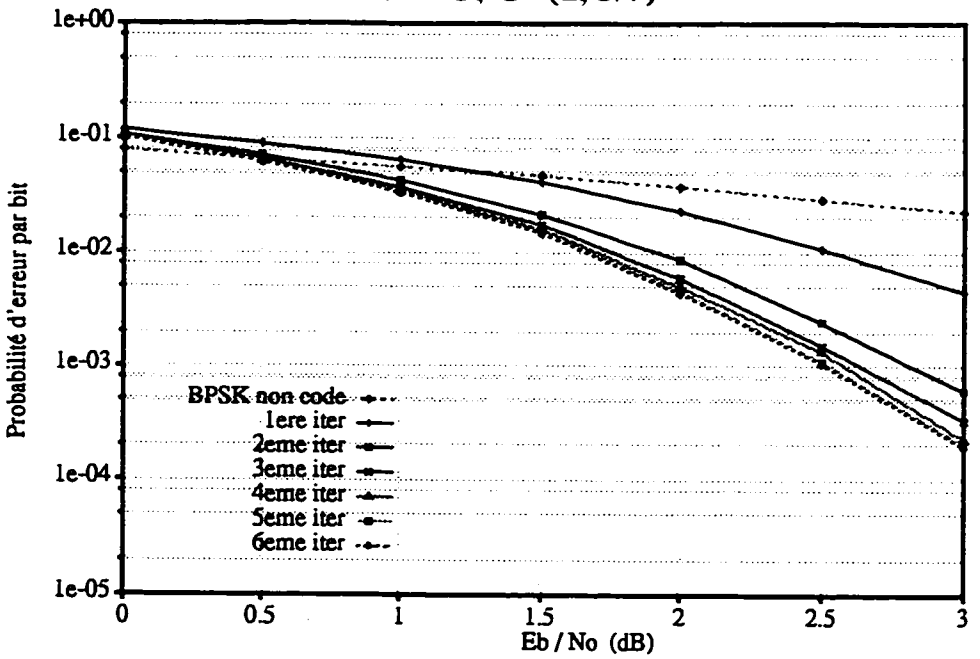


Figure A1.1.2 Performances d'erreur du code turbo de taux R=1/2, avec K=3, G=(1, 5/7), N=100, Entrel. bloc

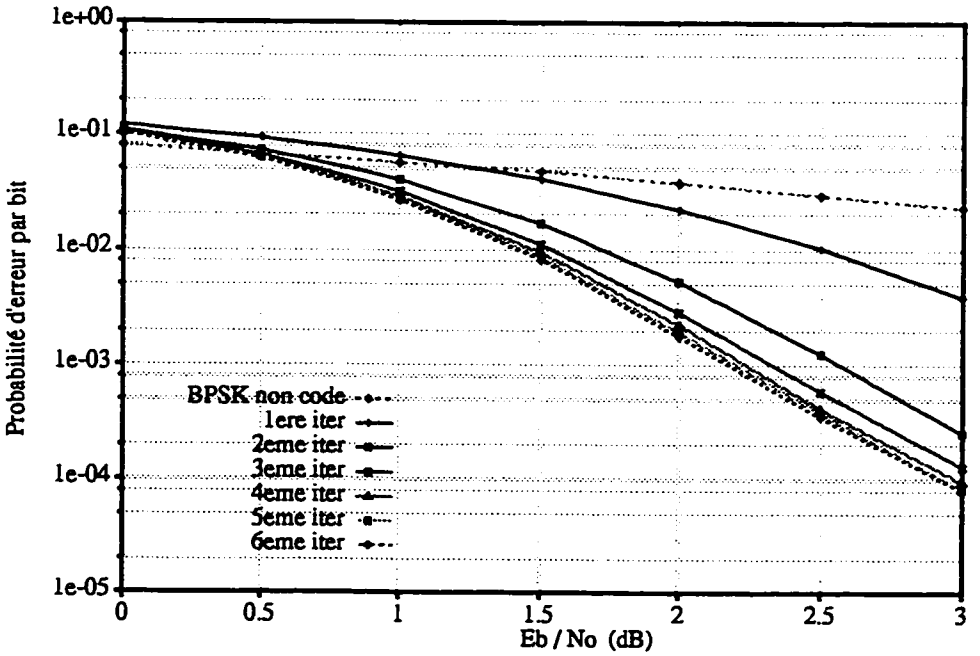


Figure A1.1.3 Performances d'erreur du code turbo de taux R=1/2, avec K=3, G=(1, 5/7), N=196, Entrel. bloc

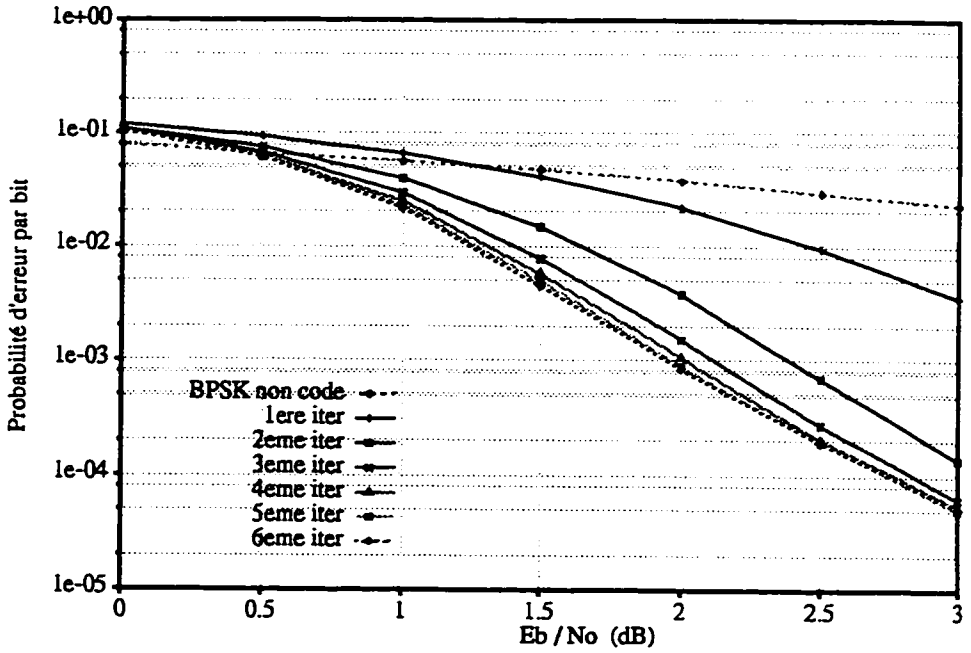


Figure A1.1.4 Performances d'erreur du code turbo de taux R=1/2, avec K=3, G=(1, 5/7), N=400, Entrel. bloc

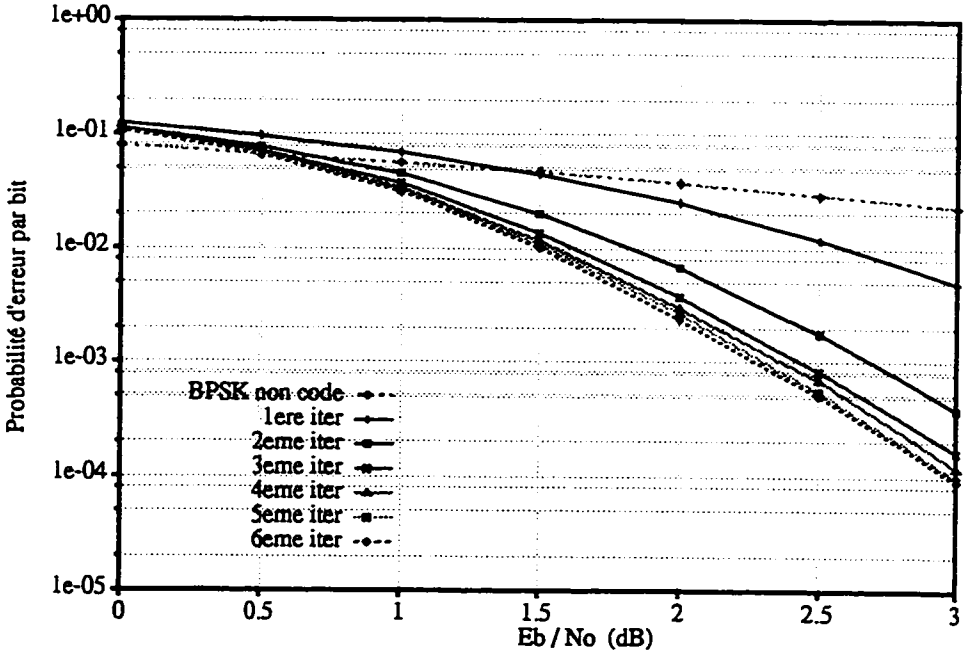


Figure A1.1.5 Performances d'erreur du code turbo de taux R=1/2, avec K=3, G=(1, 5/7), N=196, Entrel. aléatoire

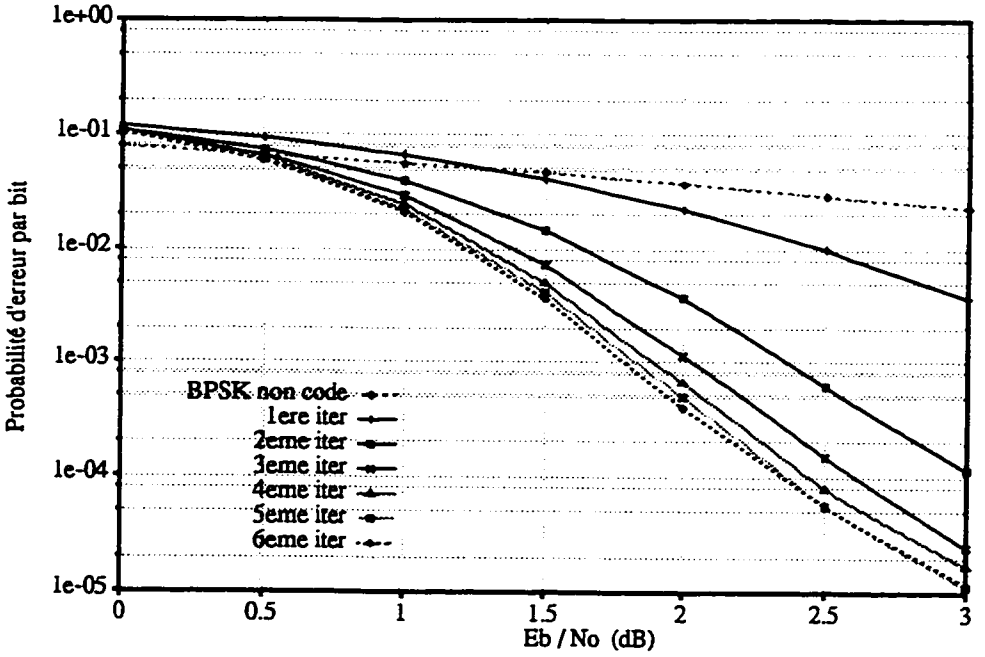


Figure A1.1.6 Performances d'erreur du code turbo de taux R=1/2, avec K=3, G=(1, 5/7), N=400, Entrel. aléatoire

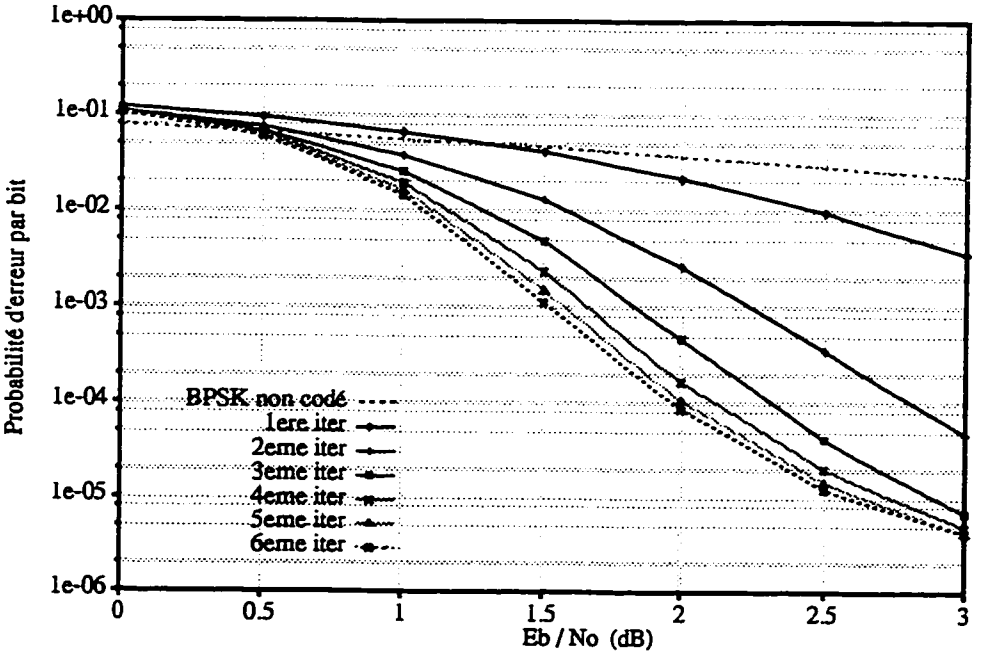


Figure A1.1.7 Performances d'erreur du code turbo de taux R=1/2, avec K=3, G=(1, 5/7), N=900, Entrel. aléatoire

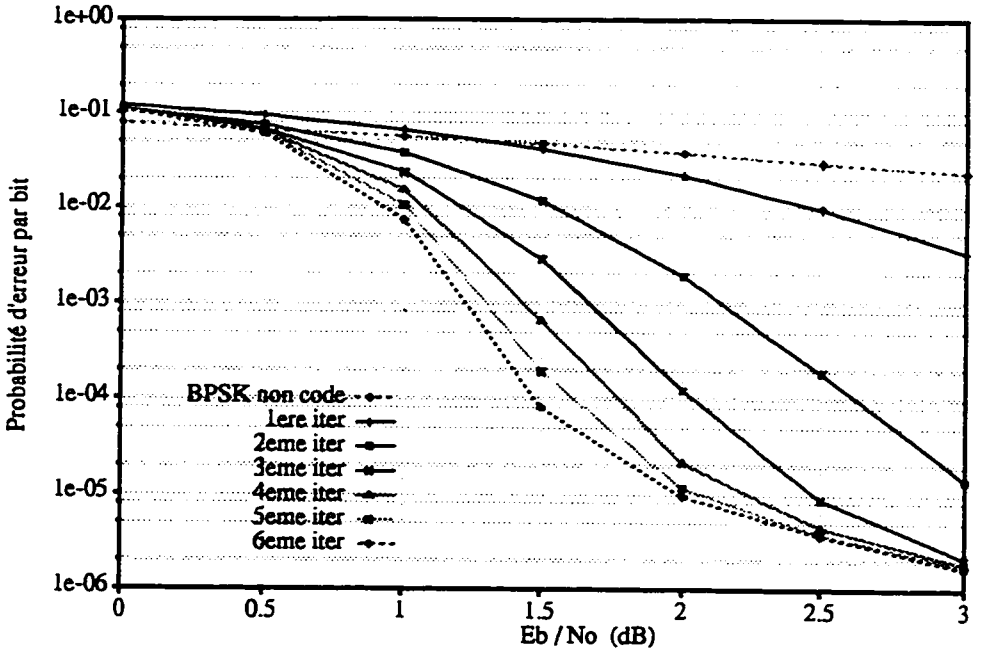


Figure A1.1.8 Performances d'erreur du code turbo de taux R=1/2, avec K=3, G=(1, 5/7), N=3600, Entrel. aléatoire

A1.1.2 Codeurs élémentaires K=5

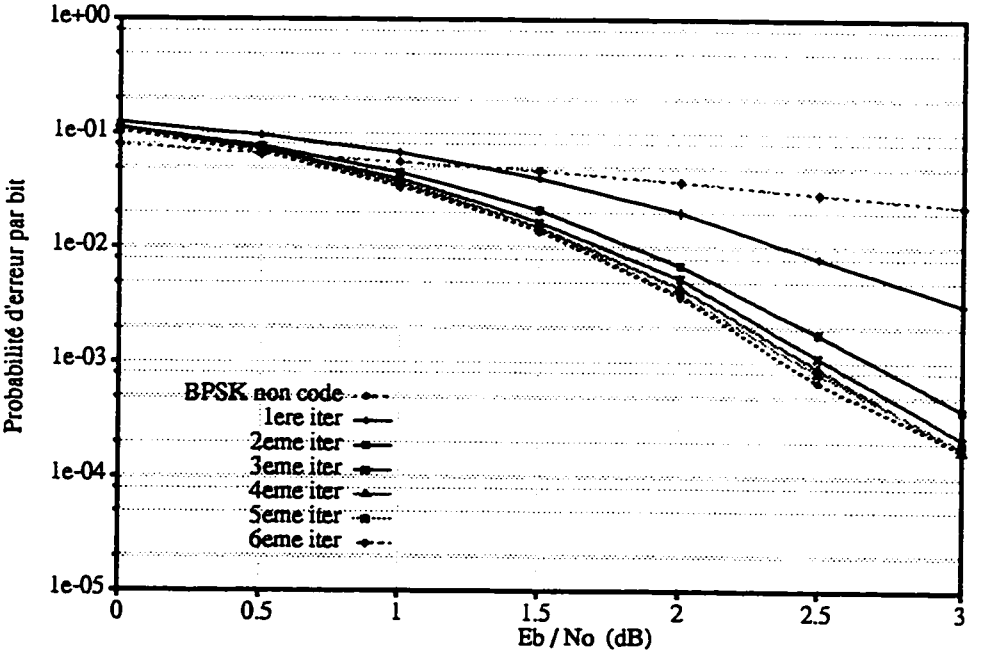


Figure A1.1.9 Performances d'erreur du code turbo de taux R=1/2, avec K=5, G=(1, 21/37), N=100, Entrel. bloc

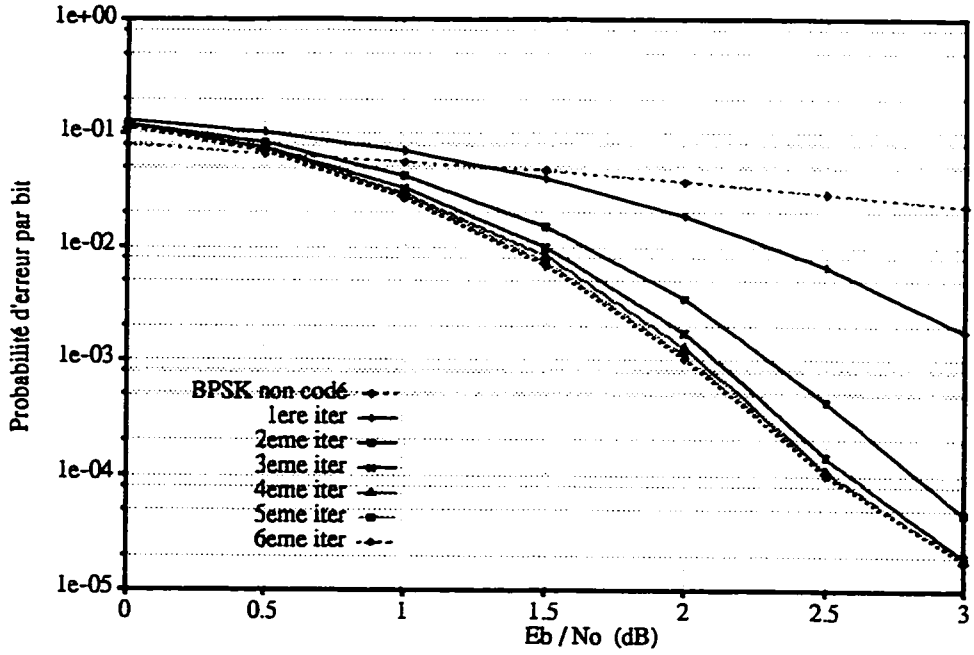


Figure A1.1.10 Performances d'erreur du code turbo de taux R=1/2, avec K=5, G=(1, 21/37), N=196, Entrel. bloc

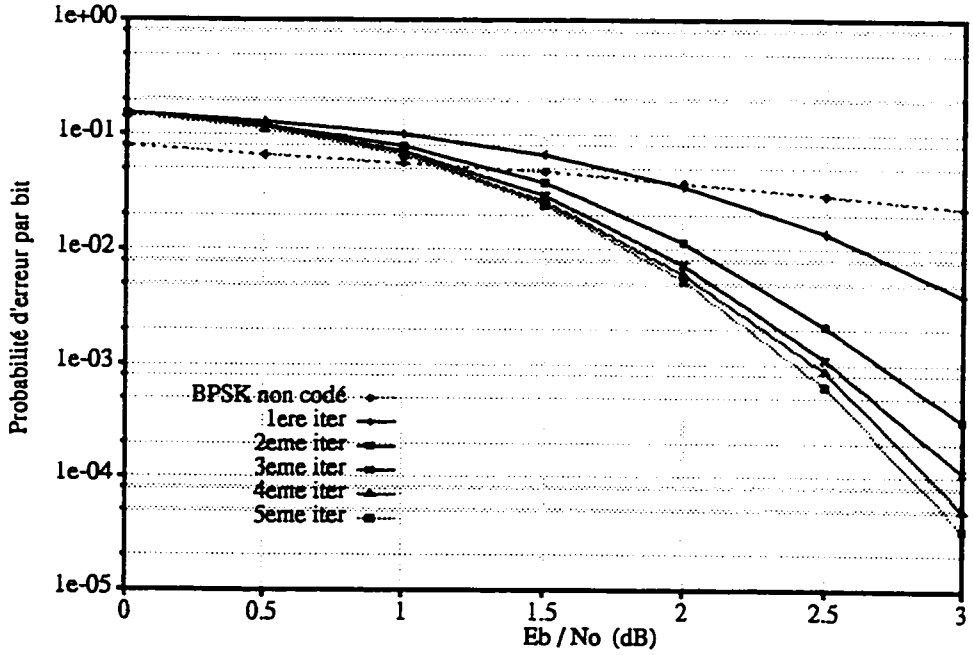


Figure A1.1.11 Performances d'erreur du code turbo de taux R=1/2, R=1/2, K=5, G=(1, 27/31), N=196, Entrel. bloc

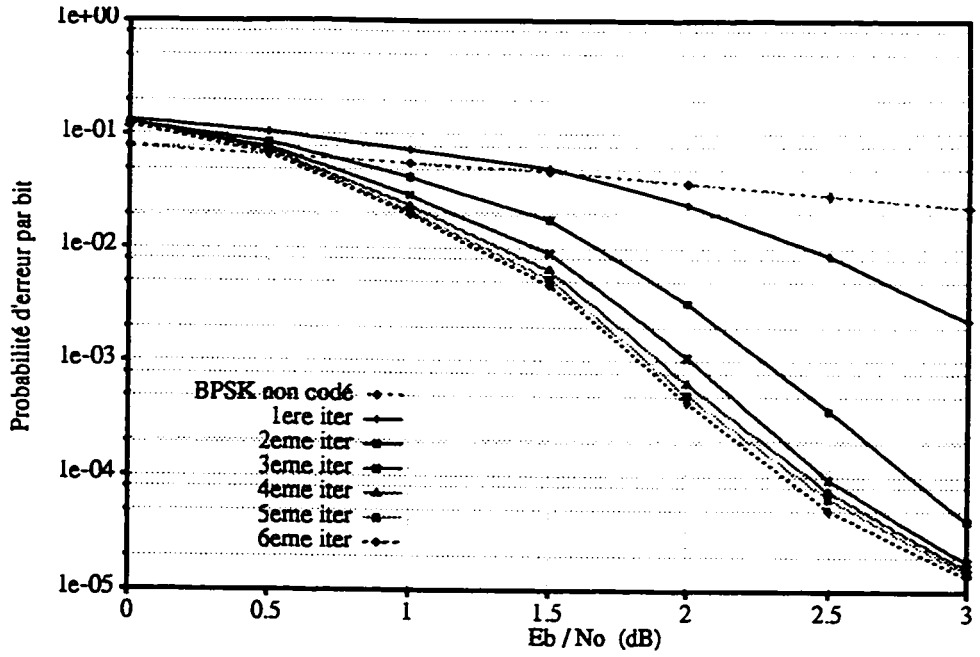


Figure A1.1.12 Performances d'erreur du code turbo de taux R=1/2, avec K=5, G=(1, 21/37), N=400, Entrel. bloc

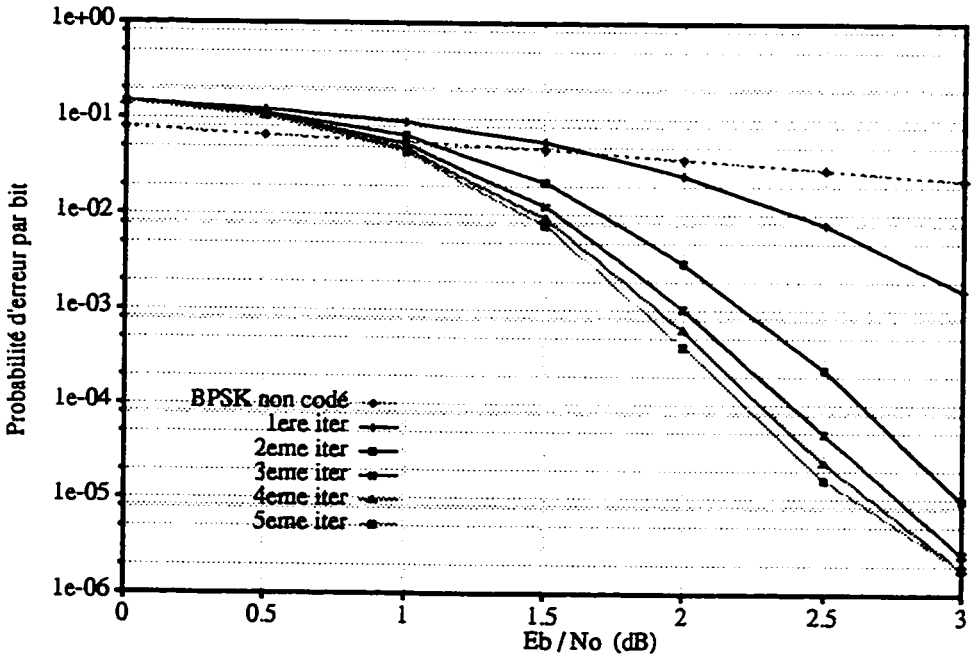


Figure A1.1.13 Performances d'erreur du code turbo de taux R=1/2, avec K=5, G=(1, 27/31), N=400, Entrel. bloc

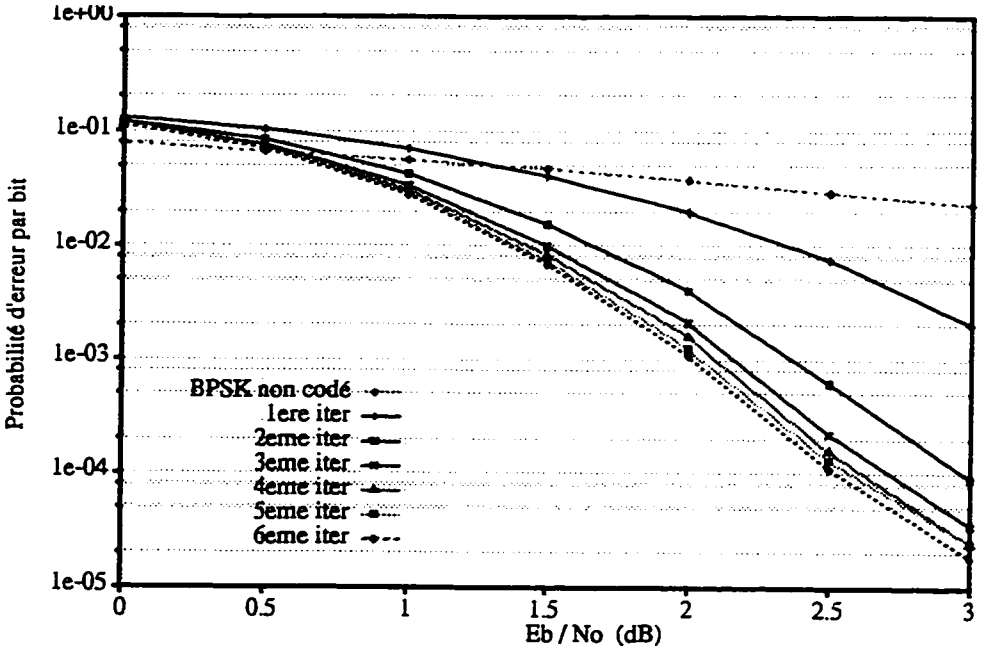


Figure A1.1.14 Performances d'erreur du code turbo de taux R=1/2, avec K=5, G=(1, 21/37), N=196, Entrel. aléatoire

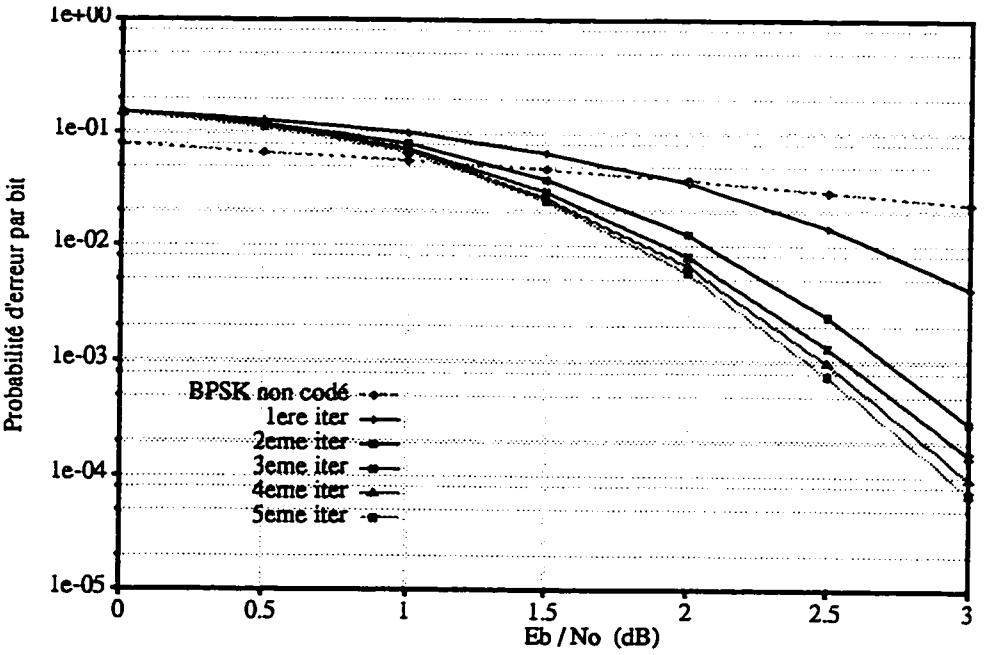


Figure A1.1.15 Performances d'erreur du code turbo de taux R=1/2, avec K=5, G=(1, 27/31), N=196, Entrel. aléatoire

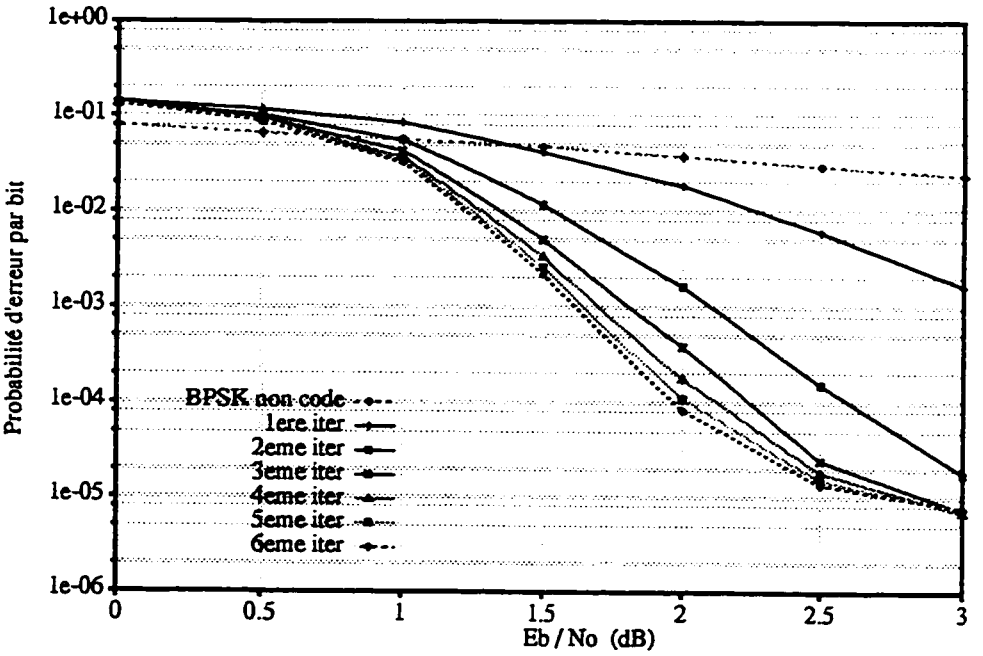


Figure A1.1.16 Performances d'erreur du code turbo de taux R=1/2, avec K=5, G=(1, 21/37), N=400, Entrel. aléatoire

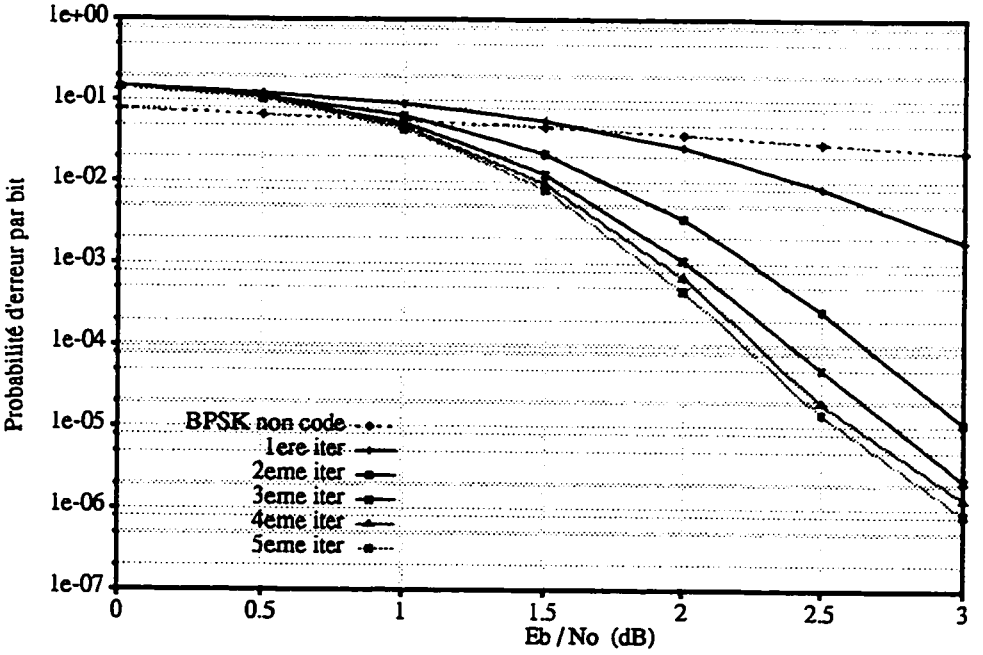


Figure A1.1.17 Performances d'erreur du code turbo de taux R=1/2, avec K=5, G=(1, 27/31), N=400, Entrel. aléatoire

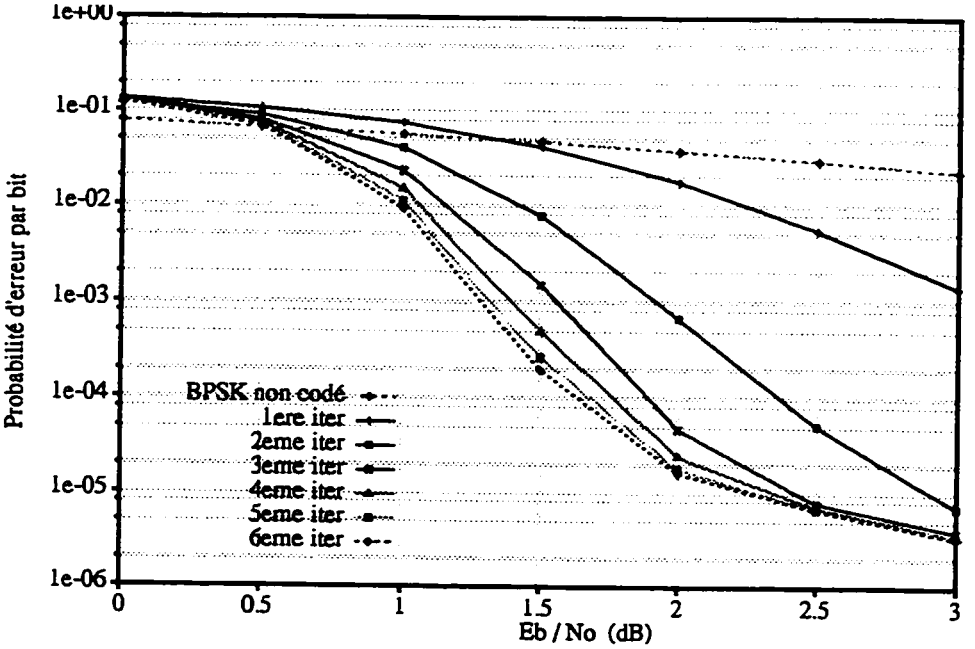


Figure A1.1.18 Performances d'erreur du code turbo de taux R=1/2, avec K=5, G=(1, 21/37), N=900, Entrel. aléatoire

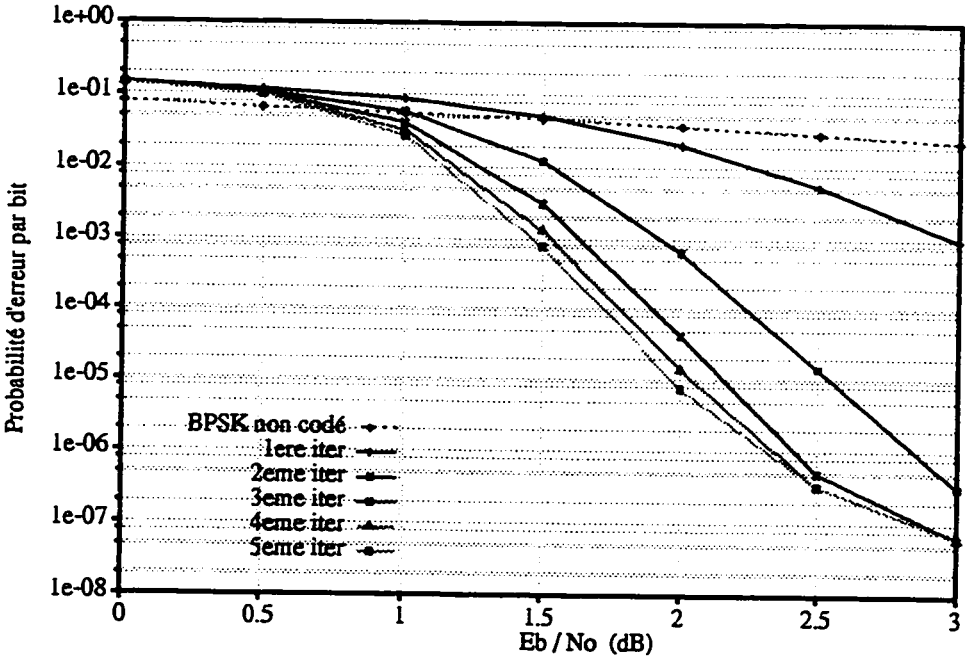


Figure A1.1.19 Performances d'erreur du code turbo de taux R=1/2, avec K=5, G=(1, 27/31), N=900, Entrel. aléatoire

A1.2 Résultats de simulation pour un taux de codage global

R=1/3

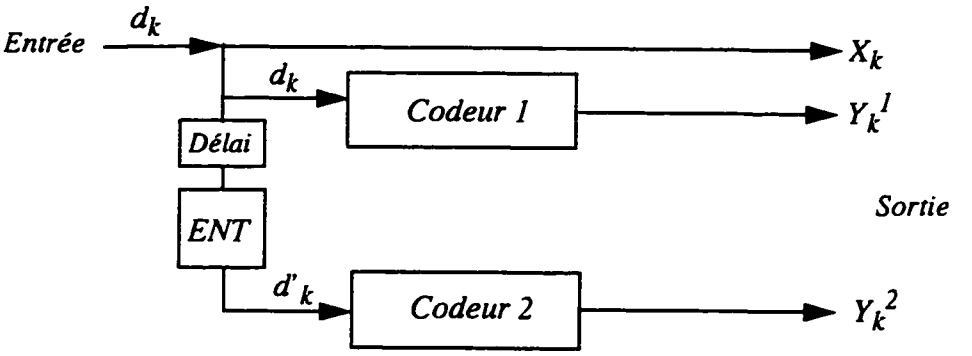


Figure A1.2.1 Structure du codeur turbo

A1.2.1 Codeurs élémentaires K=3 G=(1, 5/7)

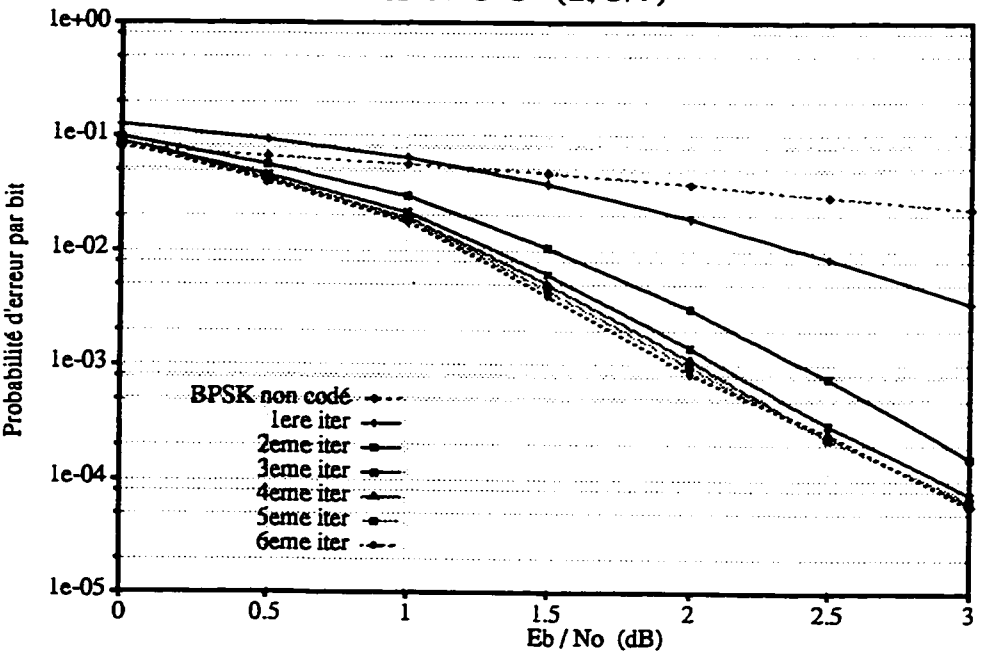


Figure A1.2.2. Performances d'erreur du code turbo de taux R=1/3, avec K=3, G=(1, 5/7), N=100, Entrel. bloc

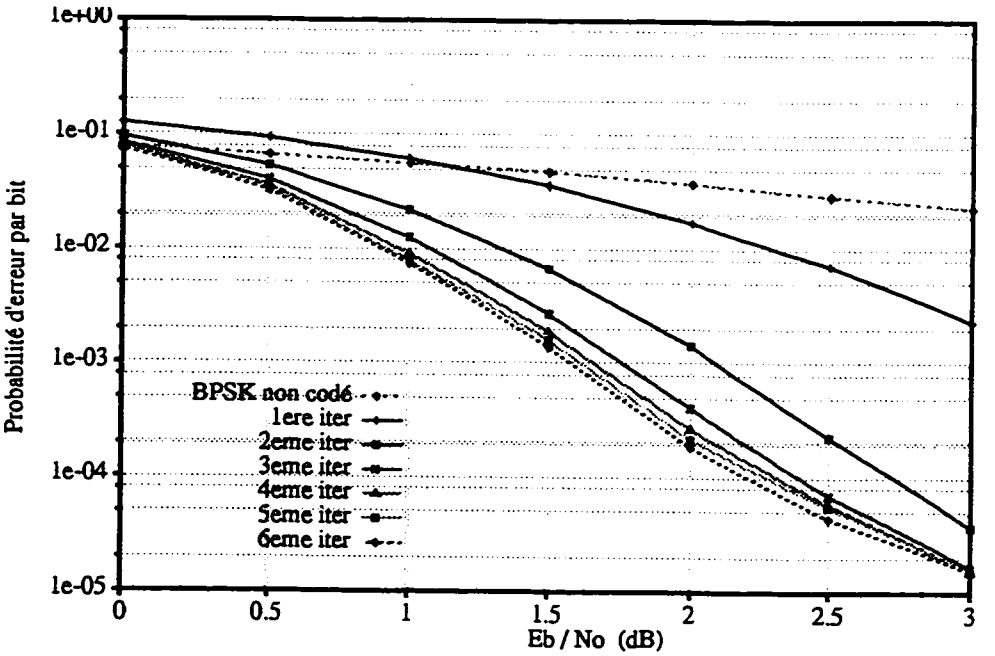


Figure A1.2.3 Performances d'erreur du code turbo de taux R=1/3, avec K=3, G=(1, 5/7), N=196, Entrel. bloc

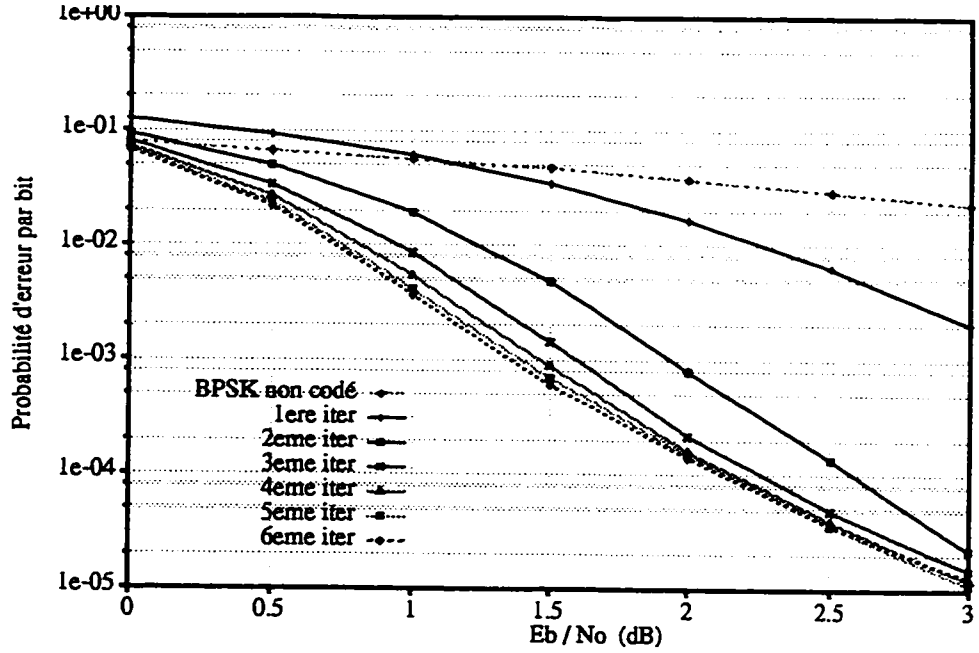


Figure A1.2.4 Performances d'erreur du code turbo de taux R=1/3, avec K=3, G=(1, 5/7), N=400, Entrel. bloc

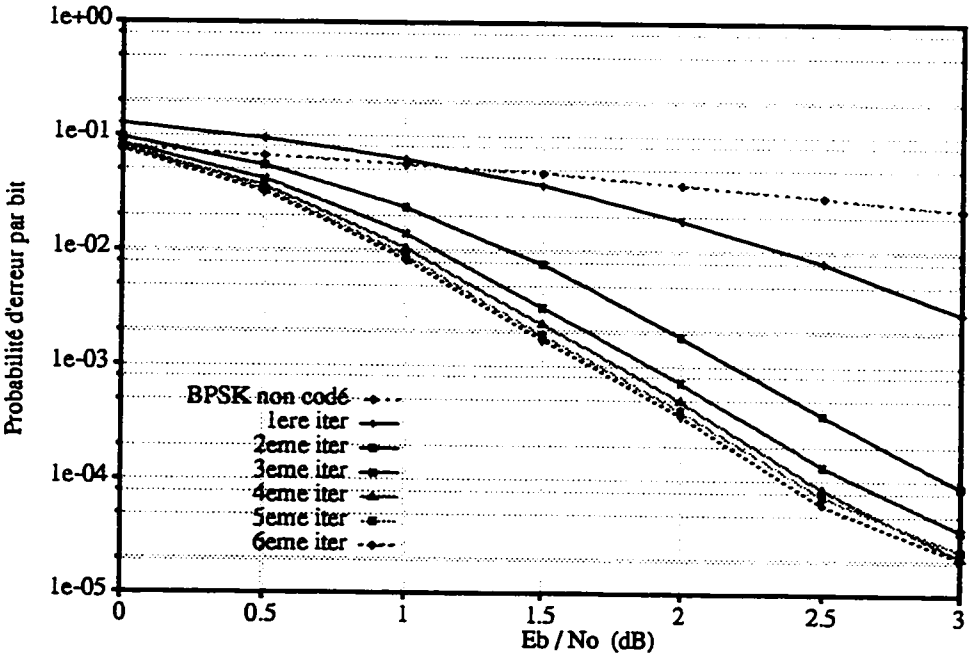


Figure A1.2.5 Performances d'erreur du code turbo de taux R=1/3, avec K=3, G=(1, 5/7), N=196, Entrel. aléatoire

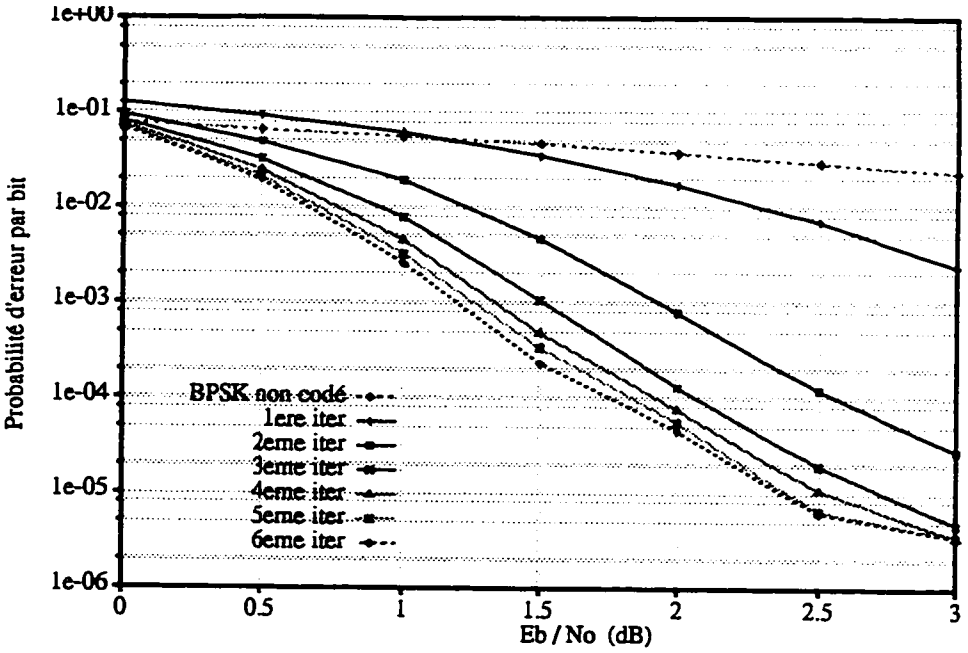


Figure A1.2.6 Performances d'erreur du code turbo de taux R=1/3, avec K=3, G=(1, 5/7), N=400, Entrel. aléatoire

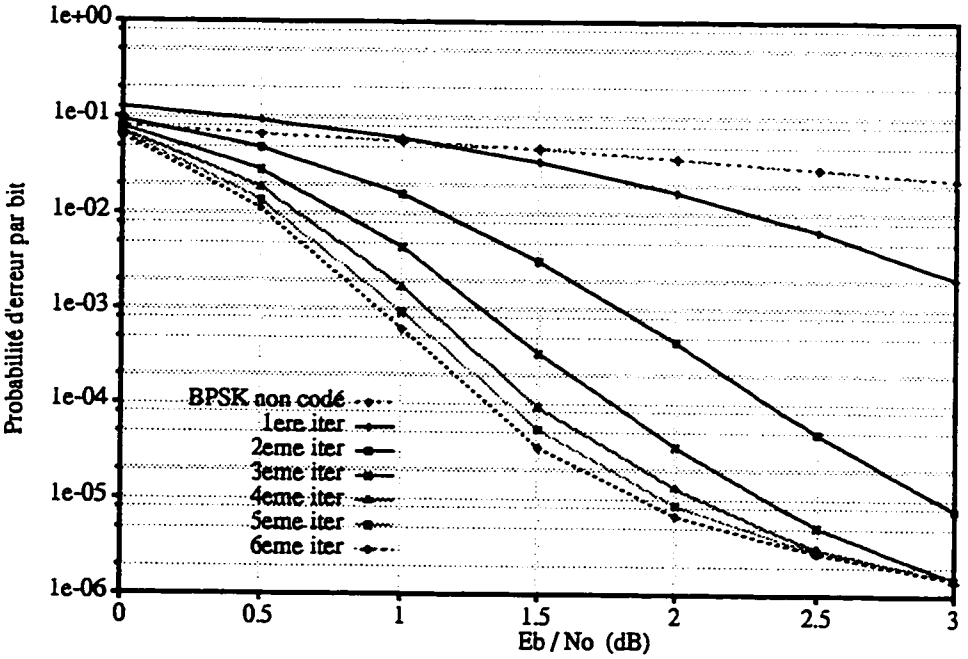


Figure A1.2.7 Performances d'erreur du code turbo de taux R=1/3, avec K=3, G=(1, 5/7), N=900, Entrel. aléatoire

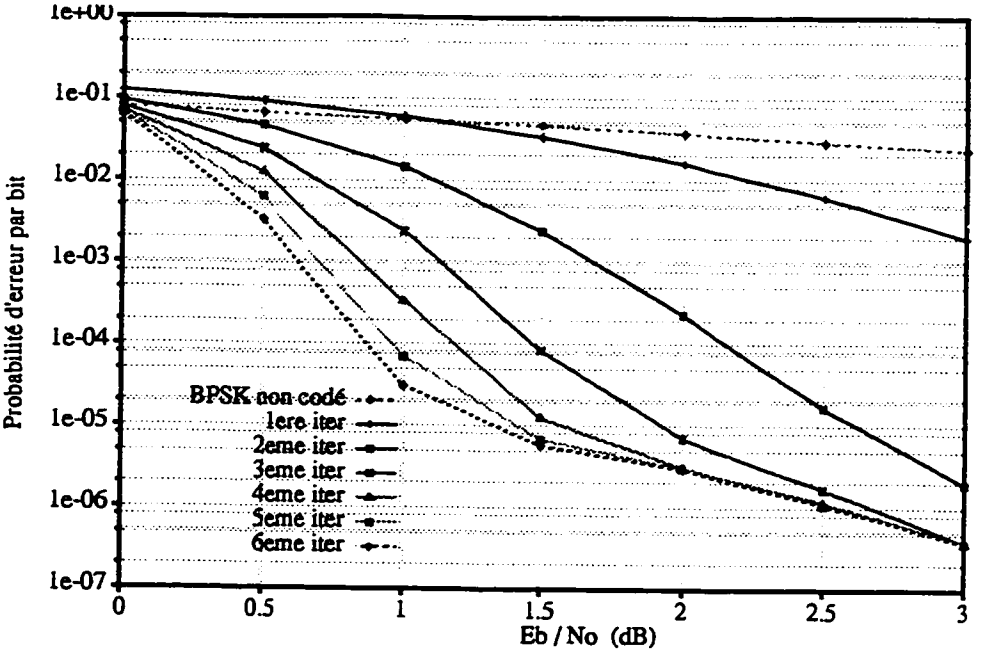


Figure A1.2.8 Performances d'erreur du code turbo de taux R=1/3, avec K=3, G=(1, 5/7), N=3600, Entrel. aléatoire

A1.2.2 Codeurs élémentaires K=5, G=(1, 21/37)

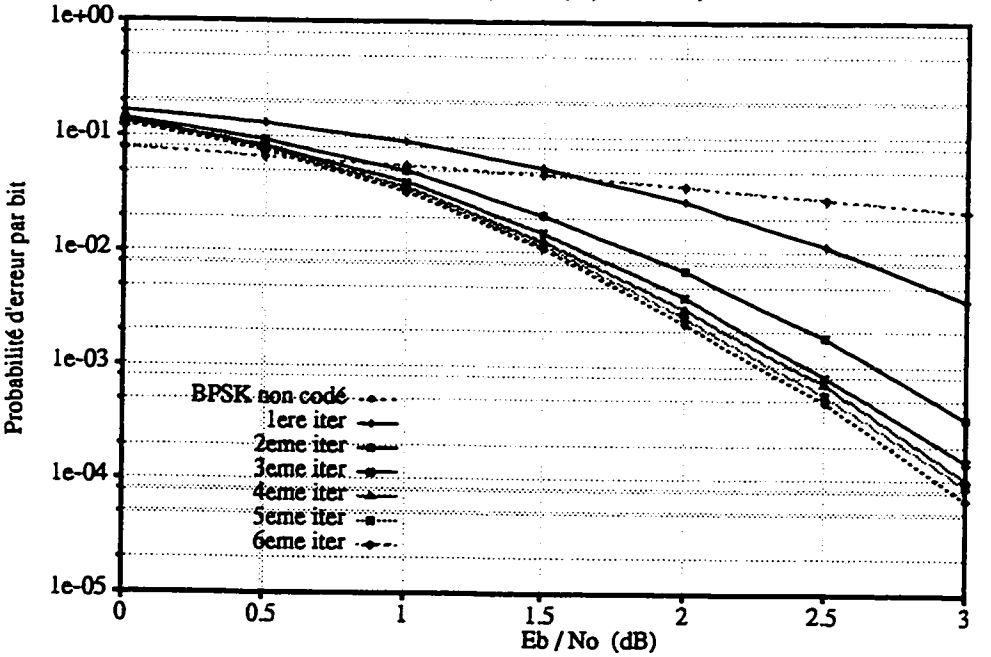


Figure A1.2.9 Performances d'erreur du code turbo de taux R=1/3, avec K=5, G=(1, 21/37), N=100, Entrel. bloc

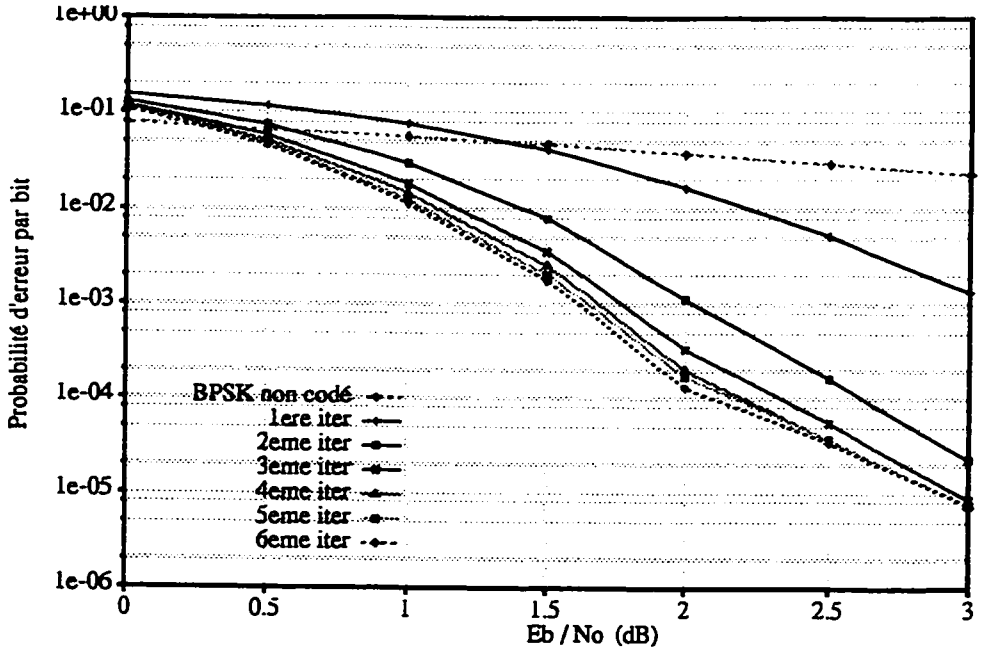


Figure A1.2.10 Performances d'erreur du code turbo de taux R=1/3, avec K=5, G=(1, 21/37), N=196, Entrel. bloc

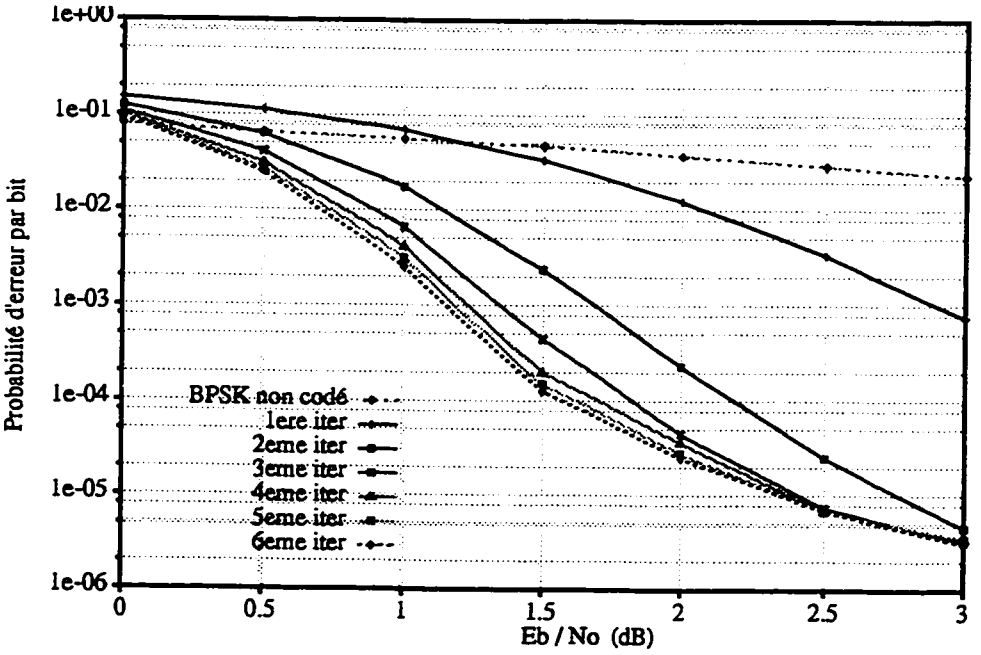


Figure A1.2.11 Performances d'erreur du code turbo de taux R=1/3, avec K=5, G=(1, 21/37), N=400, Entrel. bloc

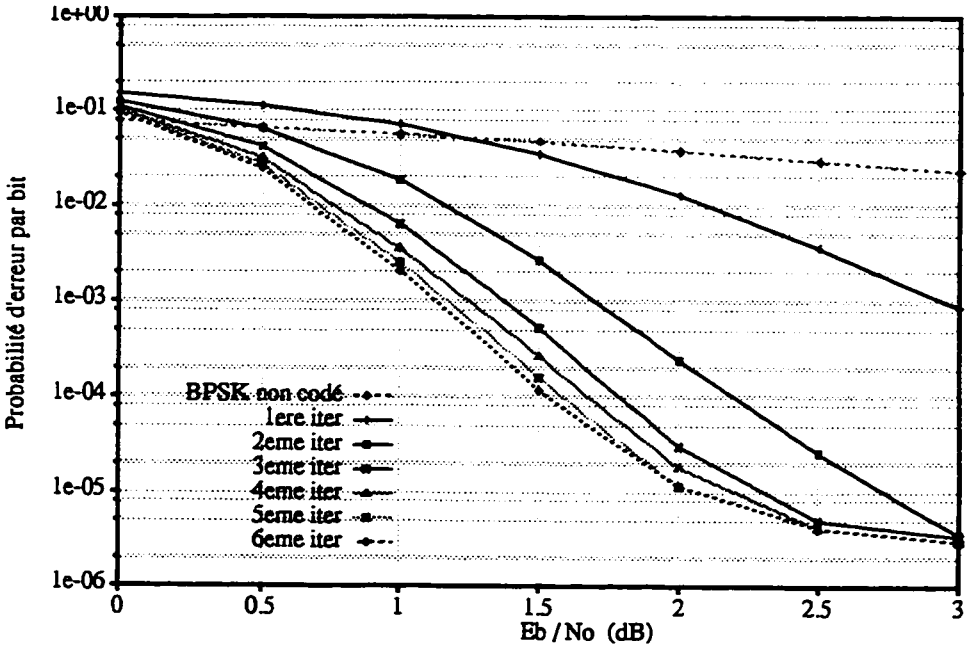


Figure A1.2.12 Performances d'erreur du code turbo de taux R=1/3, avec K=5, G=(1, 21/37), N=400, Entrel. aléatoire

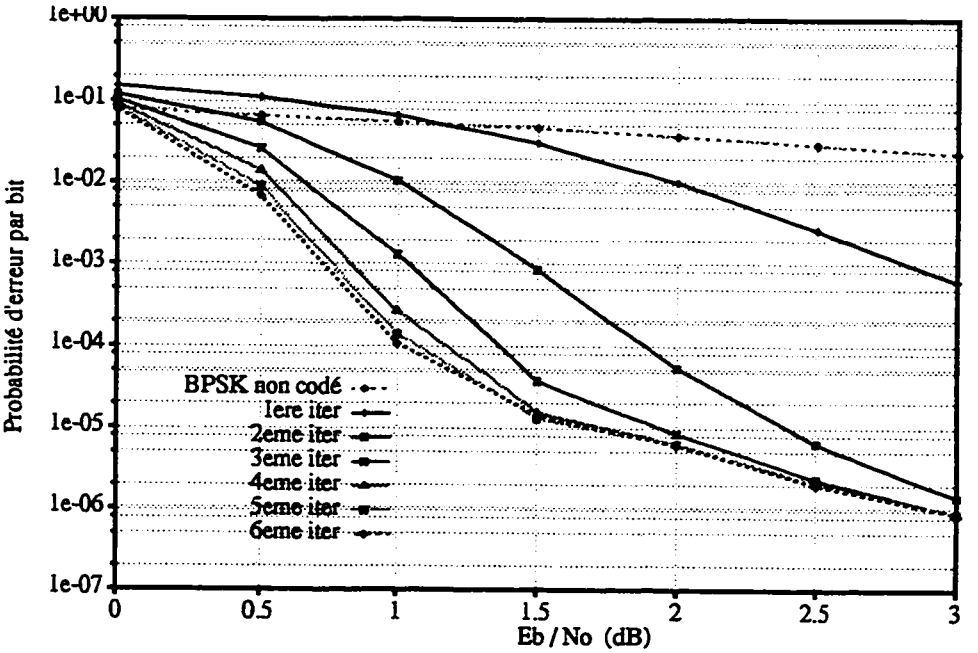


Figure A1.2.13 Performances d'erreur du code turbo de taux R=1/3, avec K=5, G=(1, 21/37), N=900, Entrel. aléatoire

ANNEXE 2

Résultats de simulation pour les taux de codage R=4/6, R=6/8, R=8/10 et R=14/16

Les performances d'erreurs obtenues par simulation des codes turbo perforés sont rapportées dans cet annexe, en fonction du rapport signal sur bruit E_b/N_0. Tous les codes présentés sont obtenus par la perforation du même code turbo d'origine, de taux R=1/3, et résultant de la concaténation de deux codeurs CRS de taux R=1/2 chacun et dont la longueur de contrainte est K=3 ou K=5. Les blocs utilisés sont toujours de longueur N<1000 bits et les taux de codage testés sont R=4/6, R=6/8, R=8/10 et R=14/16.

Nous avons également représenté les patrons de perforation utilisés, choisis parmi plusieurs patrons testés, pour avoir donné les meilleures performances. La recherche du meilleur patron n'est cependant pas exhaustive.

A2.1 Résultats de simulation pour un taux de codage global

R=4/6

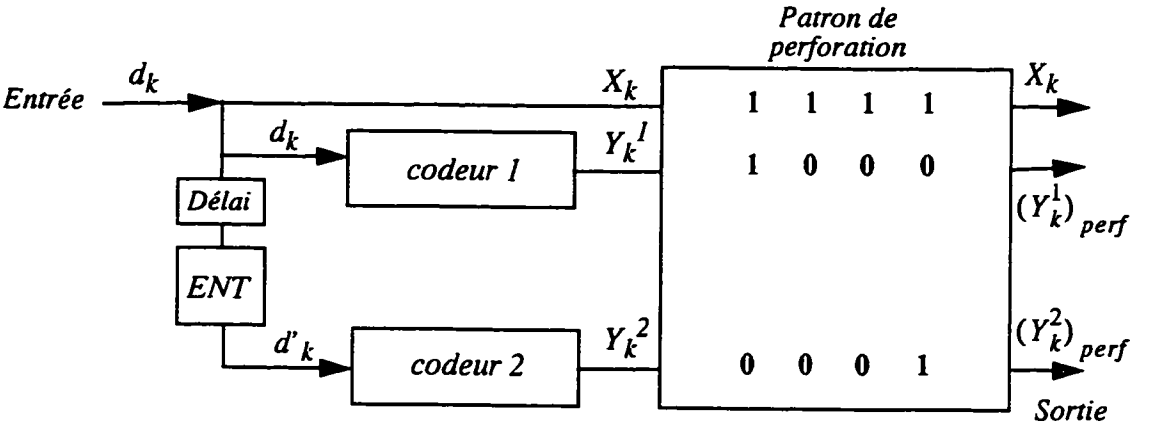


Figure A2.1.1 Structure du codeur turbo

A2.1.1 codeurs élémentaires K=3, G=(1, 5/7)

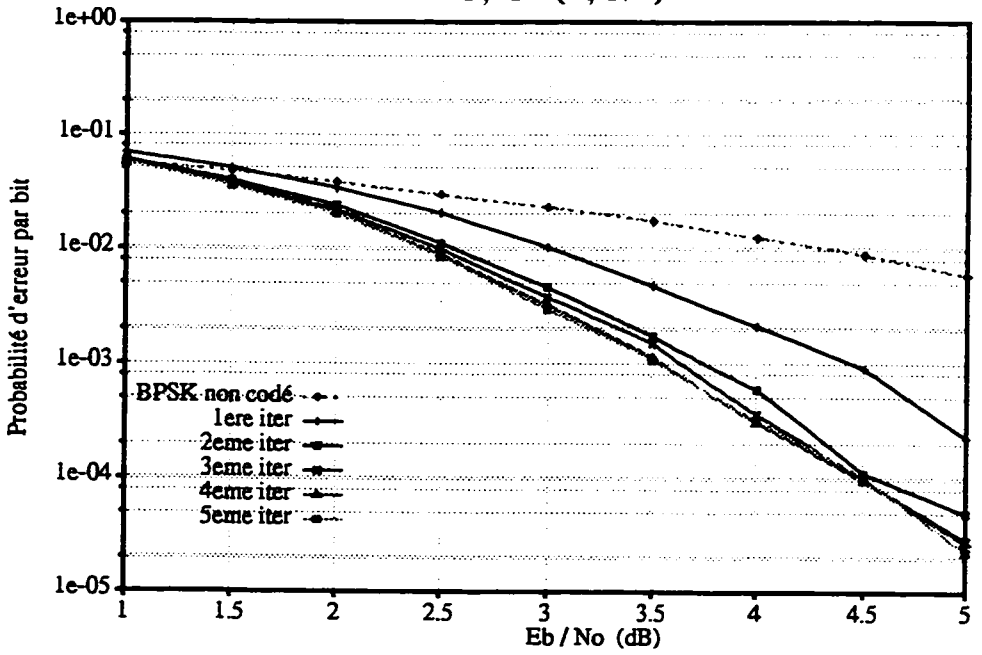


Figure A2.1.2 Performances d'erreur du code turbo de taux R=4/6, avec K=3, G=(1, 5/7), N=81, Entrel. bloc

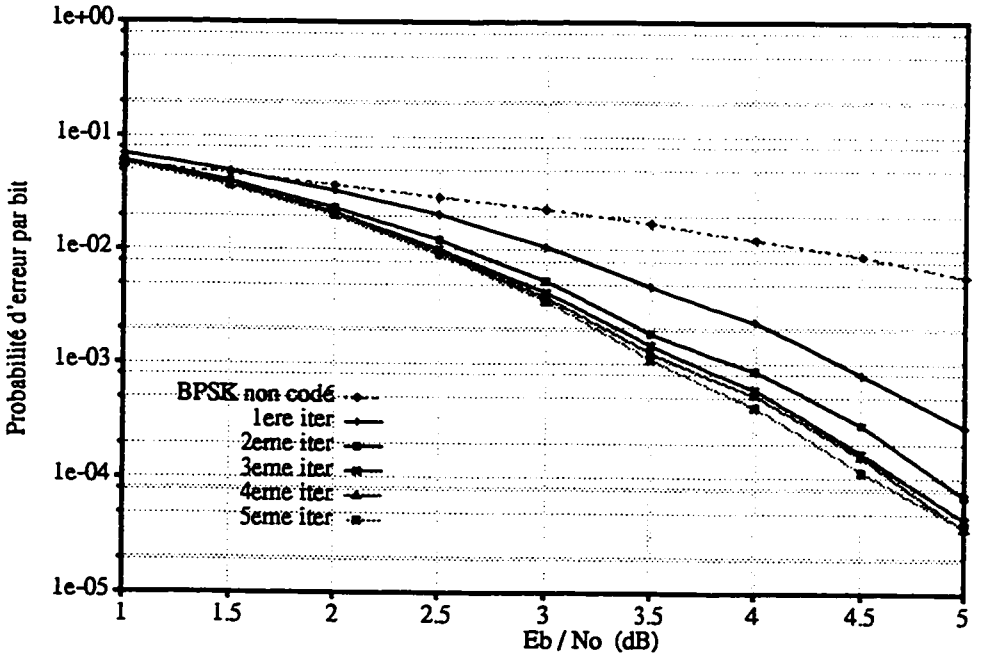


Figure A2.1.3 Performances d'erreur du code turbo de taux R=4/6, avec K=3, G=(1, 5/7), N=81, Entrel. aléatoire

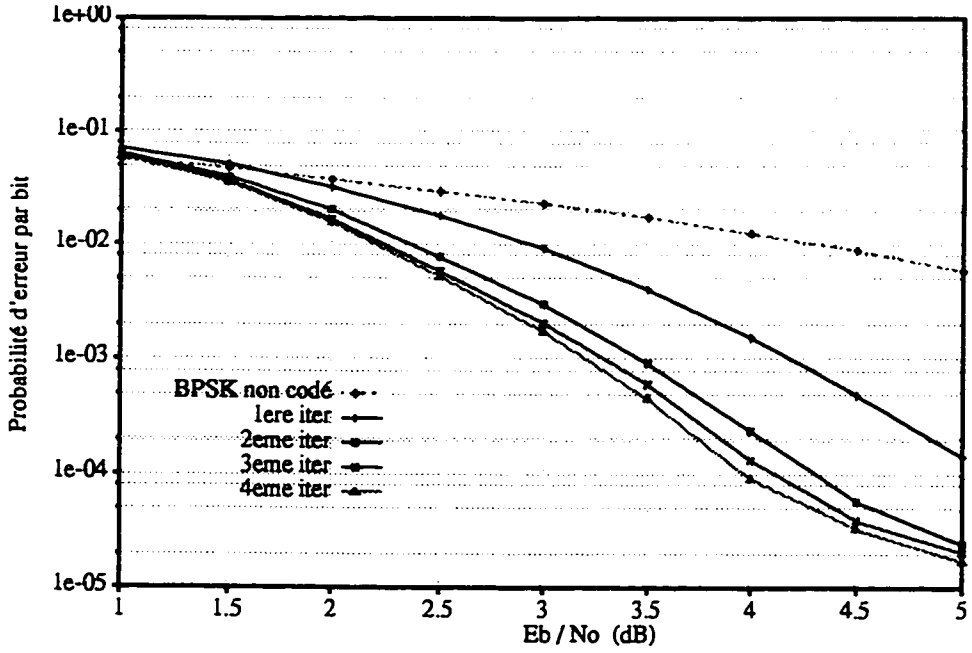


Figure A2.1.4 Performances d'erreur du code turbo de taux R=4/6, avec K=3, G=(1, 5/7), N=169, Entrel. aléatoire

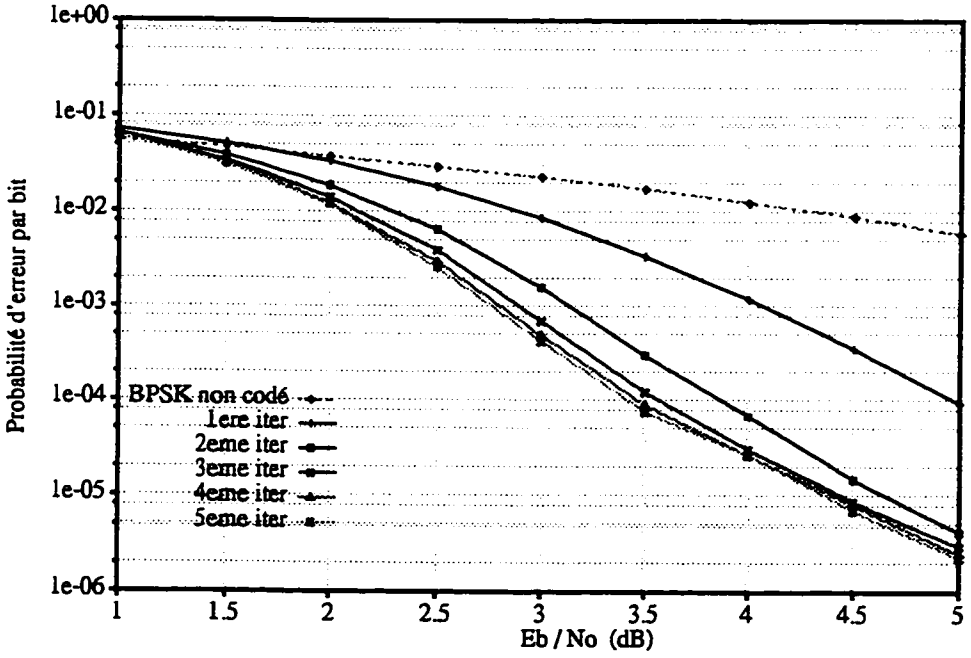


Figure A2.1.5 Performances d'erreur du code turbo de taux R=4/6, avec K=3, G=(1, 5/7), N=400, Entrel. aléatoire

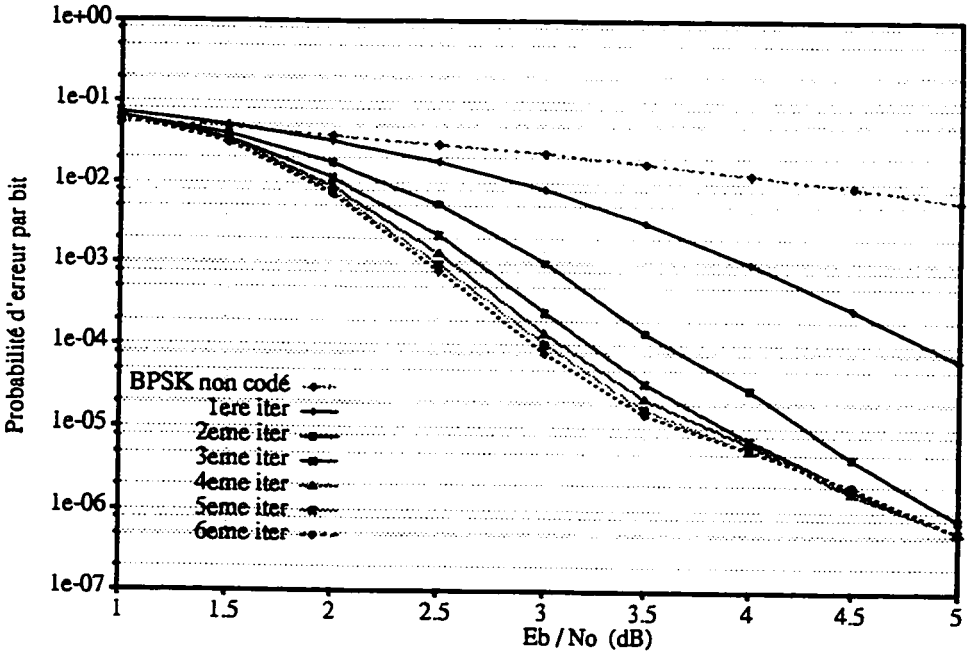


Figure A2.1.6 Performances d'erreur du code turbo de taux R=4/6, avec K=3, G=(1, 5/7), N=900, Entrel. aléatoire

A2.1.2 codeurs élémentaires K=5

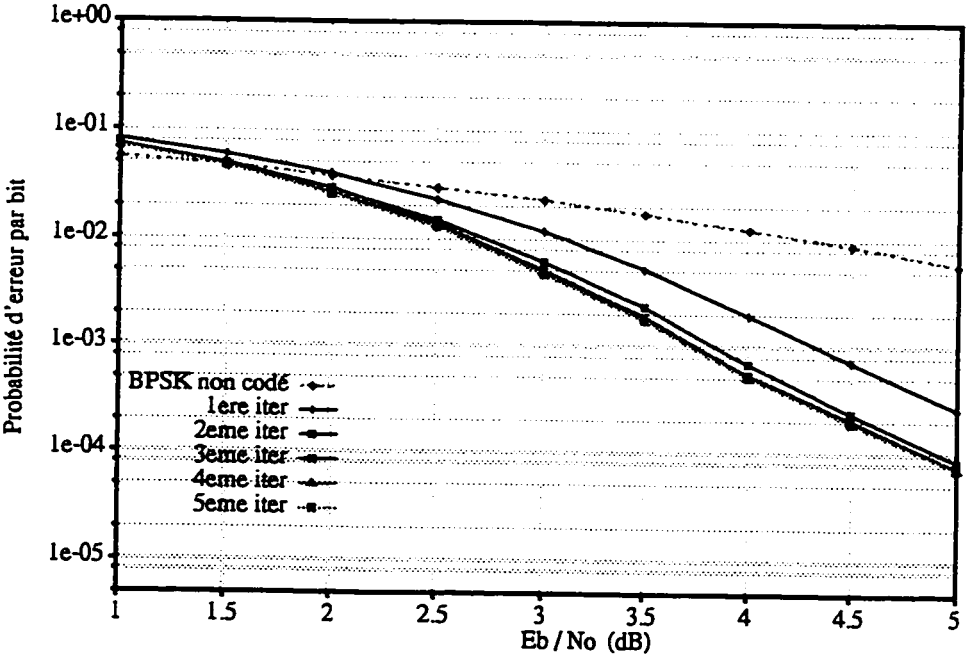


Figure A2.1.7 Performances d'erreur du code turbo de taux R=4/6, avec K=5, G=(1, 21/37) N=81, Entrel. bloc

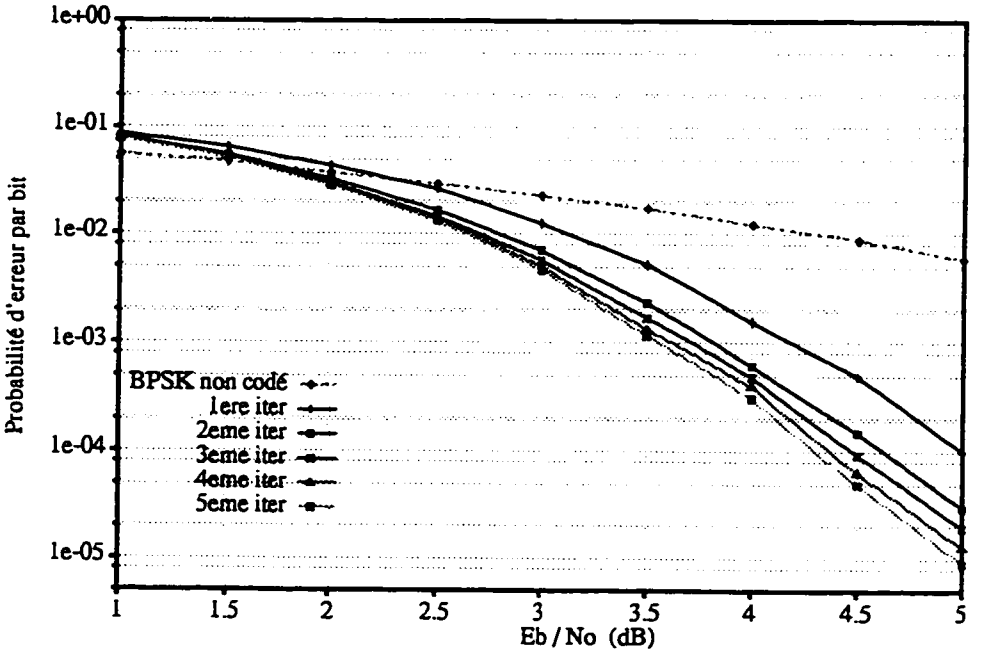


Figure A2.1.8 Performances d'erreur du code turbo de taux R=4/6, avec K=5, G=(1, 27/31) N=81, Entrel. bloc

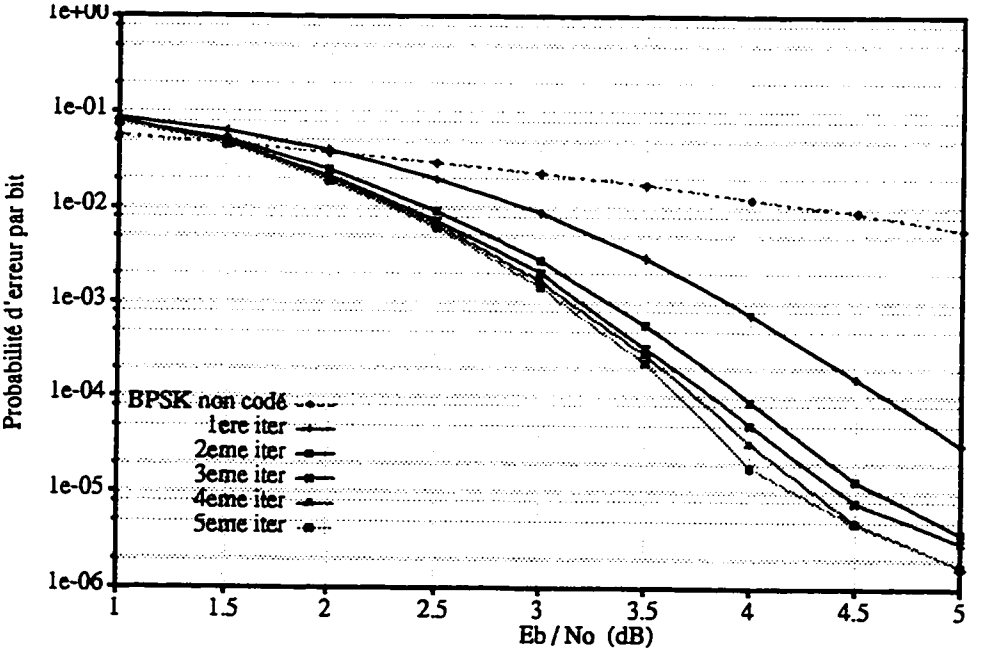


Figure A2.1.9 Performances d'erreur du code turbo de taux R=4/6, avec K=5, G=(1, 27/31), N=169, Entrel. aléatoire

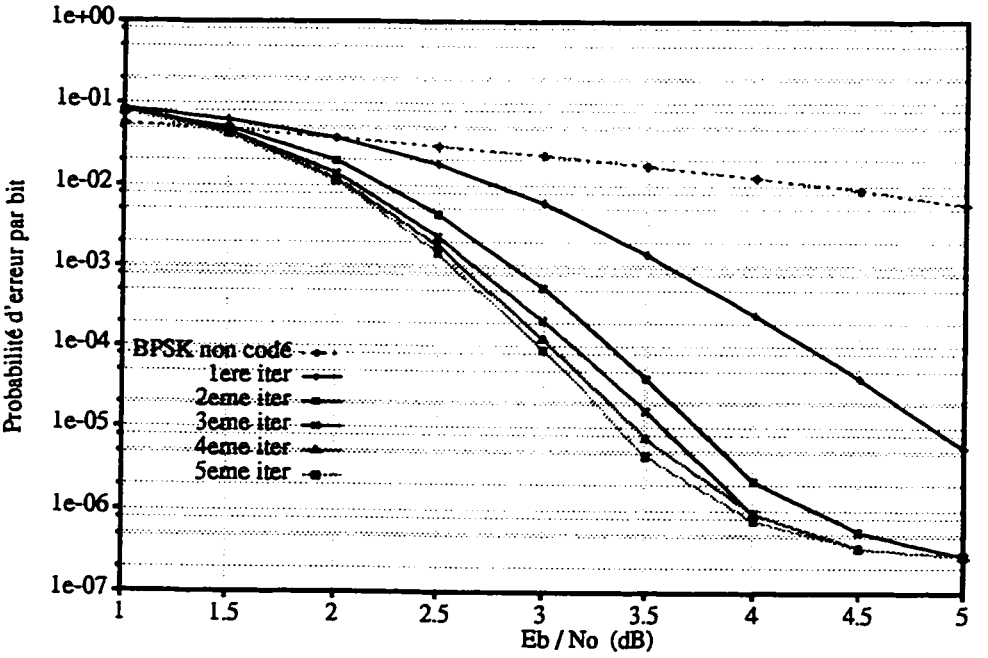


Figure A2.1.10 Performances d'erreur du code turbo de taux R=4/6, avec K=5, G=(1, 27/31), N=400, Entrel. aléatoire

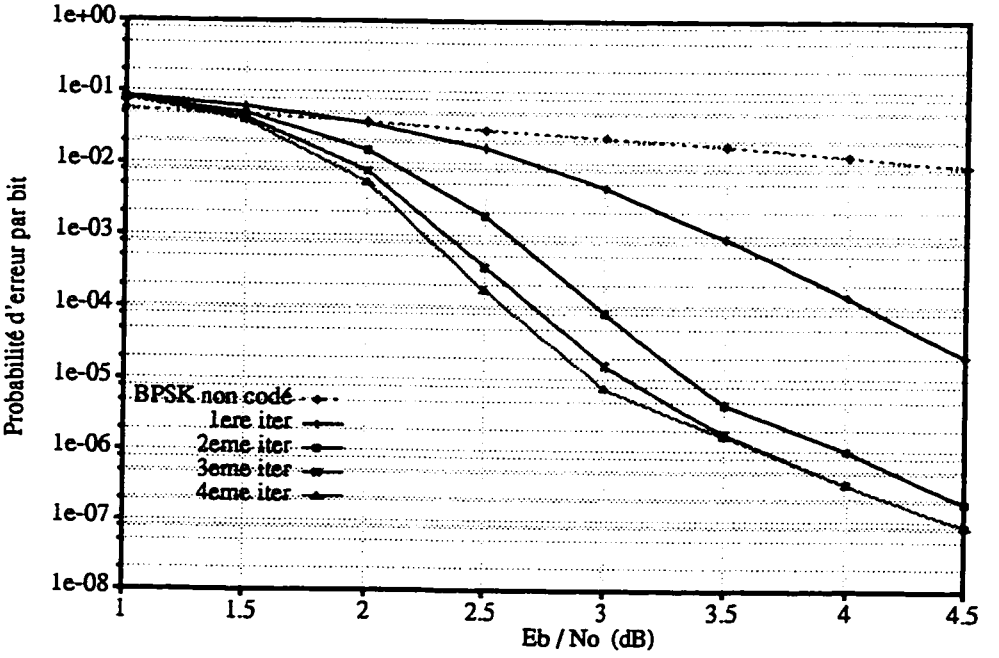


Figure A2.1.11 Performances d'erreur du code turbo de taux R=4/6, avec K=5, G=(1, 27/31), N=900, Entrel. aléatoire

A2.2 Résultats de simulation pour un taux de codage global

R=6/8

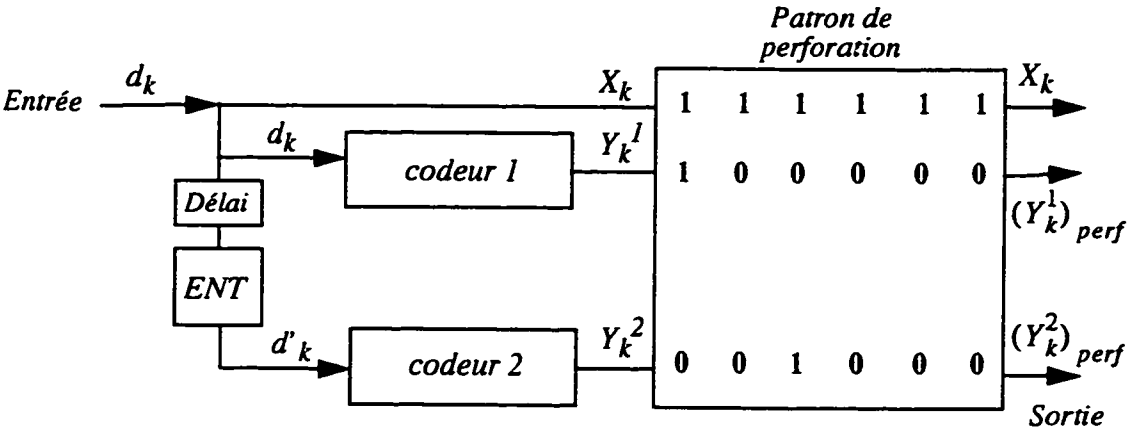


Figure A2.2.1 Structure du codeur turbo

Codeurs élémentaires K=5, G=(1, 27/31)

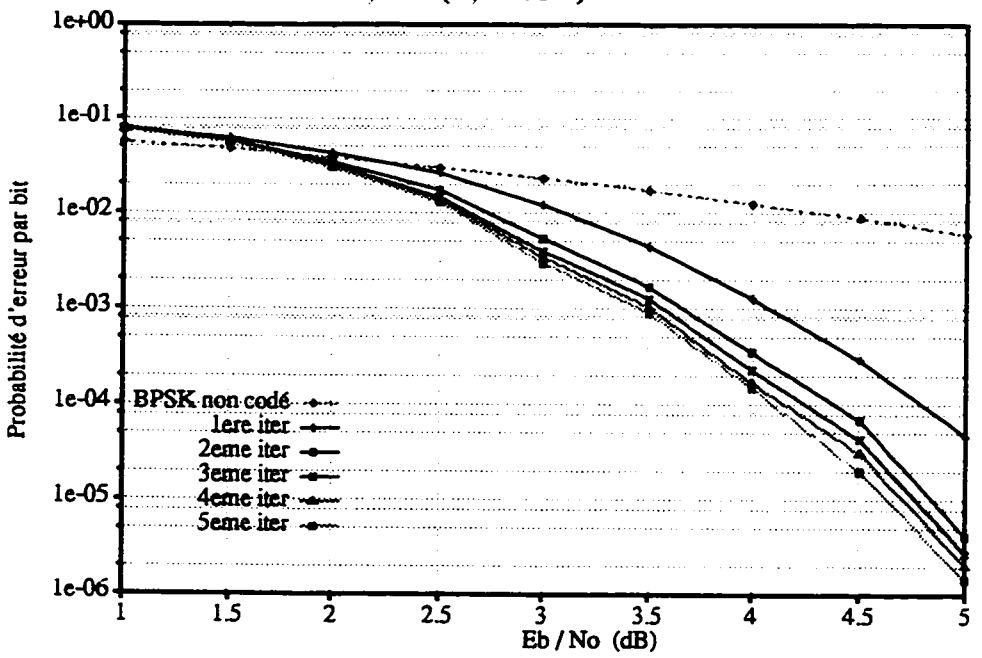


Figure A2.2.2 Performances d'erreur du code turbo de taux R= 6/8, avec K=5, G=(1, 27/31), N=169, Entrel. bloc

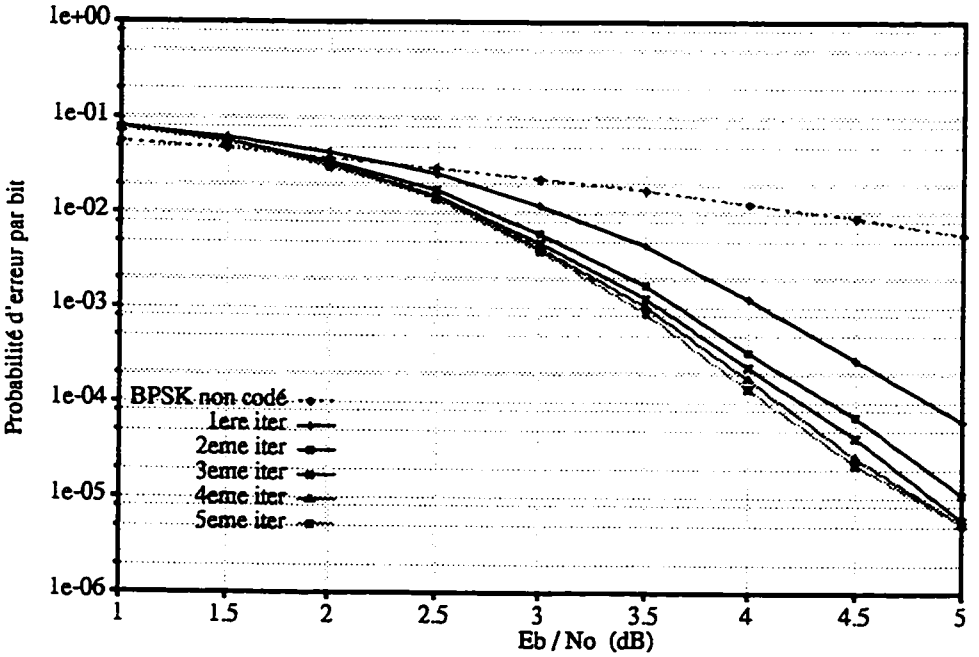


Figure A2.2.3 Performances d'erreur du code turbo de taux R= 6/8, avec K=5, G=(1, 27/31), N=169, Entrel. aléatoire

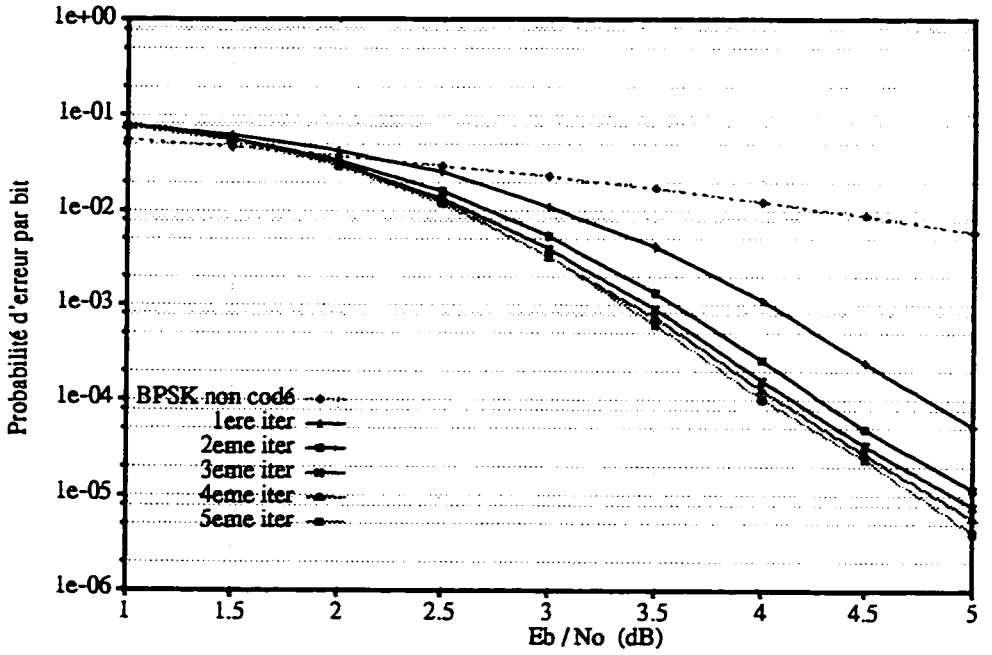


Figure A2.2.4 Performances d'erreur du code turbo de taux R= 6/8, avec K=5, G=(1, 27/31), N=196, Entrel. bloc

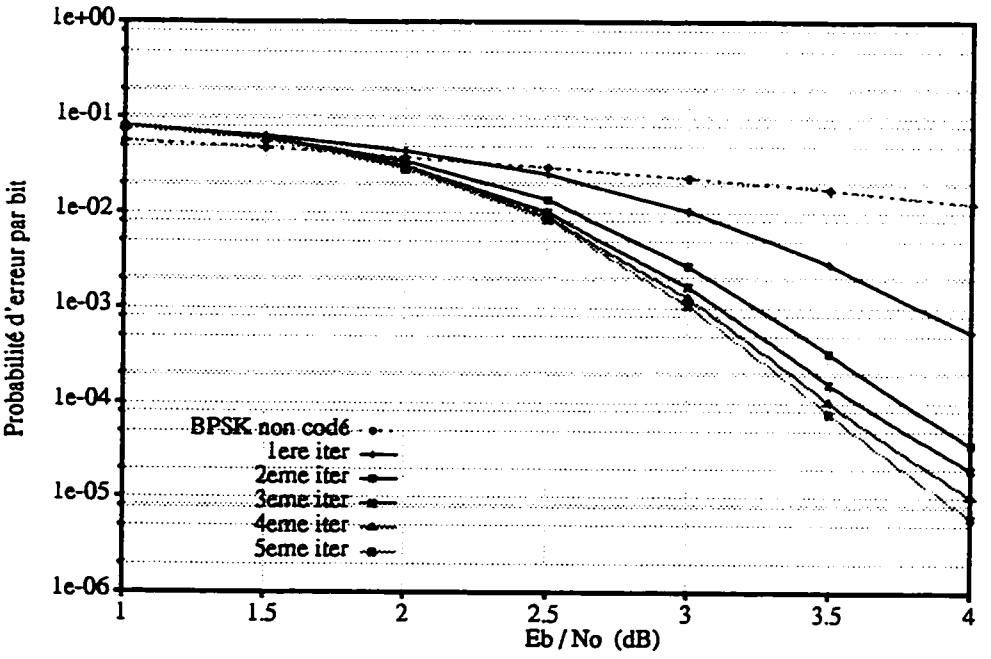


Figure A2.2.5 Performances d'erreur du code turbo de taux R= 6/8, avec K=5, G=(1, 27/31), N=400, Entrel. aléatoire

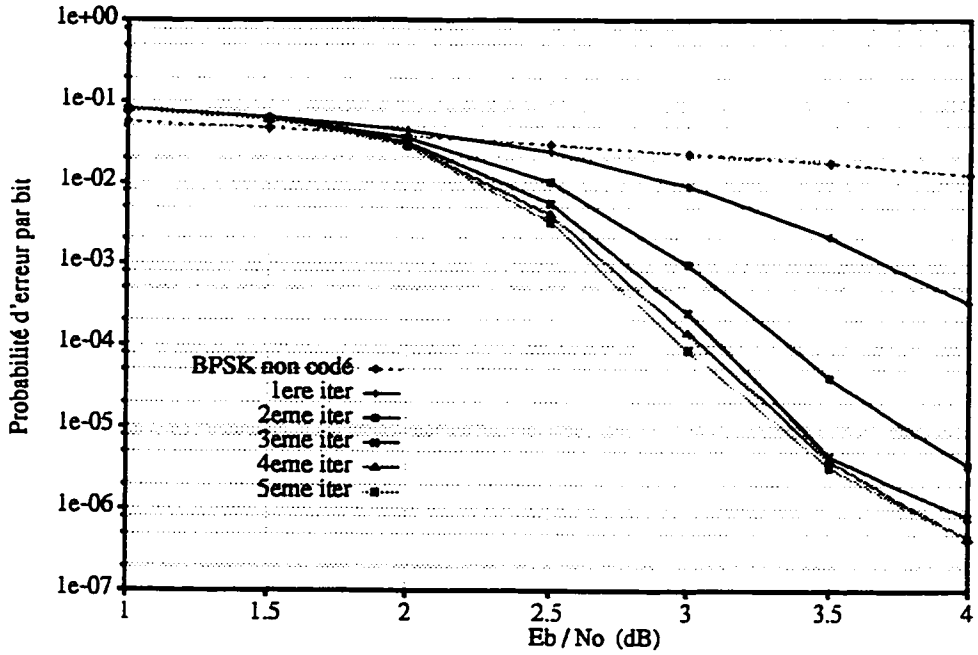


Figure A2.2.6 Performances d'erreur du code turbo de taux R= 6/8, avec K=5, G=(1, 27/31), N=900, Entrel. aléatoire

A2.3 Résultats de simulation pour un taux de codage global

R=8/10

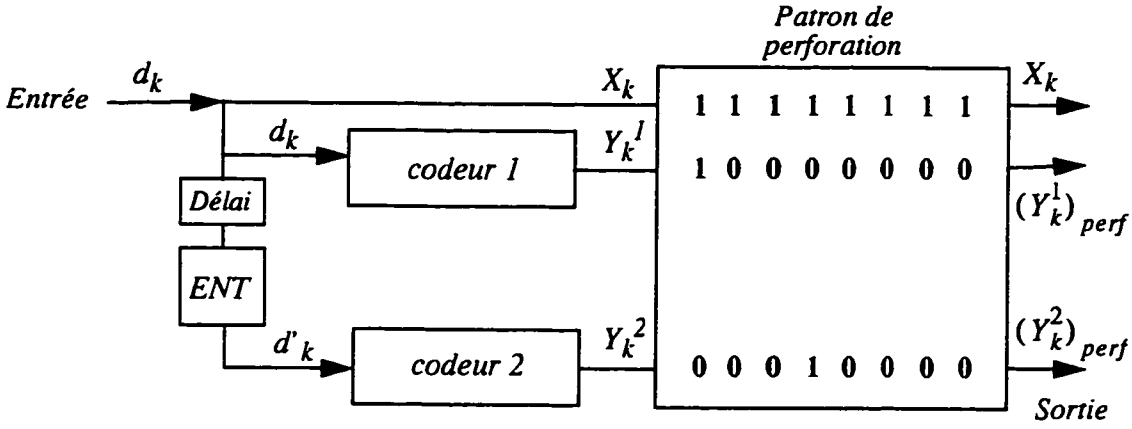


Figure A2.3.1 Structure du codeur turbo

Codeurs élémentaires K=5, G=(1, 27/31)

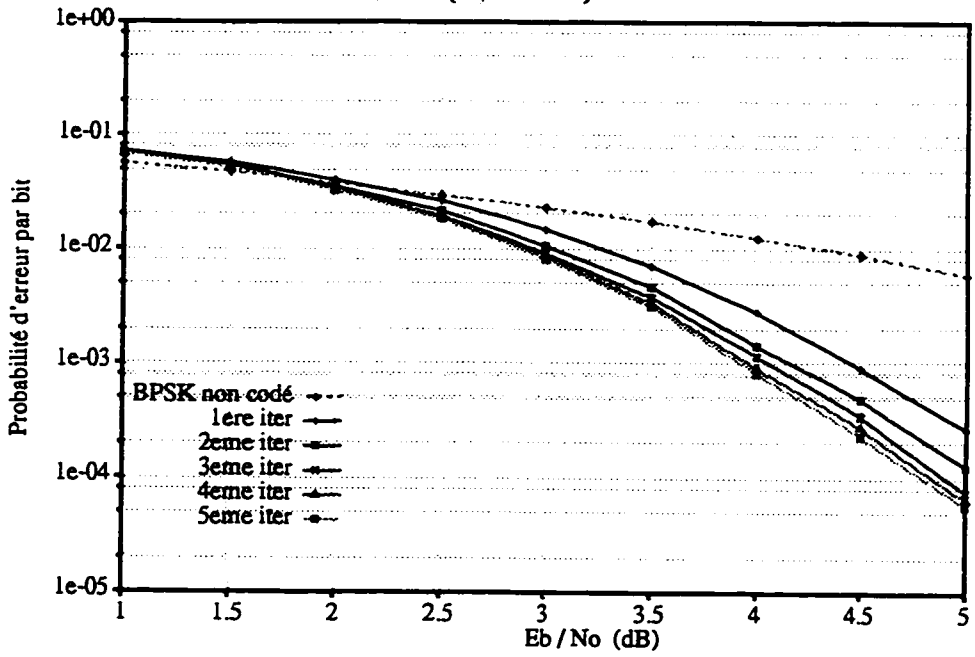


Figure A2.3.2 Performances d'erreur du code turbo de taux R=8/10, avec K=5, G=(1, 27/31), N=81, Entrel. bloc

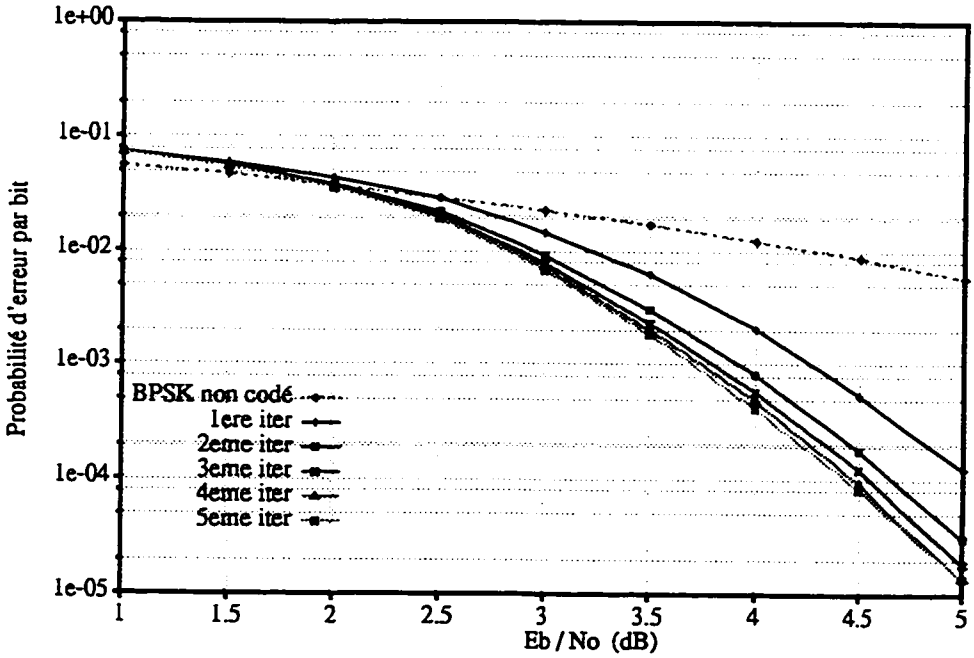


Figure A2.3.3 Performances d'erreur du code turbo de taux R= 6/8, avec K=5, G=(1, 27/31),N=169, Entrel. aléatoire

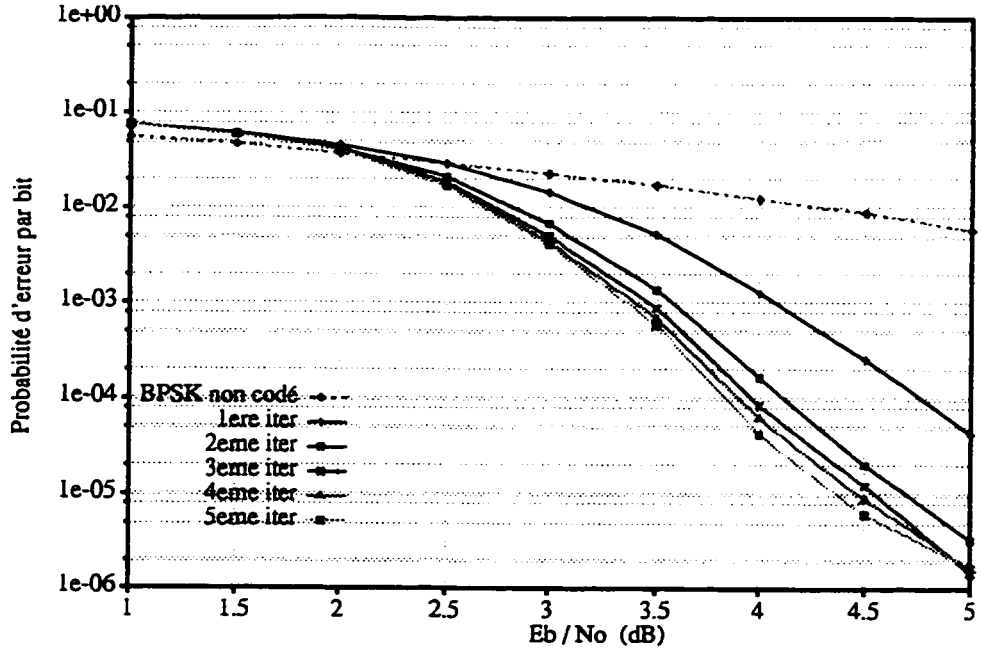


Figure A2.3.4. Performances d'erreur du code turbo de taux R= 6/8, avec K=5, G=(1, 27/31),N=400, Entrel. aléatoire

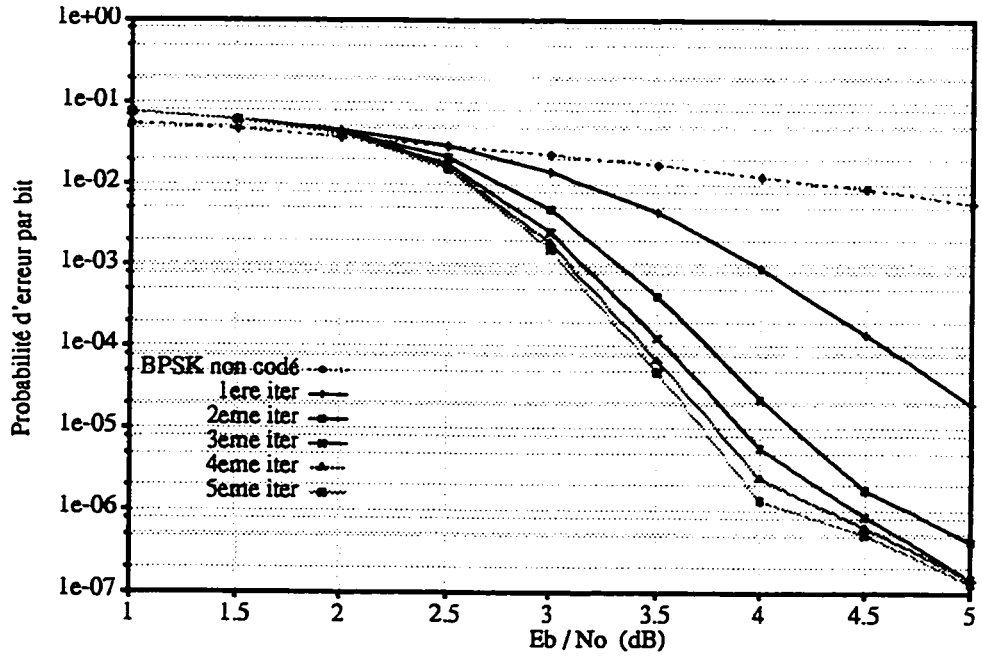


Figure A2.3.5 Performances d'erreur du code turbo de taux R= 6/8, avec K=5, G=(1, 27/31),N=900, Entrel. aléatoire

A2.4 Résultats de simulation pour un taux de codage global

R=14/16

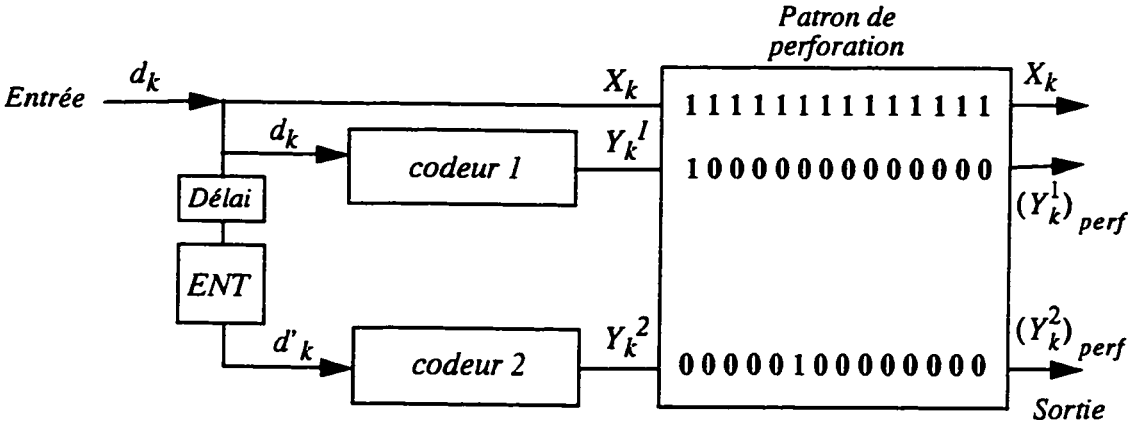


Figure A2.4.i Structure du codeur turbo

Codeurs élémentaires K=5, G=(1, 27/31)

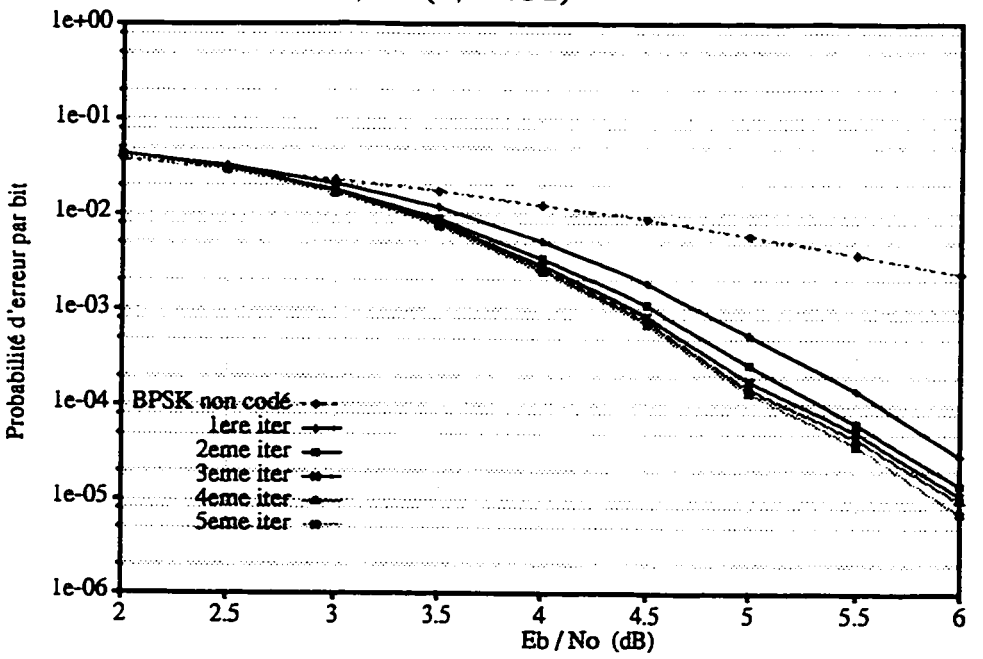


Figure A2.4.2 Performances d'erreur du code turbo de taux R= 14/16, avec K=5, G=(1, 27/31), N=169, Entrel. aléatoire

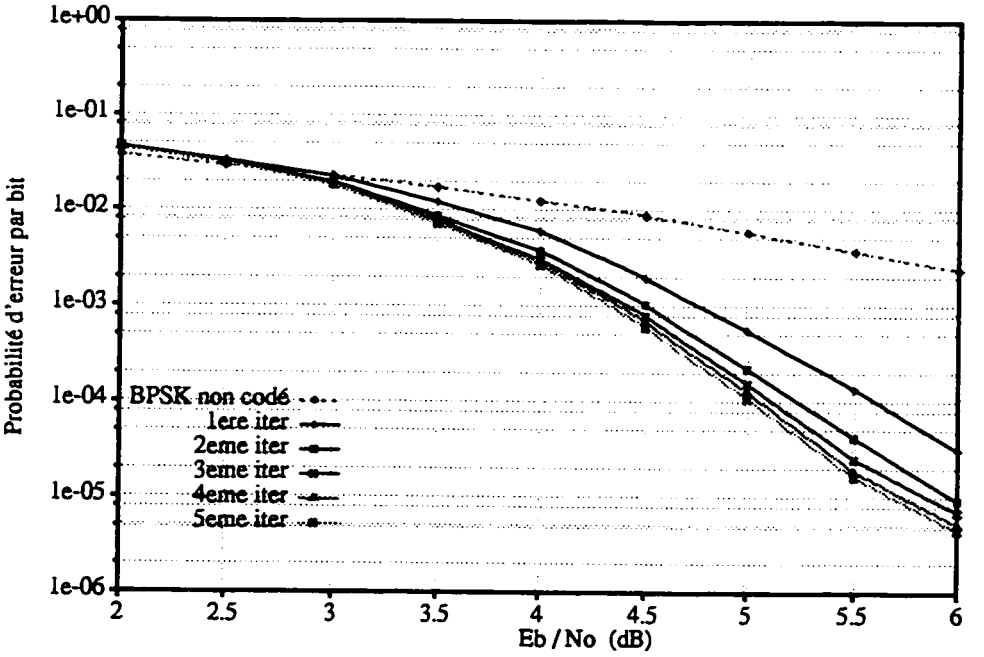


Figure A2.4.3 Performances d'erreur du code turbo de taux R= 6/8, avec K=5, G=(1, 27/31), N=225, Entrel. bloc

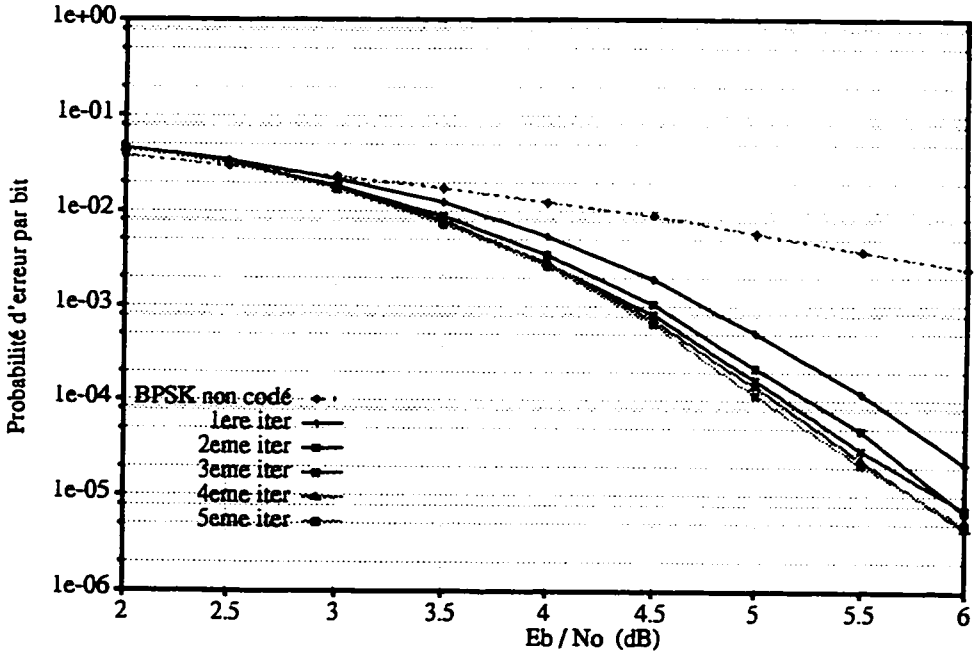


Figure A2.4.4 Performances d'erreur du code turbo de taux R= 6/8, avec K=5, G=(1, 27/31), N=225, Entrel. aléatoire

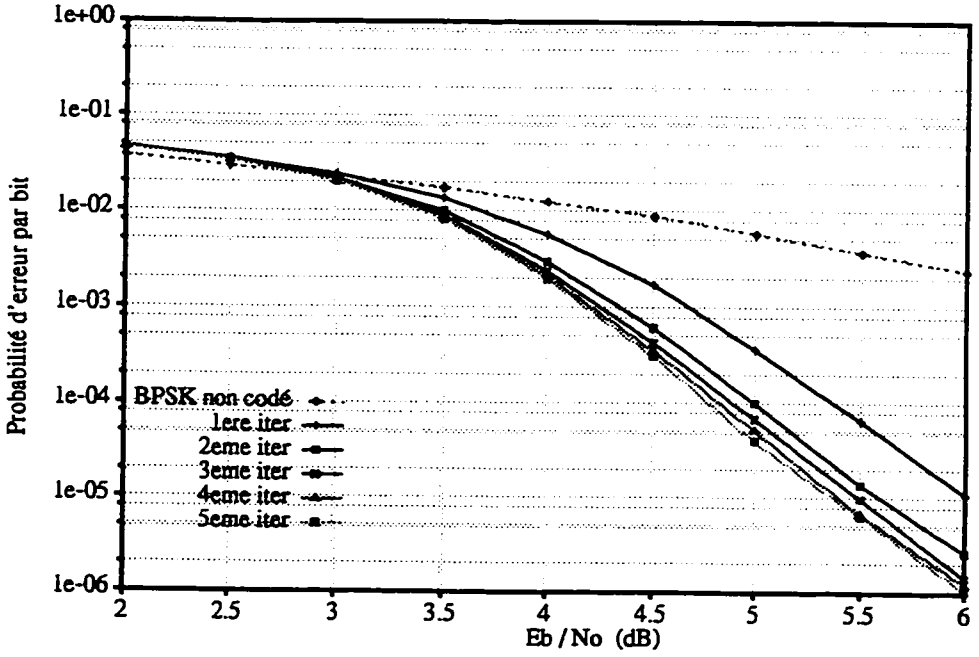


Figure A2.4.5 Performances d'erreur du code turbo de taux R= 6/8, avec K=5, G=(1, 27/31), N=400, Entrel. aléatoire

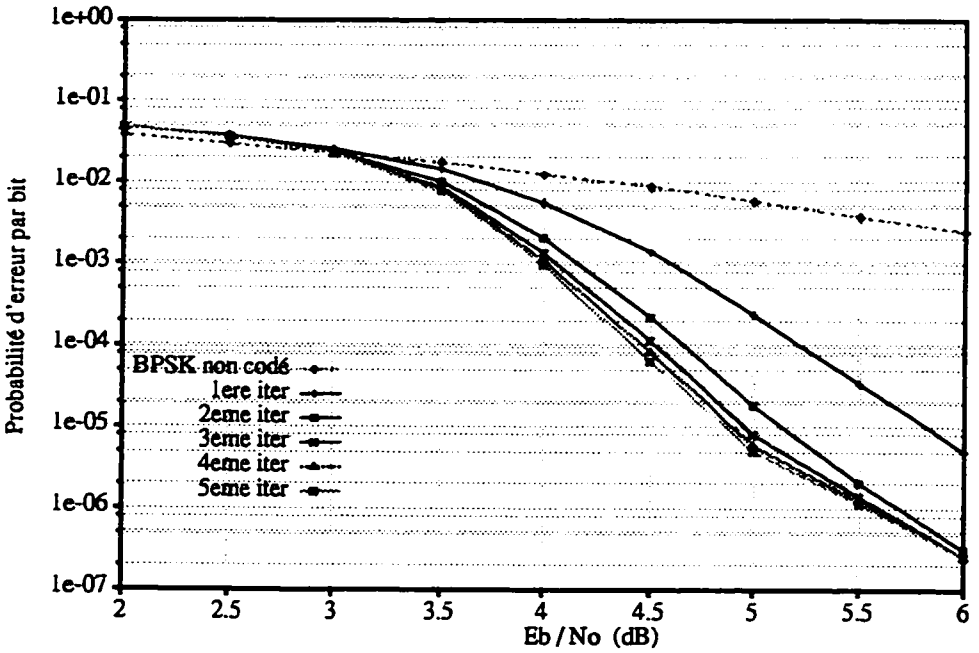
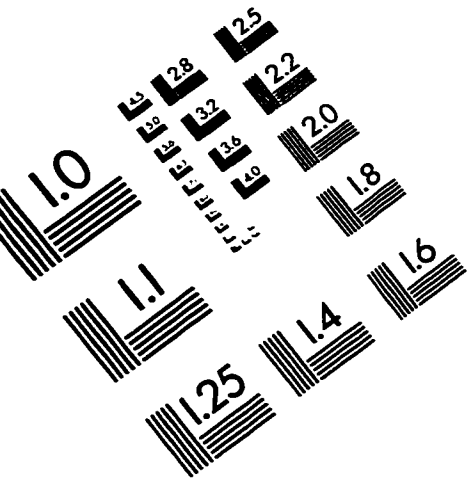
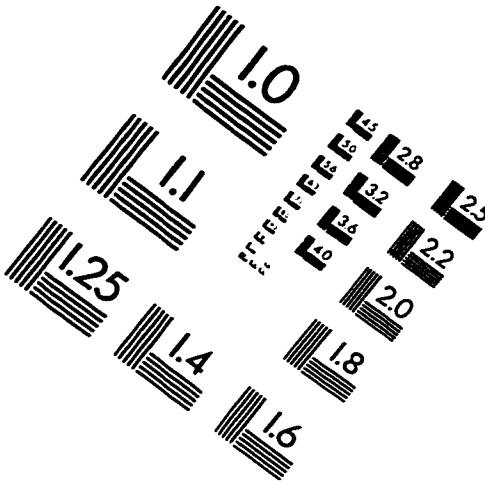
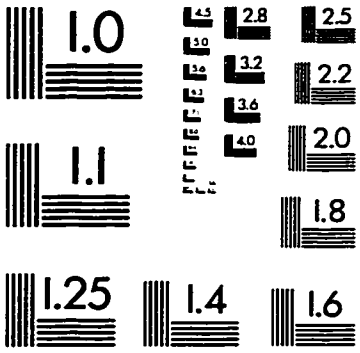


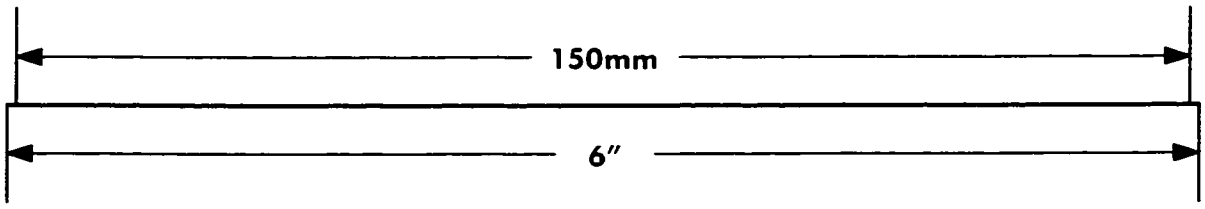
Figure A2.4.6 Performances d'erreur du code turbo de taux R= 6/8, avec K=5, G=(1, 27/31), N=900, Entrel. aléatoire

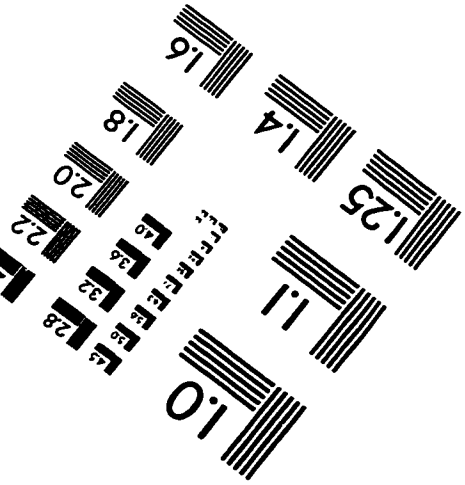
IMAGE EVALUATION TEST TARGET (QA-3)

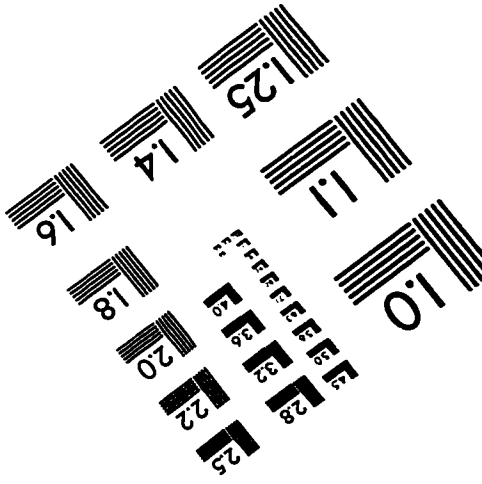












APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989