Titre: Title:	Algorithme de décodage adaptatif pour codes convolutionnels
Auteur: Author:	François Chan
Date:	1989
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Chan, F. (1989). Algorithme de décodage adaptatif pour codes convolutionnels [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/57941/

UNIVERSITE DE MONTREAL

ALGORITHME DE DECODAGE ADAPTATIF POUR CODES CONVOLUTIONNELS

par

François CHAN

DEPARTEMENT DE GENIE ELECTRIQUE

ECOLE POLYTECHNIQUE

MEMOIRE PRESENTE EN VUE DE L'OBTENTION

DU GRADE DE MAÎTRE ES SCIENCES APPLIQUEES (M. Sc. A.)

Décembre 1989

ISBN 0-315-58110-7

UNIVERSITE DE MONTREAL

ECOLE POLYTECHNIQUE DE MONTREAL

Ce mémoire intitulé:

ALGORITHME DE DECODAGE ADAPTATIF POUR CODES CONVOLUTIONNELS

présenté par: François CHAN

en vue de l'obtention du grade de: M.Sc.A.

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

Mme. Catherine ROSENBERG, Doc. Sc., Présidente

M. David HACCOUN, Ph.D., Directeur de recherche

M. Jean CONAN, Ph.D., Membre du jury

Sommaire

Le but de cette recherche est de déterminer un algorithme de décodage, qui donne une probabilité d'erreur faible mais qui ne présente pas les problèmes des deux principales techniques de décodage probabiliste: l'effort de calcul de l'algorithme de Viterbi est très élevé et croît exponentiellement avec la longueur de contrainte tandis que pour le décodage séquentiel, l'effort de calcul est faible en moyenne mais très variable et provoque des débordements et effacements.

L'algorithme proposé est une modification de l'algorithme de Viterbi (AV); au lieu de garder les 2^{K-1} états à chaque niveau du treillis, seuls certains des meilleurs états sont gardés. Les noeuds, dont la probabilité d'appartenir au chemin correct est faible, sont éliminés. Ce décodeur s'adapte à la qualité du canal: le travail fourni est plus élevé s'il y a du bruit dans le canal.

Comparativement à l'algorithme de Viterbi, le nombre moyen de survivants est réduit, surtout si le canal est bon. L'analyse de la probabilité d'erreur montre que, pour un même code, la dégradation par rapport à l'algorithme de Viterbi est négligeable et des simulations sur ordinateur le confirment. L'effort de calcul réduit permet l'utilisation de codes dont la longueur de contrainte K peut atteindre 10, alors qu'avec l'AV, K=7 représente la longueur maximale, en pratique. Un code ayant une longueur de contrainte plus élevée va donner une meilleure performance

d'erreur. Nous montrons aussi que la distribution de l'effort de calcul est exponentielle et par conséquent, contrairement au décodage séquentiel, la probabilité de débordement du tampon d'entrée est faible.

Abstract

The objective of this research is to determine a new decoding algorithm for convolutional codes, which provides a small probability of error but does not present the problems associated with the two main probabilistic decoding algorithms: the computational effort of the Viterbi algorithm is very high and increases exponentially with the constraint length, whereas for sequential decoding, the computational effort is small on average but very variable, and causes overflows and erasures.

The proposed algorithm is a modification of the Viterbi algorithm (VA) ; instead of keeping the 2^{K-1} states at each trellis level, only some of the best states are kept. The nodes, which are unlikely to belong to the correct path, are eliminated. This decoder adapts itself to the quality of the channel: the workload increases when there is noise in the channel.

Compared to the VA, the average number of survivors is reduced, especially if the channel is good. The error probability analysis shows that the degradation from the VA is negligible and this is confirmed by computer simulations. Since the computational load is reduced, codes with a constraint length K up to 10 can be used, while for the VA, K=7 is the maximal length, in practice. A code with a larger constraint length gives a better error performance. We also show that the computational effort distribution is exponential and consequently, contrary to sequential decoding, the probability of input buffer overflow is small.

Remerciements

Je voudrais d'abord exprimer toute ma gratitude à mon directeur de recherche, M. David Haccoun. Je lui suis très reconnaissant pour son excellente supervision, ses judicieux conseils, ainsi que pour son aide financière.

Mes plus sincères remerciements vont aussi à Mme. Catherine Rosenberg et à M. Jean Conan pour leurs remarques et commentaires, qui ont permis d'améliorer ce mémoire.

Je dois également remercier mes amis de la section Communication qui m'ont aidé au cours de mes études de maîtrise. Finalement, merci à ma famille pour le soutien constant.

Table des Matières

Liste des Figures

Liste des Tableaux

Chapitre 1

Introduction

1.1 Systèmes de communication codée

L'augmentation du traffic de données dans les télécommunications a entrainé un besoin de systèmes de transmission numérique fiables et efficaces. Le contrôle des erreurs est donc devenu un des problèmes majeurs en communication.

En 1948, Shannon [5] a montré qu'en codant et en décodant les données de façon appropriée, une probabilité d'erreur arbitrairement faible peut être obtenue si l'information est transmise à un taux inférieur à un taux maximal, appelé la capacité C du canal. Ce résultat remarquable indique que la limite de performance imposée par le bruit du canal n'est pas la probabilité d'erreur mais le taux de transmission. Pour améliorer la fiabilité d'un canal, il n'est donc pas indispensable d'augmenter la puissance de transmission (rapport signal-à-bruit), il suffit d'utiliser du codage.

Le but du codage est d'introduire de la redondance dans l'information. Ainsi, on émet plus de bits qu'il n'en faut et cette redondance est utilisée pour détecter une erreur et demander une retransmission (technique ARQ: Automatic Repeat

Request) ou encore pour corriger une erreur (FEC: Forward Error Correction). Une troisième possibilité consiste à combiner les deux techniques précédentes pour obtenir un système hybride FEC/ARQ [1].

Le codage correcteur d'erreur permet de fournir un gain de codage, défini comme la différence en dB entre les valeurs de E_b/N_0 requises pour une probabilité d'erreur donnée entre ce système de codage et la modulation PSK cohérente sans codage. Ici, E_b est l'énergie reçue par bit d'information, N_0 la densité spectrale du bruit et le rapport signal-à-bruit E_b/N_0 sert à comparer différents systèmes de codage. Le gain de codage est obtenu au détriment d'une augmentation de la largeur de bande.

Il y a deux types de codes: les codes en bloc et les codes convolutionnels [2]. Pour les premiers, les données sont d'abord groupées en blocs avant d'être codées. En codage convolutionnel, les données sont codées de façon continue. Pour une même complexité, les codes convolutionnels sont reconnus être supérieurs aux codes en bloc [3,4] et ils sont les seuls à être considérés dans ce mémoire.

Les algorithmes de décodage probabiliste les plus utilisés sont l'algorithme de Viterbi [4], le décodage séquentiel [1,5], ainsi que des variantes plus récentes de ces algorithmes [6–10]. Le but de cette recherche est de diminuer la probabilité d'erreur de ces algorithmes, pour une complexité donnée ou encore, de diminuer la complexité, pour une performance d'erreur donnée. Nous allons décrire les techniques de décodage classiques et indiquer leurs limitations. Un nouvel algorithme de décodage est ensuite proposé pour pallier aux inconvénients des techniques précédentes tout en gardant leurs avantages. Cet algorithme doit donner une probabilité d'erreur faible, proche de celle d'un algorithme optimal, tel l'algorithme de Viterbi. De plus, l'effort de calcul doit être peu élevé et moins variable que celui du décodage séquentiel. Cet algorithme est obtenu en modifiant l'algorithme de Viterbi et en le rendant adaptatif. Ses propriétés et performances sont analysées et vérifiées par

simulation sur ordinateur.

1.2 Structure des codes convolutionnels

Dans cette section, le codage convolutionnel est introduit. Seules les notions qui vont être utilisées par la suite, sont traitées. Cette description suit de près celle faite par Haccoun [1].

1.2.1 Codeur convolutionnel

Un codeur convolutionnel de taux de codage R=1/V peut être représenté par une machine séquentielle à états finis. Le taux de codage R est le rapport entre le nombre de bits entrant en même temps dans le codeur et le nombre de symboles codés à la sortie. Le codeur comprend un registre à décalage de K étages ou cellules, V additionneurs modulo 2 et un commutateur à la sortie. K, le nombre d'étages du registre, est appelé la longueur de contrainte du code. Le code est spécifié par l'ensemble des connexions entre les étages et les V additionneurs. On peut donc décrire un code en donnant les V vecteurs de connexion, chacun de dimension K:

$$G_j = (g_{j1}, g_{j2}, ..., g_{jK}) \qquad j = 1, 2, ..., V$$

où g_{ji}=1 si l'étage i du registre et l'additionneur j sont reliés et g_{ji}=0, sinon. Les vecteurs G_j sont appelés les générateurs du code.

Un codeur convolutionnel a une certaine mémoire et les symboles codés ne dépendent pas seulement du dernier bit entré mais aussi des n bits précédents, n dépendant de la mémoire. Un exemple de codeur convolutionnel de longueur de contrainte K=3 et de taux R=1/2 est montré à la figure 1.1.

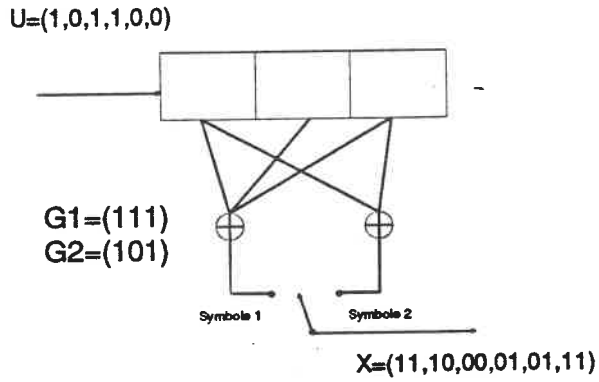

Figure 1.1: Codeur convolutionnel (K=3, R=1/2)

Nous allons maintenant décrire les étapes du codage. Au début, toutes les cellules du registre sont initialisées à zéro. Les bits d'information entrent dans le registre, un bit à la fois. Après chaque décalage, le commutateur échantillonne les sorties des V additionneurs. Les V symboles codés obtenus sont alors modulés et transmis. Une queue de (K- 1) symboles connus, généralement des zéros, est ajoutée à la fin de la séquence d'information pour initialiser de nouveau le registre à zéro. Comme exemple, la séquence

$$\underline{U} = (1, 0, 1, 1, 0, 0)$$

est codée en utilisant le codeur de la figure 1.1. Cette séquence a seulement quatre bits d'information, les deux derniers bits 0 constituent la queue. A la sortie du décodeur, la séquence codée est

$$\underline{X} = (11, 10, 00, 01, 01, 11)$$

L'addition d'une queue à la fin de la séquence d'information diminue le taux de codage. Ainsi, si la longueur de la séquence est égale à L et celle de la queue, à

(K-1), le taux de codage devient

$$R = \frac{L}{V[L + (K-1)]} \tag{1.1}$$

Si $L \gg K$, $R \approx 1/V$, mais si le bloc d'information n'est pas très grand, l'effet de la queue sur le taux n'est plus négligeable.

1.2.2 Diagramme d'état

Un codeur convolutionnel, qui est une machine à états finis, peut être décrit à l'aide d'un diagramme d'état. L'état du codeur est le contenu des (K- 1) premiers étages du registre: ces (K-1) cellules et le nouveau symbole entré sont suffisants pour déterminer les V symboles à la sortie, ainsi que l'état suivant. Le nombre total d'états distincts est 2^{K-1}.

Par exemple, le diagramme d'état du codeur précédent a quatre états et deux branches entrent et sortent de chaque noeud (voir fig. 1.2). Ces branches représentent les transitions du codeur d'un état à l'autre. Les V symboles codés obtenus à chaque transition, ainsi que le bit d'information ayant causé cette transition, sont indiqués sur les branches. A la figure 1.2, le bit d'information est écrit entre parenthèses.

1.2.3 Arbre

Un codeur convolutionnel peut aussi être représenté par un arbre. Chaque noeud de l'arbre de la figure 1.3 caractérise un état du codeur et deux branches émergent de chacun d'entre eux: la branche supérieure correspond à un bit d'information 0 et la branche inférieure à un bit 1. Les branches portent chacune V symboles codés et leur extrémité ou noeud caractérise le nouvel état du codeur. Une séquence

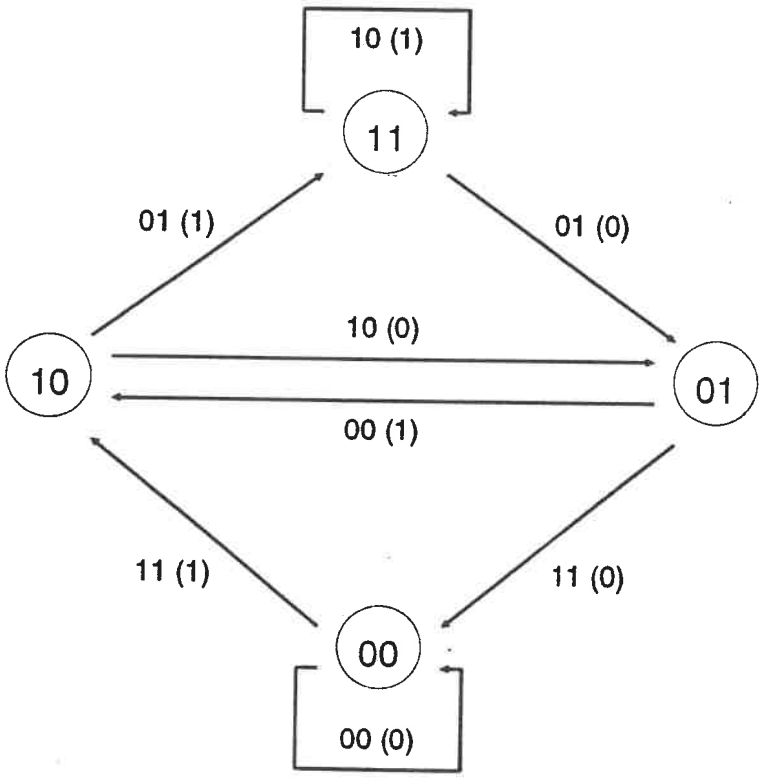

Figure 1.2: Diagramme d'état du codeur de la figure 1.1

d'information correspond donc à un chemin dans l'arbre. Le chemin correspondant à la séquence $\underline{U} = (1, 0, 1, 1, 0, 0)$ est indiqué en gras sur la figure 1.3.

1.2.4 Treillis

Une observation attentive de l'arbre montre que la structure se répète après (K- 1) niveaux. En effet, au niveau L, L $\geq$ K-1, l'arbre a 2^L noeuds, mais le codeur a seulement $2^{K-1} = 4$ états distincts: plusieurs noeuds doivent donc correspondre au même état. On dit que les chemins ont convergé. En tenant compte de ces convergences, on peut redessiner l'arbre en fusionnant les noeuds de même niveau

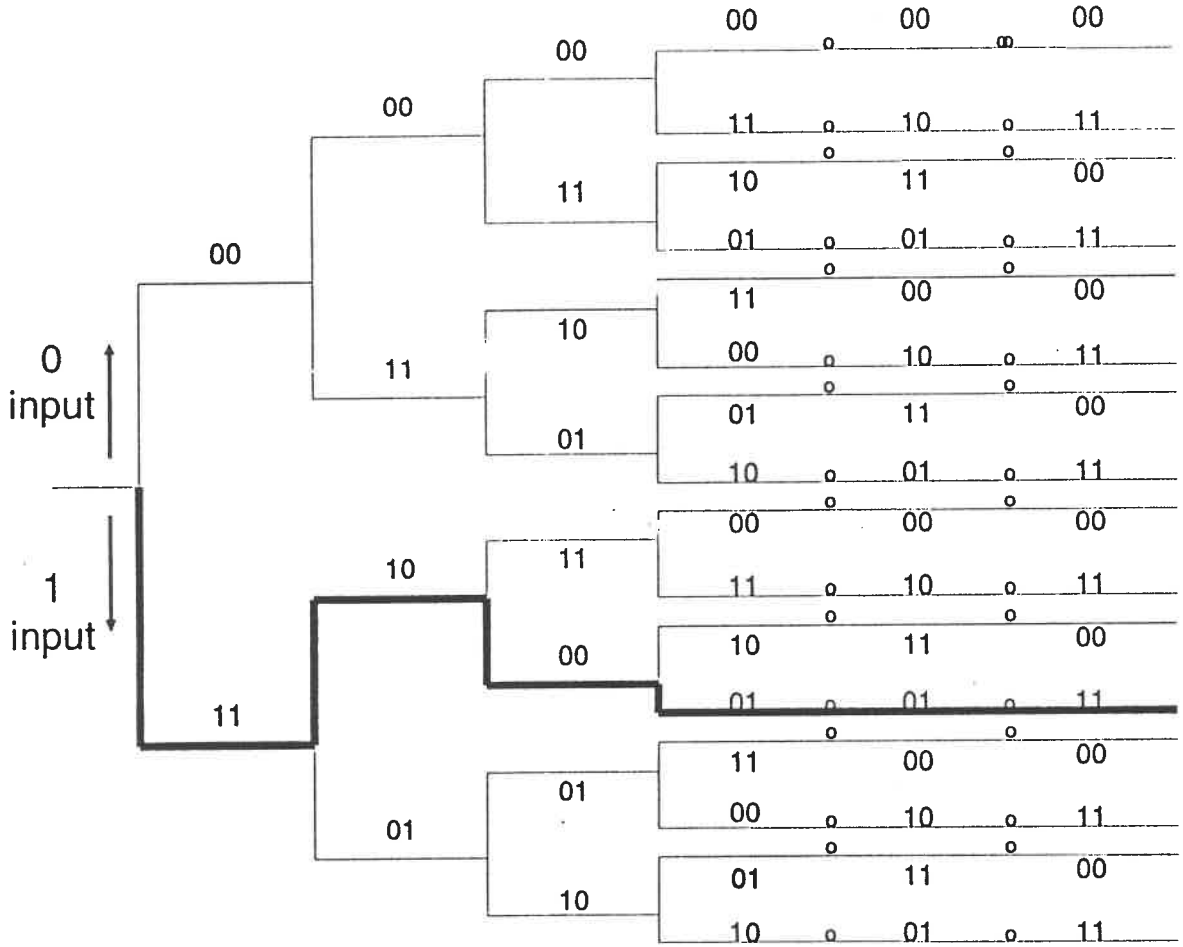

Figure 1.3: Arbre du codeur de la figure 1.1

qui correspondent au même état. On obtient alors un treillis qui n'a plus la redondance de l'arbre (voir figure 1.4). Le treillis a 2^{K-1} noeuds pour toute profondeur supérieure ou égale à (K-1).

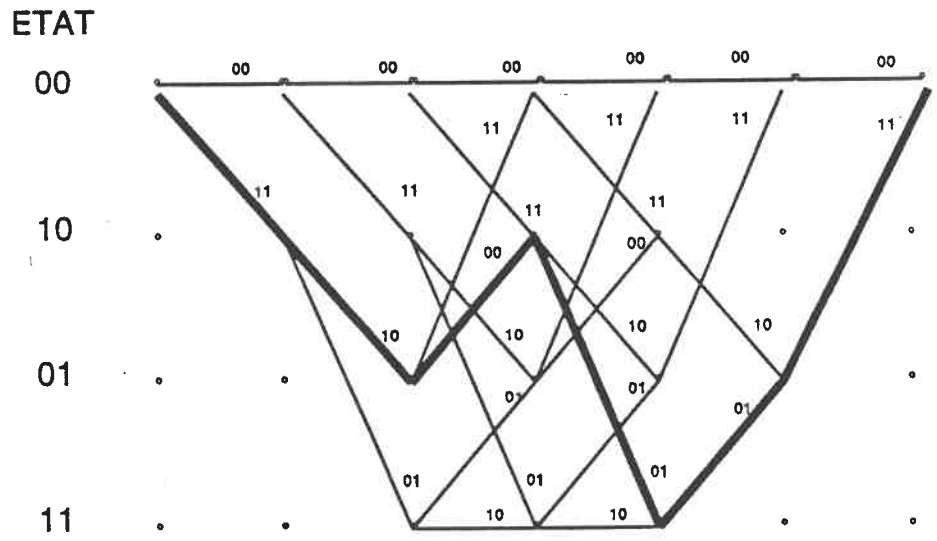

Figure 1.4: Treillis correspondant à l'arbre de la figure 1.3

1.2.5 Propriétés de la distance des codes convolutionnels

La distance de Hamming entre deux mots de code ou séquences ayant n branches est le nombre de positions dans lesquelles les deux mots diffèrent.

La fonction de distance des colonnes d'ordre n, notée $d_c(n)$, est la distance minimale entre toutes les paires de mots de code de longueur n branches, qui sont issus du même point et qui divergent de ce point. La distance minimale d_{min} du code est la distance minimale entre deux mots de code de longueur K branches, où K est la longueur de contrainte du code. La distance libre d_{free} est la distance

de Hamming minimale entre deux chemins quelconques dans l'arbre ou le treillis. Ainsi,

$$d_{min} = d_c(K) \tag{1.2}$$

et

$$d_{free} = \lim_{n \to \infty} d_c(n) \tag{1.3}$$

Il n'existe pas de bornes connues sur la longueur requise des deux chemins pour obtenir d_{free}, mais en général, la distance cesse de croître après quelques longueurs de contrainte. Il est évident que

$$d_{min} \leq d_{free} \tag{1.4}$$

La distance libre d_{free} est la mesure de la performance d'un code. Viterbi [4] a montré que la probabilité d'erreur d'un code convolutionnel décroît exponentiellement avec d_{free} (voir chapitre 2). Comme d_{free} augmente avec la longueur de contrainte K, un code sera puissant et sa performance sera d'autant meilleure que K est grand. Une liste des meilleurs codes de taux 1/2 [11] est donnée à l'Annexe A. Ces codes, qui ont une distance libre maximale, sont ceux qui vont être utilisés dans les simulations subséquentes.

Si on considère une séquence d'information, dont la longueur est supérieure ou égale à la longueur nécessaire pour obtenir d_{free}, le pouvoir de correction d'un code, c'est-à-dire le nombre maximal de symboles erronés qu'un code peut corriger, est:

$$t = \left\lfloor \frac{d_{free} - 1}{2} \right\rfloor \tag{1.5}$$

où $\lfloor x \rfloor$ est la partie entière de x.

1.3 Résumé du mémoire et contributions

1.3.1 Plan du mémoire

Au chapitre 2, la mesure de vraisemblance utilisée pour le décodage probabiliste est introduite. Les deux techniques traditionnelles de décodage probabiliste, l'algorithme de Viterbi et le décodage séquentiel, sont décrites et leurs inconvénients principaux sont mentionnés.

Le chapitre 3 présente trois algorithmes qui ont été récemment proposés pour remplacer les deux techniques précédentes: l'algorithme M-chemins, l'algorithme M-piles et l'algorithme M. Bien que des améliorations soient obtenues, ces algorithmes ne sont pas entièrement satisfaisants.

Cela nous a incité à déterminer un nouvel algorithme de décodage probabiliste au chapitre 4. Les avantages d'un algorithme adaptatif, dont l'effort de décodage s'adapte au bruit du canal, sur un algorithme dont l'effort est constant et supérieur à l'effort moyen du décodeur adaptatif, sont exposés et expliquent pourquoi nous avons choisi un tel algorithme. Les propriétés et l'opération de cet algorithme adaptatif sont étudiées.

Le chapitre 5 donne les performances de cet algorithme, telles qu'obtenues par analyse et simulation. Nous nous intéressons plus particulièrement à la probabilité d'erreur, à la complexité et à la taille maximale du tampon d'entrée. En les comparant aux performances de l'algorithme de Viterbi, nous constatons que cet algorithme est supérieur à ce dernier.

1.3.2 Contributions

Un nouvel algorithme adaptatif de décodage probabiliste est déterminé dans ce mémoire. Cet algorithme, appelé l'Algorithme de Viterbi Adaptatif (AVA), satisfait les propriétés désirées: il présente les avantages de l'algorithme de Viterbi (performance d'erreur quasi-optimale) et du décodage séquentiel (effort de calcul faible en moyenne et indépendant de la longueur de contrainte).

L'influence de la longueur de l'historique du chemin, du nombre maximal de survivants et du seuil de rejet sur la performance est examinée. On peut ainsi déterminer comment choisir ces paramètres pour que la performance de l'AVA soit optimale.

La probabilité de premier événement-erreur de cet algorithme est calculée et comparée à celle de l'algorithme de Viterbi. Cette analyse permet de montrer que la dégradation de la performance de l'AVA par rapport à l'AV est minime. Les simulations confirment que la performance d'erreur est proche de celle d'un décodeur de Viterbi, mais l'effort de calcul moyen requis est moindre.

Nous avons déduit que la fonction de répartition de l'effort de calcul suit une loi exponentielle. Par conséquent, la variabilité de l'effort de calcul ne constitue pas un problème car la décroissance est exponentielle (donc rapide). De plus, il s'ensuit que la file d'attente du tampon d'entrée suit aussi une loi exponentielle. Un tampon de taille raisonnable est donc suffisant pour éviter tout débordement.

Finalement, de nombreux résultats de simulation sont fournis pour des codes ayant un taux R=1/2 et une longueur de contrainte $K \leq 12$. Le canal considéré est un BSC ("Binary Symmetric Channel"). En particulier, la probabilité d'erreur, l'effort de calcul moyen et la taille maximale de la file d'attente sont considérés. Ces données contribuent à la compréhension de l'opération de ce nouvel algorithme et donnent sa performance à différentes valeurs du rapport signal-à-bruit. Nous

pouvons ainsi déterminer quel code choisir et quel est l'effort de calcul moyen du décodeur pour obtenir une certaine probabilité d'erreur. Les simulations montrent qu'un décodeur, dont l'effort de calcul moyen est similaire à celui d'un décodeur de Viterbi ayant K=7, peut utiliser un code ayant une longueur de contrainte plus élevée et ainsi, donner une meilleure performance d'erreur.

Chapitre 2

Décodage probabiliste des codes convolutionnels

Il existe deux moyens de décoder les codes convolutionnels: le décodage probabiliste et le décodage à seuil. Le décodage probabiliste consiste à déterminer à partir de la séquence reçue, la séquence d'information transmise la plus vraisemblable. Les deux techniques de décodage probabiliste les plus utilisées sont l'algorithme de Viterbi et le décodage séquentiel, qui seront décrits dans cette section. Certains codes, satisfaisant des propriétés d'orthogonalité, peuvent être décodés en utilisant le décodage à seuil et en effectuant des opérations algébriques. On considère les séquences d'entrée seulement sur une longueur de contrainte et non toute leur longueur. Comme la distance minimale d_{min} (pour une longueur de contrainte) est inférieure à la distance libre d_{free}, le décodage à seuil donne une moins bonne performance que le décodage probabiliste. De plus, les codes sont eux-mêmes moins puissants à cause, précisément, de la condition d'orthogonalité. Le décodage à seuil ne sera pas traité dans ce mémoire.

2.1 Décodage à vraisemblance maximale

Le but du décodeur est de déterminer le message transmis à partir de la séquence reçue, en basant sa décision sur certains critères de performance. Le critère le plus logique et le plus pratique est la minimisation de la probabilité d'erreur de la décision. Soit m le message émis par une source, $\underline{X}^{(m)}$ le mot de code transmis et $\underline{Y}$ la séquence reçue. Minimiser la probabilité d'erreur de décodage revient à choisir le message $\hat{m}$ qui satisfait [3]:

$$P(\hat{m}/\underline{Y}) \geq P(m'/\underline{Y}) \qquad \forall m' \neq \hat{m} \tag{2.1}$$

Une erreur de décodage est commise si $\hat{m} \neq m$. Comme:

$$P(m/\underline{Y}) = P(\underline{X}^{(m)}/\underline{Y}) = \frac{P(m)P(\underline{Y}/\underline{X}^{(m)})}{P(\underline{Y})}$$

où P(m) est la probabilité à priori du message m. L'équation 2.1 peut donc s'écrire:

$$P(\hat{m})P(\underline{Y}/\underline{X}^{(\hat{m})}) \geq P(m')P(\underline{Y}/\underline{X}^{(m')}) \qquad \forall m' \neq \hat{m} \tag{2.2}$$

Si les messages sont équiprobables, nous obtenons:

$$P(\underline{Y}/\underline{X}^{(\hat{m})}) \geq P(\underline{Y}/\underline{X}^{(m')}) \tag{2.3}$$

Les probabilités conditionnelles $P(\underline{Y}/\underline{X}^{(m)})$ sont appelées les fonctions de vraisemblance. Un décodeur à vraisemblance maximale choisit le message maximisant cette fonction. Si les messages sont équiprobables, les équations 2.1 et 2.3 sont équivalentes et le décodeur à vraisemblance maximale minimise la probabilité d'erreur. Ainsi, maximiser la vraisemblance revient à minimiser cette probabilité.

Pour une branche i, un canal discret sans mémoire et un code de taux 1/V [1],

$$\begin{aligned} P(\underline{Y}_i/\underline{X}_i^{(m)}) &= P(y_{i1}/x_{i1}^{(m)})P(y_{i2}/x_{i2}^{(m)}) \cdots P(y_{iV}/x_{iV}^{(m)}) \\ &= \prod_{j=1}^{V} P(y_{ij}/x_{ij}^{(m)}) \end{aligned} \tag{2.4}$$

où les x_j sont les V symboles codés pour une branche donnée et les $P(y/x)$, les probabilités de transition du canal.

La fonction de vraisemblance logarithmique γ_i sera utilisée pour la branche i car elle est plus pratique [1]:

$$\gamma_i \triangleq \log P(\underline{y}_i/\underline{x}_i^{(m)}) = \sum_{j=1}^{V} \log P(y_{ij}/x_{ij}^{(m)}) \tag{2.5}$$

Pour les N premières branches du chemin, la fonction est donc:

$$\Gamma_N = \sum_{k=1}^{N} \gamma_k \tag{2.6}$$

Il s'agit de trouver le chemin ayant la fonction de vraisemblance maximale dans le treillis (algorithme de Viterbi) ou dans l'arbre (décodage séquentiel).

2.2 Algorithme de Viterbi

L'algorithme de Viterbi (AV) est une technique pratique et efficace pour trouver dans le treillis le chemin transmis le plus vraisemblable. Comme nous l'avons vu au chapitre 1, l'utilisation du treillis, contrairement à l'arbre, permet d'éliminer la redondance dans les chemins examinés. Pour un canal binaire symétrique, le chemin le plus probable est déterminé en utilisant la fonction de vraisemblance logarithmique, encore appelée métrique, définie ci-dessous [1], [4]. Soit $\underline{Y}$ la séquence de N symboles reçue et $\underline{X}^{(m)}$ une séquence possiblement transmise et qui diffère en d_m symboles de $\underline{Y}$. Donc, la distance de Hamming entre $\underline{X}^{(m)}$ et $\underline{Y}$ est d_m. La fonction de vraisemblance est:

$$\begin{aligned} \log P(\underline{Y}/\underline{X}^{(m)}) &= \log(p^{d_m}(1-p)^{N-d_m}) \\ &= d_m \log p + (N-d_m)\log(1-p) \end{aligned} \tag{2.7}$$

$$
\begin{aligned}
&= N log(1-p) - d_m \log\left(\frac{1-p}{p}\right) \\
&= -A - Bd_m \qquad\qquad A,B > 0 \qquad\qquad (2.8)
\end{aligned}
$$

Maximiser la métrique est donc équivalent à minimiser la distance de Hamming sur un canal binaire symétrique ayant une probabilité de transition $p < 1/2$ [1], [4].

A chaque niveau du treillis, parmi tous les chemins convergeant à un même noeud ou état (si le taux de codage est R=1/V, ce nombre de chemins est deux), seul le meilleur est gardé et devient le survivant à ce noeud. Les autres sont éliminés car après ce niveau, les métriques de tous les chemins issus du même état vont varier de la même façon. La métrique du chemin gardé sera donc toujours supérieure à celles des autres chemins convergents et c'est pour cela que nous les rejetons. Les garder ne sert à rien. Cette procédure est répétée pour tous les noeuds d'un même niveau et on obtient donc 2^{K-1} survivants. Ces chemins sont ensuite prolongés et donnent 2^K chemins convergeant deux par deux aux 2^{K-1} noeuds du niveau suivant. Les opérations de prolongation et d'élimination sont poursuivies à chaque niveau du treillis jusqu'au niveau final.

Cet algorithme est optimal [4] car aucun autre chemin ne pourra avoir une métrique meilleure que celle du chemin décodé. L'élimination d'un des deux chemins convergents permet de réduire considérablement l'effort de décodage sans affecter l'optimalité car, comme il a été dit, les chemins éliminés ne pourront jamais être meilleurs que les survivants. Ainsi, pour une séquence ayant N bits, il n'est pas nécessaire de comparer les 2^N métriques des 2^N chemins pour réaliser le décodage à vraisemblance maximale. En utilisant le treillis, on peut seulement considérer 2^{K-1} chemins.

Si les deux chemins convergents ont la même métrique, le survivant est choisi au hasard. La probabilité d'avoir pris la bonne décision à ce noeud est donc de 50%.

Une queue de (K-1) 0 est encodée et transmise à la fin de la séquence d'information pour déterminer quel survivant, parmi les 2^{K-1}, est le chemin décodé. En décodant la queue, seule la branche correspondant au bit 0 est prolongée. Le nombre de survivants est donc réduit par deux et après (K-1) branches, il reste un seul chemin dans le treillis: c'est le chemin le plus vraisemblable, celui qui est à une distance minimale de la séquence reçue. Il est possible qu'un autre chemin ait la même distance mais qu'il ait été éliminé arbitrairement au noeud où les deux chemins ont convergé. Ceci constitue une limite de l'AV.

La figure 2.1 illustre l'opération de décodage à l'aide d'un exemple. Nous supposons que la séquence transmise est la séquence codée du chapitre 1

$$\underline{X} = (11, 10, 00, 01, 01, 11)$$

et que la séquence reçue a subi deux transitions à cause du bruit:

$$\underline{Y} = (01, 10, 00, 01, 00, 11)$$

Le premier et le dixième symbole codé ont subi des transitions. Une croix sur un chemin signifie que ce chemin est éliminé car sa distance est plus grande que celle de l'autre chemin qui converge au même noeud. Le chemin a été décodé sans erreur malgré les deux symboles codés erronés.

2.2.1 Simplification de l'algorithme

Le décodage d'un message de N bits requiert $N2^{K-1}$ opérations, où une opération consiste à calculer et à comparer les métriques des deux chemins convergeant au même noeud et à prolonger les chemins survivants. Une mémoire considérable est nécessaire pour stocker les 2^{K-1} chemins survivants de N bits, si N est grand.

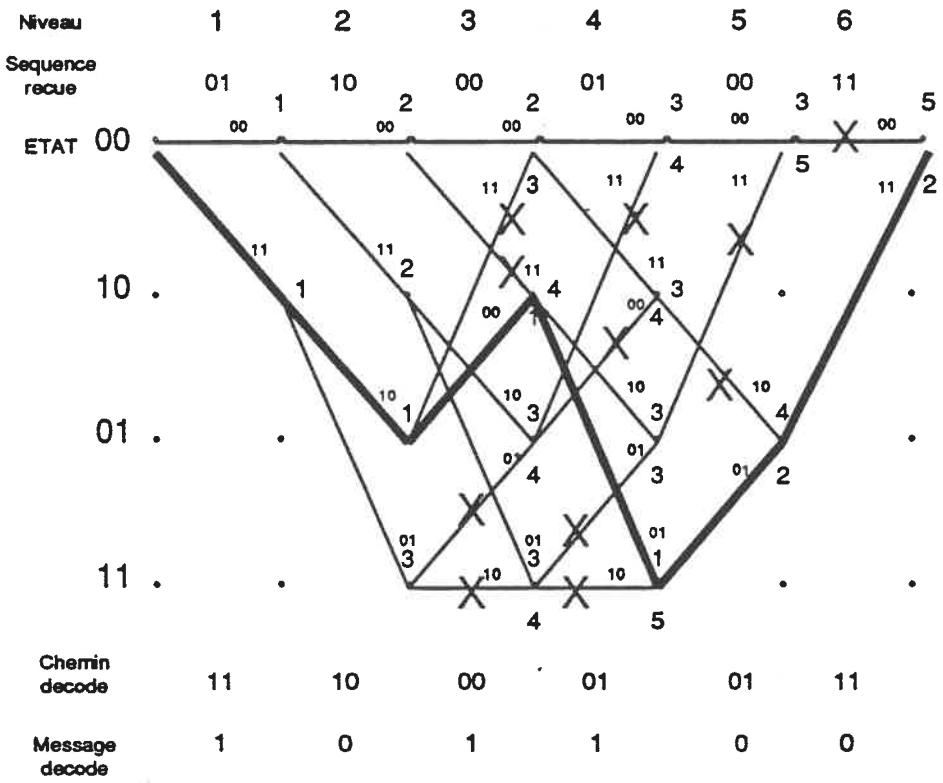

Figure 2.1: Exemple de décodage par l'algorithme de Viterbi

De plus, attendre que les N bits d'information soient décodés avant de délivrer le premier bit à l'usager peut constituer un délai inacceptable.

Cependant, des simulations extensives [12, 13] ont permis d'observer que les chemins survivants proviennent du même noeud situé à plusieurs longueurs de contrainte plus tôt dans le treillis. Soit M le nombre de branches entre le niveau de ce noeud et le niveau présent. Il suffit de conserver une partie du treillis de longueur M et de délivrer les bits d'information décodés au fur et à mesure. M est appelé la longueur de l'historique du chemin du décodeur ou la longueur de récupération.

L'algorithme devient l'algorithme de Viterbi tronqué. Le décodeur va délivrer le bit d'information le plus ancien du chemin survivant ayant la meilleure métrique. Forney [14] a démontré théoriquement qu'avec cette simplification, la dégradation de la probabilité d'erreur moyenne sur celle d'un décodeur optimal non tronqué est négligeable si M est assez grand. Les simulations réalisées par Smith [13] ont montré qu' en pratique, un historique de quatre ou cinq longueurs de contrainte est suffisant. La performance n'est pas améliorée même si la longueur de l'historique est augmentée davantage.

2.2.2 Probabilité d'erreur

Nous allons donner une borne sur la probabilité d'un premier événement-erreur et sur la probabilité d'erreur par bit, qui est le nombre moyen de bits incorrects (causés par un événement-erreur) sur le nombre total de bits d'information. On dit qu'un premier événement-erreur s'est produit au niveau j du treillis si le chemin correct est éliminé pour la première fois à ce niveau. Cela veut dire qu'un chemin incorrect est plus proche (au point de vue de la distance de Hamming) de la séquence reçue que le chemin correct.

Performance générale des codes convolutionnels

La probabilité moyenne d'un premier événement-erreur sur l'ensemble des codes convolutionnels choisis aléatoirement est bornée par [4]:

$$\overline{P(E)} < \frac{2^{-K(R_0/R)}}{1-2^{-[(R_0/R)-1]}} \qquad R < R_0 \tag{2.9}$$

R_0 est un paramètre qui ne dépend que du canal et s'écrit:

$$R_0 = -\log_2 \sum_y \left[\sum_x P(x)\sqrt{P(y/x)} \right]^2$$

où x est le symbole codé transmis, y le symbole reçu et $P(y/x)$ la probabilité de transition du canal.

Une borne sur la probabilité d'erreur par bit est donnée par [4]

$$\overline{P(B)} < \frac{2^{-K(R_0/R)}}{(1-2^{-[(R_0/R)-1]})^2} \tag{2.10}$$

On constate que la probabilité d'erreur des codes convolutionnels décroît exponentiellement avec la longueur de contrainte K. En théorie, elle peut donc être arbitrairement faible si K est grand. Cependant, l'effort de calcul de l'algorithme de Viterbi et la mémoire requise pour stocker les survivants et les métriques croissent exponentiellement avec K. En pratique, il n'est donc pas possible d'utiliser des codes ayant un K élevé et d'obtenir une probabilité d'erreur arbitrairement faible. Une valeur d'environ 8 est considérée la limite pratique pour K [2].

Après avoir considéré les codes convolutionnels en général, il est maintenant intéressant d'analyser un code en particulier. L'analyse suivante s'inspire des travaux de Viterbi [1], [4].

Performance spécifique d'un code

Si nous connaissons le spectre d'un code convolutionnel, c'est-à-dire la distribution des distances de Hamming de tous les chemins qui divergent à l'origine du chemin nul et reconvergent à l'état zéro à un instant ultérieur, et les séquences d'information correspondantes (le nombre de bits 1 correspondant au chemin), la performance d'erreur de ce code peut être analysée. Pour simplifier et sans perte de généralité, nous supposons que la séquence contenant uniquement des zéros est transmise dans un canal binaire symétrique. Comme exemple, le code convolutionnel K=3, de taux R=1/2 est considéré.

Sur la figure 2.2 (a), le chemin incorrect de trois branches est à une distance 5 du

chemin correct: il diffère du chemin correct dans cinq positions. Donc, un premier événement-erreur va se produire si la séquence reçue concorde avec le chemin incorrect dans trois positions ou plus, c'est-à-dire si le bruit dans le canal a causé trois transitions ou plus aux cinq symboles codés en lesquels les deux chemins diffèrent. Si p est la probabilité de transition du canal, la probabilité d'un premier événement-erreur pour ce chemin de distance 5 est [1]

$$P_5 = \sum_{i=3}^{5} \binom{5}{i} p^i (1-p)^{5-i}$$

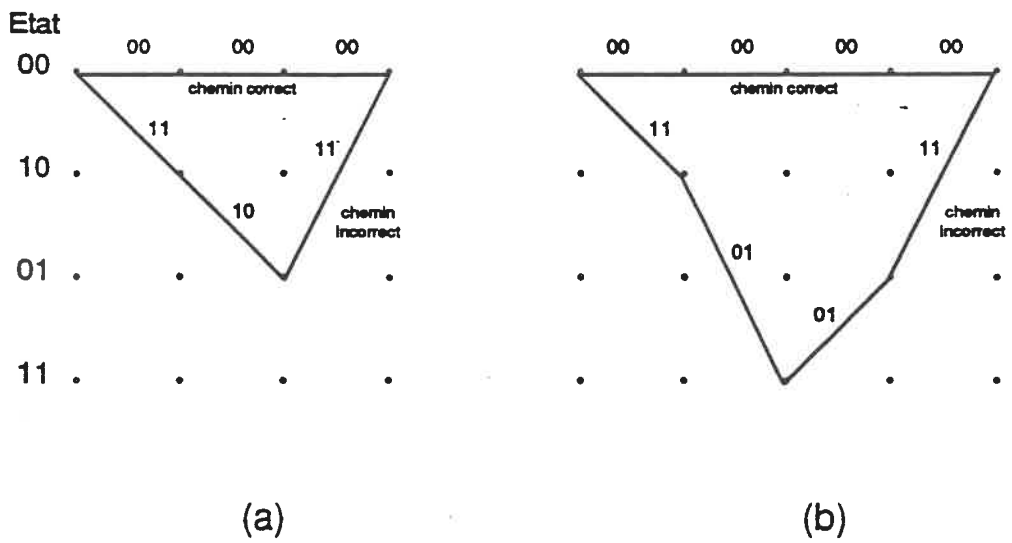

Figure 2.2: Premier événement-erreur pour le code de la figure 1.1. (a) chemin correct de poids impair (b) chemin correct de poids pair
Référence: [1]

Mais il n'est pas certain que ce chemin ait survécu aux niveaux précédents et il se peut que ce soit le chemin de quatre branches, ayant une distance 6 (voir

fig. 2.2 (b)), qui est comparé au chemin correct. S'il y a trois erreurs aux symboles qui sont différents entre les chemins correct et incorrect, les métriques sont égales au noeud de convergence. Comme le survivant est choisi au hasard dans ce cas, la probabilité d'avoir une erreur est de 1/2. S'il y a quatre transitions ou plus, une erreur va résulter sans équivoque et nous obtenons donc [1]

$$P_6 = \frac{1}{2}\binom{6}{3}p^3(1-p)^3 + \sum_{i=4}^{6}\binom{6}{i}p^i(1-p)^{6-1}$$

Par conséquent, si le chemin incorrect est à une distance d du chemin correct, la probabilité P_d d'un premier événement-erreur est [1]

$$P_d = \begin{cases} \sum_{i=\frac{d+1}{2}}^{d}\binom{d}{i}p^i(1-p)^{d-i} & d \text{ impair} \\ \sum_{i=\frac{d}{2}+1}^{d}\binom{d}{i}p^i(1-p)^{d-i} + \frac{1}{2}\binom{d}{d/2}(p(1-p))^{d/2} & d \text{ pair} \end{cases}$$

Une borne supérieure sur la probabilité d'un premier événement-erreur peut être obtenue en faisant la somme des probabilités d'erreur causées par tous les chemins incorrects à une distance l du chemin correct [1]

$$P(E) < \sum_{l=d_{free}}^{\infty} a_l P_l \tag{2.11}$$

où P_l est exprimé par l'équation précédente et a_l est le nombre total de chemins incorrects ayant une distance l.

La probabilité d'erreur par bit s'obtient en multipliant chaque terme de la probabilité d'événement-erreur par le nombre de bits en erreur sur ce chemin incorrect. Si nous supposons que le chemin correct est le chemin n'ayant que des zéros, ce nombre est le nombre de bits d'information 1 du chemin. Soit c_l le nombre total de bits 1 sur tous les chemins à une distance l, alors [1]:

$$P(B) < \sum_{l=d_{free}}^{\infty} c_l P_l \tag{2.12}$$

On voit que plus la distance libre est grande, plus la probabilité d'erreur est faible. Par conséquent, en décodage de Viterbi, pour un K donné, le code ayant la distance libre maximale est utilisé. Une liste de ces codes optimaux est donnée à l'Annexe A.

2.3 Décodage séquentiel

Nous avons vu dans la section précédente que l'AV ne peut être utilisé que pour des codes ayant une longueur de contrainte assez petite ($K \leq 8$). Par conséquent, si on veut utiliser des codes plus puissants (ayant une plus grande longueur de contrainte) pour améliorer la performance, on doit recourir à une méthode sous-optimale. Le décodeur séquentiel explore seulement une partie de l'arbre et permet de trouver le meilleur chemin parmi les différents chemins examinés.

Dans un canal sans mémoire, la métrique de branche utilisée est donnée par [1]:

$$\gamma_j = \sum_{i=1}^{V} \left[\log_2 \frac{P(y_{ji}/x_{ji})}{P(y_{ji})} - R \right] \tag{2.13}$$

où x_{ji} est le symbole codé transmis, y_{ji}, le symbole reçu, $P(y_{ji}/x_{ji})$, la probabilité de transition du canal pour les symboles x_{ji} et y_{ji} et $R = 1/V$ est le taux de codage. Pour un chemin de longueur N branches, la métrique totale devient:

$$\Gamma = \sum_{j=1}^{N} \gamma_j$$

Cette métrique tend à croître le long du chemin correct.

L'algorithme de Zigangirov-Jelinek, l'AZJ, et l'algorithme de Fano sont les deux principaux algorithmes de décodage séquentiel. Seul le premier, à cause de sa simplicité, sera considéré dans ce mémoire. Une pile est utilisée pour stocker l'information sur tous les chemins examinés, qui sont placés par ordre décroissant

de leur métrique. Le sommet de la pile contient donc le meilleur chemin, dénoté par le noeud de son extrémité. Ce chemin est prolongé d'une branche et retiré de la pile. Les métriques de ses successeurs sont calculées et les deux nouveaux chemins sont insérés dans la pile. Le décodage est fini lorsque le chemin au sommet de la pile est de même longueur que la séquence reçue.

Comme exemple, la séquence de la section précédente est décodée en utilisant l'AZJ (voir figure 2.3). D'après l'équation 2.13, les métriques de branche sont

$$\begin{aligned} \gamma_j &= 1 + 2\log_2 p = -8.29 \qquad \text{(2 symboles différents)} \\ \gamma_j &= 1 + \log_2 p(1-p) = -3.70 \qquad \text{(1 symbole différent)} \\ \gamma_j &= 1 + 2\log_2(1-p) = 0.88 \qquad \text{(0 symbole différent)} \end{aligned}$$

si p=0.04. Les valeurs de ces métriques sont arrondies à -9, -4 et +1. Les noeuds insérés dans la pile, ainsi que les métriques sont donnés ci-dessous:

Etape #	Contenu de la pile: Noeud(métrique)
1	0(0)
2	1(-4), 2(-4)
3	2(-4), 3(-8), 4(-8)
4	5(-3), 3(-8), 4(-8), 6(-13)
5	8(-2), 3(-8), 4(-8), 7(-12), 6(-13)
6	10(-1), 3(-8), 4(-8), 9(-12), 7(-12), 6(-13)
7	11(-5), 3(-8), 4(-8), 9(-12), 7(-12), 6(-13)
8	12(-4), 3(-8), 4(-8), 9(-12), 7(-12), 6(-13)

Le chemin décodé, celui qui est au sommet de la pile, est le chemin dont l'extrémité est le noeud 12. C'est le même chemin qui a été obtenu avec l'algorithme de Viterbi.

En pratique, il est impensable d'insérer les nouveaux noeuds à leur place exacte

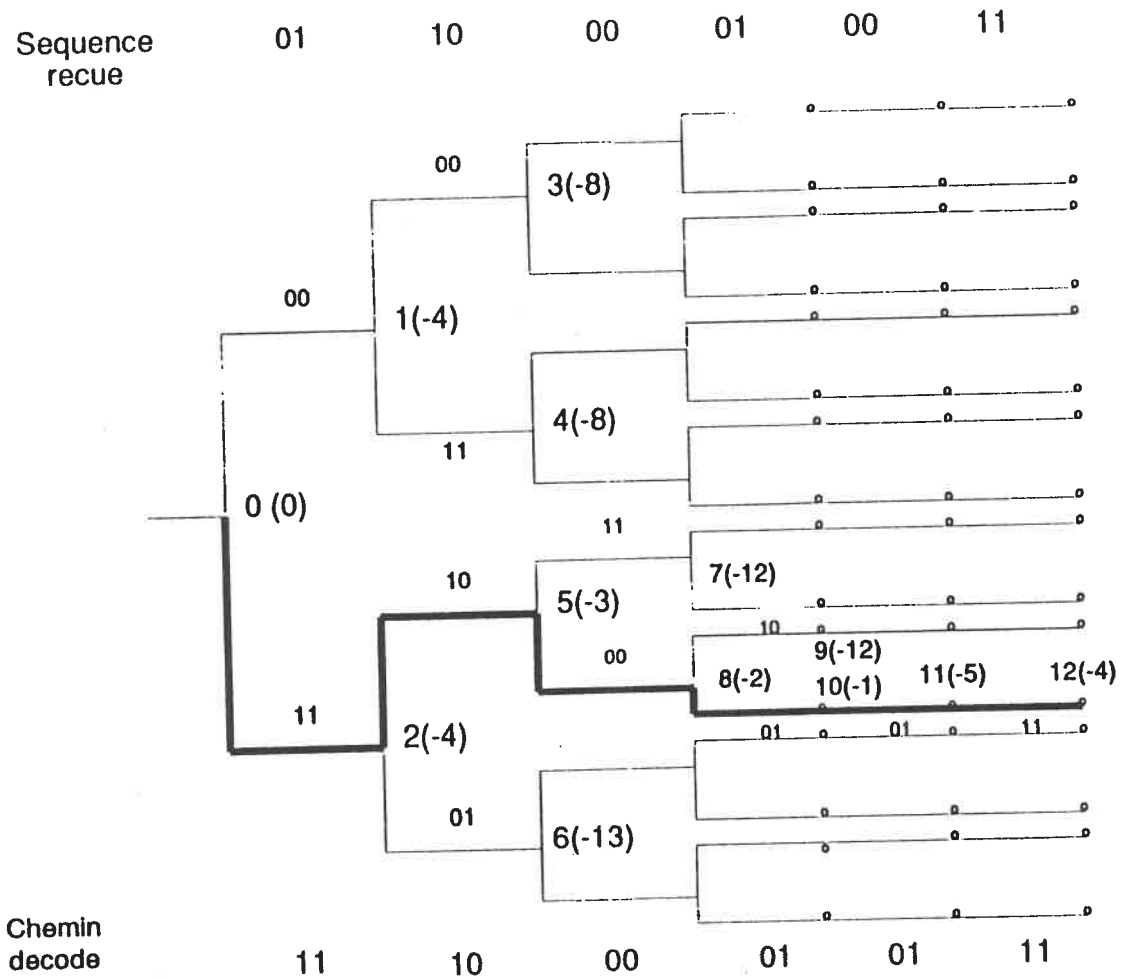

Figure 2.3: Décodage par l'algorithme de Zigangirov-Jelinek

dans la pile comme nous l'avons fait dans l'exemple précédent, car cela nécessiterait une mise en ordre (sorting) qui prendrait trop de temps. Aussi, des sous-piles de taille H sont utilisées et un noeud de métrique Γ est inséré dans la sous-pile Q si: $QH \leq \Gamma < (Q+1)H$ Le chemin à prolonger est celui qui a été entré en dernier dans la meilleure sous-pile (ayant le plus grand Q).

Le problème majeur de cet algorithme est la variabilité de l'effort de décodage. En effet, le bruit du canal peut provoquer une chute de la métrique du chemin correct. Un autre chemin, incorrect, va donc se trouver au sommet de la pile et va être prolongé. Mais quand le bruit dans le canal cesse, le métrique de ce chemin incorrect décroît. Cependant, le décodeur ne va prolonger le chemin correct à nouveau que s'il n'y a pas d'autres chemins ayant une meilleure métrique. L'examen de chemins incorrects est une perte de temps: le décodeur est obligé de retourner en arrière pour explorer des chemins incorrects au lieu de continuer sur le bon. Par conséquent, l'effort de calcul, bien que faible en moyenne, est très variable.

A cause de cette variabilité, il se peut que le décodeur n'ait pas le temps de traiter les données provenant du canal en temps réel. Ces données doivent donc être stockées dans un tampon d'entrée en attendant. Un tampon de sortie est aussi nécessaire pour régulariser le débit de sortie.

Le nombre de calculs C pour décoder un bit est une variable aléatoire ayant asymptotiquement une distribution de type Pareto [1]:

$$P(C \geq N) \approx \beta N^{-\alpha} , \qquad N \gg 1 \tag{2.14}$$

où β et α (l'exposant de Pareto) ne dépendent que du canal et du taux de codage R. Cette distribution de Pareto est le résultat de l'effet combiné de deux comportements exponentiels: la probabilité d'avoir une chute de la métrique du chemin correct supérieure à d décroît exponentiellement avec d (pour un canal sans mémoire,

la probabilité d'une salve de bruit décroît exponentiellement avec la durée de la salve), tandis que le nombre de calculs dus à la chute d croît exponentiellement avec d. La fonction de distribution cumulative 2.14 indique que le nombre de calculs ne décroît pas rapidement avec N: la décroissance est seulement algébrique et non exponentielle. Cela constitue l'inconvénient principal du décodage séquentiel.

Si $\alpha \leq 1$, le nombre moyen de calculs C tend vers l'infini et le décodage est donc pratiquement impossible: longs retours en arrière, débordement des tampons et de la pile. Ce comportement a été vérifié expérimentalement par des simulations [1, 15]. Le taux de codage R_{comp} correspondant à $\alpha = 1$ est appelé le *taux de coupure* du décodage séquentiel et on évalue approximativement α par

$$\alpha \approx \frac{R_{comp}}{R} \tag{2.15}$$

R_{comp} ne dépend que du canal et peut se calculer facilement. Il faut donc faire fonctionner le décodeur séquentiel à un taux R/R_{comp} inférieur à 1, alors qu'il n'y a pas de taux de coupure si l'AV est utilisé.

Débordement

Quelle que soit la taille du tampon d'entrée, la probabilité que le tampon déborde et que cela conduise à des effacements est non nulle. Pour un arbre ayant N branches, la probabilité de débordement est donnée par [1]:

$$P(\textit{débordement}) \approx N(GB)^{-\alpha} \tag{2.16}$$

où B est la taille du tampon, α, l'exposant de Pareto et G, le gain de vitesse du décodeur. Le gain de vitesse est le nombre d'opérations que le décodeur peut accomplir pendant le temps d'interarrivée de deux bits du canal. A cause de la distribution de Pareto, la probabilité de débordement ne varie que lentement avec

la taille du tampon; il est donc difficile d'éliminer ce problème. Par conséquent, des procédures de recouvrement doivent être prévues, telle une demande de retransmission des blocs sur le point de déborder. De plus, les données doivent être divisées en blocs de 500 ou 1000 bits.

Probabilité d'erreur

Le décodage séquentiel étant sous-optimal, la borne inférieure de la probabilité d'erreur est nécessairement la probabilité d'erreur de l'algorithme optimal de Viterbi. Pour des taux $R < R_{comp}$, une borne supérieure est donnée par [1]:

$$P(E) < Ae^{KR_{comp}/R} \tag{2.17}$$

où A est une constante et K est la longueur de contrainte du code.

Une grande longueur de contrainte ne cause pas de problème en décodage séquentiel car la complexité ne dépend que linéairement de K (alors que la dépendance est exponentielle pour l'AV). On peut donc utiliser des codes très puissants et avoir une probabilité d'erreur arbitrairement faible. Mais l'expression ci-dessus n'inclut pas les erreurs causées par les débordements. La probabilité de débordement (de l'ordre de 10^{-2} à 10^{-3}) est beaucoup plus grande que la probabilité d'erreur et c'est elle qui constitue l'inconvénient principal de cette méthode.

Pour résumer, l'effort de calcul considérable, qui croît exponentiellement avec la longueur de contrainte, représente le désavantage principal de l'algorithme de Viterbi. Pour le décodage séquentiel, c'est la variabilité de l'effort de calcul qui cause des problèmes. Des variantes de ces deux méthodes de décodage ont été proposées pour pallier aux inconvénients que nous venons de mentionner. Ces variantes sont décrites dans le chapitre suivant.

Chapitre 3

Variantes des algorithmes de décodage

Nous avons vu au chapitre précédent les inconvénients des deux principales techniques de décodage probabiliste, l'algorithme de Viterbi et le décodage séquentiel pour décoder des codes ayant une longueur de contrainte supérieure à 7: le premier nécessite une trop grande complexité tandis que l'effort de calcul pour le deuxième est trop variable. Dans ce chapitre, trois algorithmes, qui ont été proposés pour remplacer ces méthodes 'classiques' de décodage probabiliste, sont présentés.

3.1 Algorithme M-chemins

L'algorithme M-chemins ou multi-chemins (AMC), développé et analysé par Haccoun [6], est une modification de l'algorithme de Zigangirov-Jelinek. Au lieu de prolonger un seul chemin, plusieurs chemins, se trouvant au sommet de la pile, sont prolongés simultanément. De plus, une opération d'épuration est ajoutée à l'AZJ: si deux chemins convergent à un même noeud, les deux métriques sont comparées et

seul le meilleur chemin est conservé. Comme en décodage de Viterbi, l'autre chemin est éliminé. Cela permet de réduire l'effort de décodage (examiner un chemin incorrect est complètement inutile) et l'espace requis dans la pile. Si le nombre M de chemins à prolonger est fixe, l'élimination d'un chemin, que l'on sait improbable, permet aussi d'éviter que ce chemin soit gardé au détriment d'un autre, qui pourrait s'avérer être le chemin correct.

En prolongeant M chemins à la fois au lieu d'un seul, la probabilité que le chemin correct se trouve parmi ces M chemins est bien sûr augmentée. Si le bruit dans le canal est élevé, il est possible que la métrique du chemin correct chute tellement que ce chemin ne soit plus parmi les M meilleurs. Dans ce cas, comme pour l'AZJ, le décodeur doit retourner en arrière mais la probabilité que cela arrive est plus petite. L'utilisation de cet algorithme M-chémins réduit donc le nombre de retours en arrière et la variabilité de l'effort de calcul.

Le nombre de calculs pour décoder un bit est plus grand en moyenne que celui de l'AZJ, mais il est moins variable. La probabilité de nécessiter plus de (M+1) calculs pour décoder une branche est plus grande avec l'AZJ et les relations suivantes sont obtenues [6]:

$$\overline{C}_{AMC} \approx (M-1) + \overline{C}_{AZJ} \tag{3.1}$$

$$P(C_{AMC} \geq M+1) \leq P(C_{AZJ} \geq M+1) \tag{3.2}$$

où $\overline{C}$ représente le nombre moyen de calculs par bit décodé.

L'AMC étant sous-optimal, la probabilité d'erreur de cet algorithme est supérieure à celle d'un algorithme optimal, l'AV. De plus, comme il explore une plus grande partie de l'arbre que l'AZJ, il améliore la probabilité d'erreur de l'AZJ. Haccoun [6] a démontré que:

$$P(E)_{AV} \leq P(E)_{AMC} \leq P(E)_{AZJ} \tag{3.3}$$

Augmenter M permet de réduire la probabilité d'erreur et la variabilité de l'effort de calcul mais cela requiert un effort de calcul moyen plus grand. Cet algorithme peut être considéré comme une généralisation des algorithmes de décodage probabiliste [6]. L'algorithme de Viterbi et l'algorithme de Zigangirov-Jelinek semblent très différents en apparence mais en fait, ils ne sont que les deux cas particuliers extrêmes d'un algorithme généralisé à pile. Si M=1, on obtient l'AZJ et si M=2^{K-1}, l'AV. Donc, si M tend vers 2^{K-1}, on s'approche des caractéristiques de l'algorithme de Viterbi (variabilité nulle, performance d'erreur optimale), tandis que si M est très petit, on obtient plutôt l'AZJ.

En conclusion, l'AMC améliore la variabilité et la probabilité d'erreur du décodage séquentiel au détriment d'un effort de calcul plus grand en moyenne. Cependant, tant que $M < 2^{K-1}$, l'effort de calcul a toujours asymptotiquement une distribution de type Pareto et le problème majeur du décodage séquentiel (effort de calcul variable entrainant des débordements) n'est pas résolu.

3.2 Algorithme M-piles

L'algorithme M-piles, l'AMP, introduit par Chevillat et Costello [8] est aussi une modification de l'AZJ. Plusieurs piles sont utilisées de sorte que les débordements et effacements de l'AZJ sont pratiquement éliminés.

Au début, cet algorithme fonctionne de la même façon que l'AZJ: le noeud d'origine, placé dans la première pile, est étendu. Si un noeud terminal dans l'arbre est atteint sans que la première pile soit remplie, le décodage est fini et le chemin de l'origine à ce noeud final devient la séquence décodée. Dans ce cas, l'AMP a suivi exactement les mêmes étapes que l'AZJ et le résultat du décodage est donc identique.

Cependant, si la séquence reçue nécessite de longs retours en arrière et de nombreux calculs, la première pile va être pleine. Les meilleurs T noeuds de cette pile sont alors transférés dans une deuxième pile et le décodage se poursuit dans la deuxième pile en utilisant seulement ces noeuds transférés.

Si un chemin dans la deuxième pile arrive à la fin de l'arbre avant que la pile soit pleine, le noeud final est gardé comme décision provisoire dans un registre spécial. Les noeuds restant dans la deuxième pile sont éliminés et le décodage continue dans la première pile (où il y a maintenant T places libres). Si le décodeur atteint un noeud terminal avant que la première pile soit de nouveau pleine, la métrique du nouveau noeud terminal est comparée à celle de la décision provisoire. Le noeud ayant la meilleure métrique est considérée comme la décision finale de décodage. Il a été montré que cette décision est au moins aussi bonne que celle prise par l'AZJ ayant une pile arbitrairement grande [8].

Toutefois, si la première pile est encore remplie avant que la fin de l'arbre soit atteinte, une nouvelle deuxième pile est formée en transférant les meilleurs T noeuds de la première pile. Des piles additionnelles sont créées de la même façon jusqu'à ce qu'une décision provisoire soit prise. Le décodeur compare toujours la métrique d'un nouveau noeud terminal avec le contenu du registre de décision provisoire. Le meilleur noeud final est gardé et la pile est effacée. Le décodage se poursuit dans la pile précédente et s'arrête si la première pile atteint la fin de l'arbre.

Comme on l'a vu, l'idée de base de l'AMP est d'utiliser des piles additionnelles, plus petites que la première. Ainsi, si le canal est bruyant, la première pile va être pleine et le décodage se poursuit avec seulement les T meilleurs noeuds dans la deuxième pile. Comme il y a moins de noeuds, la probabilité de retourner en arrière est forcément réduite (il y a moins de chemins qui peuvent avoir une meilleure métrique que le chemin présentement examiné) et par conséquent, une décision

provisoire peut être obtenue rapidement. De plus, les T noeuds transférés sont probablement à un niveau plus avancé dans l'arbre. Contrairement à l'AZJ, la profondeur est donc avantagée et non pas seulement la métrique quand il s'agit de prolonger un noeud.

Un effacement ne peut donc pratiquement jamais se produire sans qu'une décision provisoire ne soit disponible. Malheureusement, cet algorithme a toujours les inconvénients du décodage séquentiel. L'effort de calcul est toujours variable (bien que la distribution soit exponentielle maintenant [8]) et il faut toujours grouper les données en blocs pour qu'un noeud terminal puisse être atteint. Une décision finale pour décoder un bloc difficile peut prendre un certain temps, aussi impose-t-on une limite au nombre de calculs maximal par bloc. L'inconvénient majeur de l'AMP est surtout la mémoire.

En bref, l'effort de calcul moyen est certainement plus petit que pour l'AV mais la mémoire requise est considérable.

3.3 Algorithme M

L'algorithme M ou AM a été proposé par Jelinek et Anderson pour le codage de source [17]. Cet algorithme a été utilisé par Lin et Anderson [9] pour les codes convolutionnels et par Aulin [10] pour la modulation codée. Contrairement aux deux algorithmes que nous venons de décrire, l'AMC et l'AMP, les retours en arrière ne sont pas permis: le décodeur doit toujours avancer dans le treillis. De plus, l'effort de calcul est constant: la variabilité des algorithmes précédents et les inconvénients que cela entraîne sont donc complètement éliminés.

L'idée de base de l'AM est d'utiliser le treillis (comme l'AV) et de garder seulement M noeuds à chaque niveau (au lieu de 2^{K-1}). Si le canal est bon, l'AV

accomplit beaucoup de travail inutile: la plupart des chemins examinés sont très peu probables d'être le chemin correct. En éliminant ces chemins le plus tôt possible, une économie appréciable peut donc être obtenue et le décodage peut se faire beaucoup plus vite.

Nous allons maintenant étudier cet algorithme en détail. Comme pour le décodage de Viterbi, le noeud d'origine, qui est connu, a une distance de Hamming ou métrique 0 et les (M-1) autres noeuds, une distance beaucoup plus grande, +50 par exemple. Ces M noeuds sont étendus et on obtient 2M noeuds et métriques au niveau suivant du treillis. S'il y a convergence, c'est-à-dire si deux noeuds sont identiques, celui qui a une distance plus grande est éliminé. Parmi les noeuds qui restent, les meilleurs M (ceux qui ont les distances les plus faibles), appelés survivants, sont sélectionnés et conservés. Si la longueur de l'historique du chemin est H, le bit situé H niveaux avant dans le treillis est délivré, comme pour l'algorithme de Viterbi tronqué. Le décodage se poursuit jusqu'à ce qu'on arrive au bout du treillis.

Il est évident que si M=2^{K-1}, on obtient l'AV. Cet algorithme peut donc être considéré comme l'algorithme de Viterbi où seulement M noeuds, et non tous les noeuds, sont gardés.

Effort de calcul

Le procédé complexe de mise en ordre qu'on rencontre avec les algorithmes à pile est maintenant complètement éliminé. Cependant, cet algorithme nécessite une étape importante de sélection: choisir M parmi 2M chemins. Soit $V_t(n)$ le nombre de comparaisons nécessaires pour trouver le t-ième plus grand élément parmi n. Des

algorithmes existent pour lesquels [16]:

$$E[V_t(n)] = n + t + f(n) \tag{3.4}$$

où $\lim_{n\to\infty} \frac{f(n)}{n} = 0$. Utilisant un tel algorithme, la M-ième meilleure métrique parmi 2M peut être trouvée avec seulement

$$E[V_M(2M)] = 2M + M + f(2M)$$

comparaisons. Dès que la métrique du M-ième meilleur chemin est connue (après $3M + f(2M)$ comparaisons), on peut choisir les M meilleurs chemins avec au plus 2M comparaisons. Le nombre total de comparaisons est donc approximativement de $3M + 2M = 5M$, ignorant f(2M).

En comparaison, l'AV demande 2^{K-1} comparaisons binaires pour trouver le meilleur des deux chemins convergeant à un des 2^{K-1} noeuds. De plus, $2(2^{K-1})$ métriques de branches doivent être calculées et $2(2^{K-1})$ additions doivent être faites pour obtenir les métriques des chemins. L'AM nécessite seulement le calcul de 2M distances et 2M additions. On voit que si M est plus petit que 2^{K-1}, la complexité peut être considérablement réduite.

Performance

Cet algorithme a été réalisé pour décoder des codes convolutionnels. Les exemples suivants montrent que la performance d'erreur par bit est moins bonne que celle de l'algorithme de Viterbi. Le nombre d'événements-erreurs est beaucoup plus petit pour l'algorithme M: par exemple, à $E_b/N_0 = 4.61$ dB, pour K=6, l'AV a 197 événements-erreurs alors que pour le même nombre de survivants (32), l'AM a 91 événements pour K=8 et seulement 68, si K=9. Par contre, si nous regardons le nombre de bits en erreur, la performance de l'AM n'est pas du tout satisfaisante

aux rapports de signal-à-bruit considérés. Si l'AM est utilisé, le nombre de bits incorrects de chaque événement est très grand si le chemin correct ne fait pas partie des survivants: il faut beaucoup de temps à l'AM pour se resynchroniser (pour retrouver le chemin correct), alors qu'avec l'AV, l'état correct n'est jamais perdu et le nombre de bits incorrects d'un événement est limité.

		Viterbi	Algorithme M	
		K=6	K=8	K=9
	N.even.err.	197	91	68
32 survivants	P(E)	3.94×10^{-4}	1.82×10^{-4}	1.36×10^{-4}
500000 bits	N.bits incor.	1054	5183	15109
	P(B)	2.11×10^{-3}	1.04×10^{-2}	3.02×10^{-2}
	N.even.err.	223	117	90
64 survivants	P(E)	2.23×10^{-4}	1.17×10^{-4}	9.0×10^{-5}
10^6 bits	N.bits incor.	1229	1524	5720
	P(B)	1.23×10^{-3}	1.52×10^{-3}	5.72×10^{-3}

Tableau 3.1: Performance d'erreur de l'Algorithme M ($E_b/N_0 = 4.61dB$, R = 1/2)

		Viterbi	Algorithme M	
		K=6	K=8	K=9
	N.even.err.	17	9	4
32 survivants	P(E)	1.7×10^{-5}	9.0×10^{-6}	4.0×10^{-6}
10^6 bits	N.bits incor.	69	1039	1784
	P(B)	6.9×10^{-5}	1.04×10^{-3}	1.78×10^{-3}
	N.even.err.	7	3	1
64 survivants	P(E)	7.0×10^{-6}	3.0×10^{-6}	1.0×10^{-6}
10^6 bits	N.bits incor.	30	8	259
	P(B)	3.0×10^{-5}	8.0×10^{-6}	2.59×10^{-4}

Tableau 3.2: Performance d'erreur de l'Algorithme M ($E_b/N_0 = 6.0dB$, R = 1/2)

D'après le tableau 3.2, il semble que pour M=64 et Eb/No=6.0 dB, on obtient un meilleur résultat que l'algorithme de Viterbi avec l'AM et K=8. Mais, si nous répétons la simulation pour un nombre assez grand de bits d'information, 2.5×10^6 bits, nous voyons que ce n'est pas vrai: nous avons obtenu 102 bits en erreur et 5 événements-erreurs. L'un de ces événements a causé la perte du chemin correct et le décodage s'est effectué de façon aléatoire à partir du bit d'information 2 438 594 jusqu'au bit 2 438 728, provoquant 93 bits incorrects. Le même nombre de bits décodés avec l'AV en utilisant un code ayant K=7, a donné 100 bits en erreur et 23 événements-erreurs. Ceci confirme que le nombre d'événements-erreurs est plus petit en utilisant l'AM et un code ayant une plus grande longueur de contrainte mais un événement de l'AM peut causer un nombre très élevé de bits incorrects.

En conclusion, si le rapport signal-à-bruit n'est pas très élevé, la probabilité de perdre le chemin correct, c'est-à-dire de ne pas avoir le chemin correct parmi les M survivants, est grande et conduit à de nombreux bits en erreur. Cet algorithme ne peut donc pas remplacer l'AV: pour le même nombre de survivants, la probabilité d'erreur par bit de l'AV est plus faible.

Les résultats que nous avons obtenus concordent avec les travaux de Lin [9] et Aulin [10], qui ont eux aussi rencontré les problèmes causés par la perte du chemin correct. Bien que toute variabilité de l'effort de calcul soit éliminée, l'algorithme M ne donne donc pas une bonne probabilité d'erreur par bit car dès que le chemin correct est perdu, il en résulte un grand nombre de bits incorrects. Par conséquent, les algorithmes traités dans ce chapitre ne sont pas entièrement satisfaisants même s'ils présentent certains avantages sur l'algorithme de Viterbi et le décodage séquentiel. Nous devons développer un nouvel algorithme, qui est décrit dans le chapitre suivant.

Chapitre 4

Algorithme de Viterbi adaptatif

Trois nouveaux algorithmes, les algorithmes à chemins multiples, à piles multiples et l'algorithme M, ont été décrits dans le chapitre précédent. Ils présentent des améliorations par rapport aux méthodes "classiques" (décodage de Viterbi et décodage séquentiel) mais aussi des inconvénients qui nous obligent à développer un nouvel algorithme.

4.1 Objectifs

Pour ce nouvel algorithme, nous voulons garder les avantages des méthodes connues décrites précédemment et éviter les inconvénients. Par conséquent, nous aimerions avoir les propriétés suivantes:

(a) Pour une complexité donnée, la performance d'erreur doit être meilleure que celle de l'AV (en utilisant un code ayant un plus grand pouvoir de correction) même si le rapport signal-à-bruit n'est pas très élevé.

(b) L'effort de calcul pour décoder un bit doit être inférieur à celui de l'algorithme de Viterbi et ne pas croître exponentiellement avec la longueur de contrainte K.

(c) La variabilité de l'effort de calcul doit être moindre que celle du décodage séquentiel.

(d) La mémoire requise ne doit pas être considérable.

(e) L'utilisation de blocs ne doit pas être une condition nécessaire au bon fonctionnement de l'algorithme.

Propriété (a) et (b): Il est évident que si ces propriétés ne sont pas réalisées, il n'y aura aucun avantage à utiliser cet algorithme à la place de l'AV. De plus, nous savons que les algorithmes de décodage séquentiel, tel l'algorithme de Zigangirov-Jelinek, sont asymptotiquement optimaux; cela signifie que leur probabilité d'erreur s'approche de celle d'un algorithme optimal si le rapport signal-à-bruit est très élevé. Cependant, nous voulons obtenir une performance quasi optimale même si le canal n'est pas très bon (pour R/R_{comp} compris entre 0.8 et 1).

Propriété (c): Il n'est pas toujours désirable d'éliminer complètement la variabilité de l'effort de calcul: pour des raisons d'efficacité (pour avoir un effort de calcul plus faible en moyenne), l'algorithme doit être adaptatif. En effet, le bruit dans le canal est aléatoire et affecte seulement une faible proportion des symboles reçus. Par exemple, si le taux de codage est R=1/2 et le rapport signal-à-bruit $E_b/N_0 = 4.61$dB, ce qui correspond à $R/R_{comp} = 0.99$, 4.4×2 % des symboles sont affectés. Le décodeur de Viterbi, qui fait un nombre constant de calculs pour décoder un bit, est obligé de prévoir le pire cas: il examine et garde à chaque fois tous les 2^{K-1} noeuds. Il est évident qu'un seul d'entre eux est correct. Nous avons vu que seule une petite proportion des symboles reçus a subi une transition et donc, la plupart du temps, le canal n'est pas bruité. Mais, s'il n'y a pas de transitions dans le canal, il est facile de reconnaître le noeud correct. Examiner les autres noeuds ne sert à rien. En évitant de le faire, les ressources disponibles sont utilisées beaucoup plus efficacement: on n'effectue pas de calculs inutiles et ainsi, on élimine

le gaspillage énorme de l'AV.

Toutefois, il ne faut pas tomber dans l'autre extrême, c'est-à-dire, faire trop peu de calculs. Par exemple, en explorant seulement un chemin à la fois, l'AZJ fonctionne très bien si le canal est bon. Mais dès que les symboles reçus ont subi des transitions, les retours en arrière peuvent provoquer des débordements et effacements catastrophiques.

Même si le nombre d'opérations est compris entre ces deux extrêmes, c'est-à-dire $1 \leq M \leq 2^{K-1}$ et M constant (dans le cas des nouveaux algorithmes décrits dans le chapitre précédent), le problème reste le même: M est soit trop grand, quand il n'y a pas de bruit (la majeure partie du temps), soit insuffisant, quand le bruit va affecter le canal.

Il est clair que le décodeur doit s'adapter au bruit du canal car, comme on vient de le voir, faire un nombre constant d'opérations n'est jamais satisfaisant: quelquefois, c'est insuffisant, d'autres fois, c'est trop. L'effort de calcul doit donc être variable et un tampon est nécessaire à l'entrée du décodeur pour conserver les symboles qui n'ont pas encore pu être traités. Le décodeur doit avoir un gain de vitesse G (nombre de calculs/temps d'interarrivée) supérieur à l'effort de calcul moyen car sinon, le tampon va déborder rapidement. Il est important que la file d'attente dans le tampon soit stable, sinon on retomberait dans le problème du décodage séquentiel.

Propriétés (d) et (e): Elles visent surtout à éliminer les inconvénients de l'algorithme à M piles (AMP). La mémoire requise doit être inférieure à la mémoire nécessaire à l'AMP. De plus, l'utilisation de petits blocs (500 à 750 bits) diminue l'efficacité de transmission. Par exemple, si K=24 et des blocs de 500 bits d'information sont utilisés, l'ajout d'une queue de (K-1) bits 0 à la fin du bloc pour reinitialiser le registre équivaut à une diminution de l'efficacité de plus de 4 %. L'utilisation

de blocs doit être un choix de l'utilisateur et non une contrainte imposée par le décodeur pour déterminer la séquence décodée avant le débordement de la pile ou encore pour resynchroniser le système (en cas de débordement ou perte du chemin correct).

En résumé, nous voulons obtenir la performance de l'AV mais à un effort de décodage faible en moyenne, comme avec le décodage séquentiel. Cependant, cet effort doit être moins variable car nous ne voulons pas les débordements du décodage séquentiel. Haccoun [6] a montré que l'AV et l'AZJ peuvent être considérés comme les deux cas particuliers extrêmes d'un algorithme généralisé à pile (voir section 3. 1). L'algorithme que nous allons développer se situe quelque part entre ces deux extrêmes et il est évident qu'il peut aussi être considéré comme un algorithme généralisé à pile.

4.2 Description des principes de base

Après avoir indiqué les propriétés désirées, nous allons maintenant décrire les principes de cet algorithme. Tel que mentionné dans la section 4.1, on veut utiliser un code ayant une longueur de contrainte plus élevée que celle utilisée en pratique par les décodeurs de Viterbi, sans augmenter la complexité. Il faut donc réduire le nombre de survivants à chaque niveau du treillis.

L'algorithme proposé peut être considéré comme l'algorithme de Viterbi à recherche non exhaustive. A chaque niveau du treillis, les chemins qui sont à une distance (de Hamming) supérieure à un seuil dénoté S de la séquence reçue, sont éliminés car il est très peu probable qu'ils correspondent au chemin correct. Comme avec l'AV, la métrique utilisée est la distance de Hamming: tous les chemins ayant la même longueur, il est inutile de biaiser la métrique contrairement à ce qu'on

faisait avec le décodage séquentiel. Donc, un chemin est gardé et devient l'un des survivants si sa distance est inférieure ou égale à MIN + S, MIN étant la distance au niveau précédent du treillis entre le meilleur chemin et la séquence reçue et S, le seuil de rejet choisi par l'utilisateur.

Plusieurs cases sont utilisées pour trier les survivants: les deux successeurs d'un noeud sont placés dans la case correspondant à leur distance. Ainsi, la première case contient tous les noeuds ou chemins qui sont à une distance inférieure ou égale à MIN + P1 de la séquence reçue, la deuxième case, les noeuds ayant une distance inférieure ou égale à MIN + P2, et ainsi de suite. P1, P2, $\cdots$, Pn (n étant le nombre total de cases utilisés) sont des paramètres choisis par l'usager. En général, n = S (le seuil de rejet), P1 = 1, P2 = 2, etc. L'influence de n et des paramètres Pi, i=1, $\cdots$, n, sur la performance est considérée dans la section 4.5.

Quand le triage est fini, c'est-à-dire quand tous les noeuds sont entrés dans les différentes cases, on garde tous les noeuds qui sont dans les cases dont la distance est inférieure ou égale à MIN + S en commençant par la première case. Les noeuds survivants sont ensuite comparés pour vérifier s'il y a eu convergence. En cas de convergence, on élimine le noeud dont la métrique est moins bonne.

Les noeuds sont pris d'après la règle du premier arrivé, premier servi (First In, First Out ou FIFO). Le survivant numéro 1 est donc celui qui est entré le premier dans la meilleure case non vide. Ce survivant, appelé le meilleur, est traité en premier au niveau suivant du treillis: ses deux successeurs vont être les premiers à entrer dans les cases et même si d'autres noeuds ont la même métrique et convergent au même noeud, le survivant # 1 étant le premier entré, le successeur du meilleur chemin au niveau précédent est considéré le meilleur. Le meilleur chemin au niveau précédent est donc favorisé par rapport aux autres. Ceci est désirable car s'il n'y a pas eu de transition dans le canal, le meilleur chemin doit être une prolongation du

meilleur au niveau précédent.

Si le canal est bon (les symboles reçus sont corrects) et nous avons déjà dit qu'il l'est la plupart du temps, cet algorithme possède un avantage intrinsèque sur l'AV. En effet, si deux chemins convergent à un noeud avec la même métrique ("tie" ou égalité), l'AV choisit le survivant au hasard. La probabilité de se tromper est de 50 % (voir chap. 2). Par conséquent, si un chemin incorrect a la même métrique que le chemin correct et les deux chemins convergent au même noeud, on a 50 % de chance de rejeter le chemin correct. Il n'y a aucun moyen de corriger cette situation car, comme nous l'avons déjà vu, les deux chemins vont être affectés exactement de la même façon par la suite et les métriques seront toujours égales. Il est donc inutile de garder les deux car rien ne permettrait de prendre une meilleure décision plus tard. Les égalités arrivent assez fréquemment en quantification dure: 1/4 à 1/3 des noeuds sont touchés. Heureusement, le chemin correct est affecté plus rarement car sinon, la performance serait catastrophique.

De plus, l'AV choisit aussi le meilleur noeud à un niveau du treillis au hasard si deux noeuds ont la même métrique. C'est le prédecesseur de ce noeud, L niveaux plus tôt (L étant la longueur de l'historique du chemin) qui est délivré à l'usager (bit décodé). Ce problème est cependant moins grave que les égalités car dans la plupart des cas, les noeuds survivants proviennent du même noeud L niveaux plus tôt, si L est suffisamment grand (voir chap. 2). De plus, il est beaucoup moins fréquent que les égalités.

D'autres algorithmes visent le même but que l'AVA, c'est-à-dire réduire la complexité de l'AV pour pouvoir utiliser un code ayant une longueur de contrainte plus élevée et ainsi, obtenir une probabilité d'erreur plus faible. Une liste non-exhaustive de ces algorithmes comprend l'algorithme M [9, 10, 17], qui a été décrit au chapitre 3, et l'algorithme de Viterbi réduit [22]. Pour cet algorithme, le nombre d'états est

réduit en créant des "pseudo-états", qui regroupent plusieurs états "réels". Toutefois, la propagation d'erreurs peut devenir catastrophique, ce que nous voulons éviter avec l'AVA.

En conclusion, l'algorithme proposé est intrinsèquement supérieur à l'AV si le canal n'est pas affecté par du bruit parce que le meilleur chemin au niveau précédent, qui a la plus grande probabilité d'être le chemin correct, est favorisé. Toutefois, cet avantage est juste considéré comme un boni et ce n'est pas sur cela que nous comptons pour obtenir une meilleure probabilité d'erreur que l'AV à un effort de calcul moindre en moyenne. L'amélioration provient plutôt de l'utilisation d'un code plus puissant, c'est-à-dire, ayant une plus grande longueur de contrainte.

4.3 Etapes du décodage

Les principes de cet algorithme adaptatif ont été exposés dans la section précédente. Les opérations de décodage sont maintenant décrites à l'aide de l'organigramme de la figure 4.1.

Nous supposons que le nombre de cases n est égal à S, le seuil et que

$$P1 = 1, P2 = 2, \cdots, Pn = n = S$$

Soient L, la longueur de l'historique du chemin, i, le numéro de la case qui contient les chemins, dénotés par le noeud extrême, dont la métrique est inférieure ou égale à MIN + Pi et N_{max}, le nombre maximal de survivants.

Au début, comme avec l'AV, le noeud d'origine est le noeud survivant #1 et on lui assigne une métrique ou distance de Hamming 0. Toutefois, contrairement à l'AV – qui a 2^{K-1} survivants à tous les niveaux du treillis –, il n'est pas nécessaire d'avoir d'autres survivants ici.

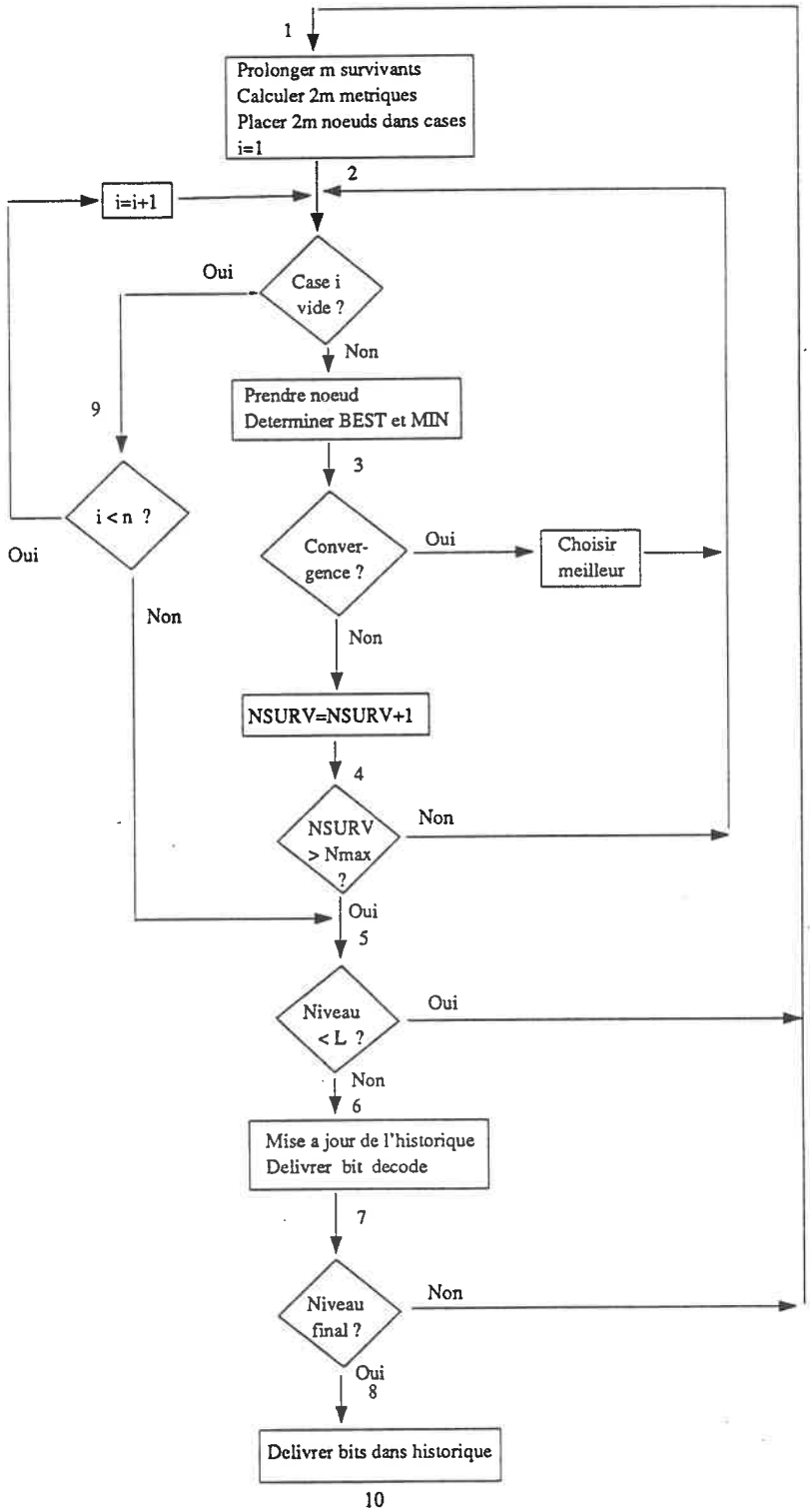

Figure 4.1: Organigramme de l'Algorithme de Viterbi Adaptatif

1. Prolonger les noeuds survivants du niveau précédent. Soit m, le nombre de ces noeuds (au début, m=1, bien sûr). On obtient donc 2m successeurs et 2m métriques. Ces 2m noeuds sont placés dans des cases selon leur métrique, en suivant les principes de triage établis dans la section précédente. Donner à i la valeur 1 (on va commencer par la case 1).

2. Si la case i n'est pas vide, prendre tous les noeuds de cette case en commençant par celui qui a été entré le premier et déterminer le meilleur noeud (noté BEST) à ce niveau du treillis et sa métrique MIN. Si la case i est vide, passer à 9.

3. Vérifier s'il y a convergence (deux noeuds identiques). Si oui, garder le meilleur, éliminer l'autre et aller à 2. Sinon, augmenter le nombre de survivants NSURV de 1. Le nombre de survivants n'est donc pas augmenté en cas de convergence: ceci permet d'éviter qu'un des deux chemins convergents, qui ne pourra jamais être le meilleur (voir explication dans la section de l'AV), prenne la place d'un autre chemin, qui risque d'être le chemin correct, le nombre de survivants étant limité à N_{max}.

4. Si le nombre de survivants est inférieur à N_{max}, aller à 2.

5. Si le niveau du treillis est inférieur à L, aller à 1.

6. Mettre l'historique du chemin à jour. Délivrer le noeud précédant BEST, L niveaux plus tôt (ce noeud permet de déterminer le bit d'information transmis).

7. Si le niveau final du treillis n'est pas atteint, aller à 1.

8. Délivrer les noeuds restant dans l'historique. Aller à 10.

9. Si i est supérieur ou égal à n, aller à 5. Si inférieur, augmenter i de 1 et aller à 2.

10. Fin.

Il faut noter que les étapes 5 à 8 sont identiques à celles que l'AV doit accomplir. La seule différence entre l'AV et l'AVA réside dans la sélection des survivants (étape

2 à 4 et 9).

Comme exemple, à la figure 4.2, la séquence reçue utilisée dans l'exemple de décodage de l'AV et de l'AZJ

$$Y = (01, 10, 00, 01, 00, 11)$$

est décodée en utilisant cet algorithme. Le seuil de rejet et le nombre de cases utilisé est 1, c'est-à-dire, un noeud est rejeté si sa distance est supérieure à MIN + 1, où MIN est la distance minimale au niveau précédent. Ainsi, au niveau 2 du treillis, le quatrième noeud (correspondant à l'état 11) est rejeté car sa distance est 3 et MIN au niveau précédent est 1 ($3 >$ MIN + 1). Tel que prévu, si le canal est bon, le nombre de survivants de l'AVA est inférieur à celui de l'AV ($2^{K-1} = 4$): au niveau 3, il y a deux survivants et au niveau 4, un seul. Le chemin décodé est le même que celui obtenu par l'AV.

4.4 Adaptation de l'effort de calcul et stabilité

Nous allons montrer dans cette section comment l'AVA s'adapte au canal.

S'il n'y a pas de bruit dans le canal, seul un petit nombre N_{min} de chemins se trouvent à une distance inférieure ou égale à MIN + S. Ce nombre est constant. Les distances des chemins incorrects augmentent rapidement et ces chemins sont éliminés. Le nombre de survivants est donc N_{min} et l'effort de décodage est petit. Cependant, si les symboles reçus ont subi une transition à cause du bruit dans le canal, la distance entre le chemin correct et la séquence reçue augmente, tandis que la distance de certains chemins incorrects peut ne pas augmenter. Nous supposons que le chemin correct n'est pas perdu, c'est-à-dire qu'il fait toujours partie des survivants. Tous les chemins ont donc approximativement la même métrique: il y

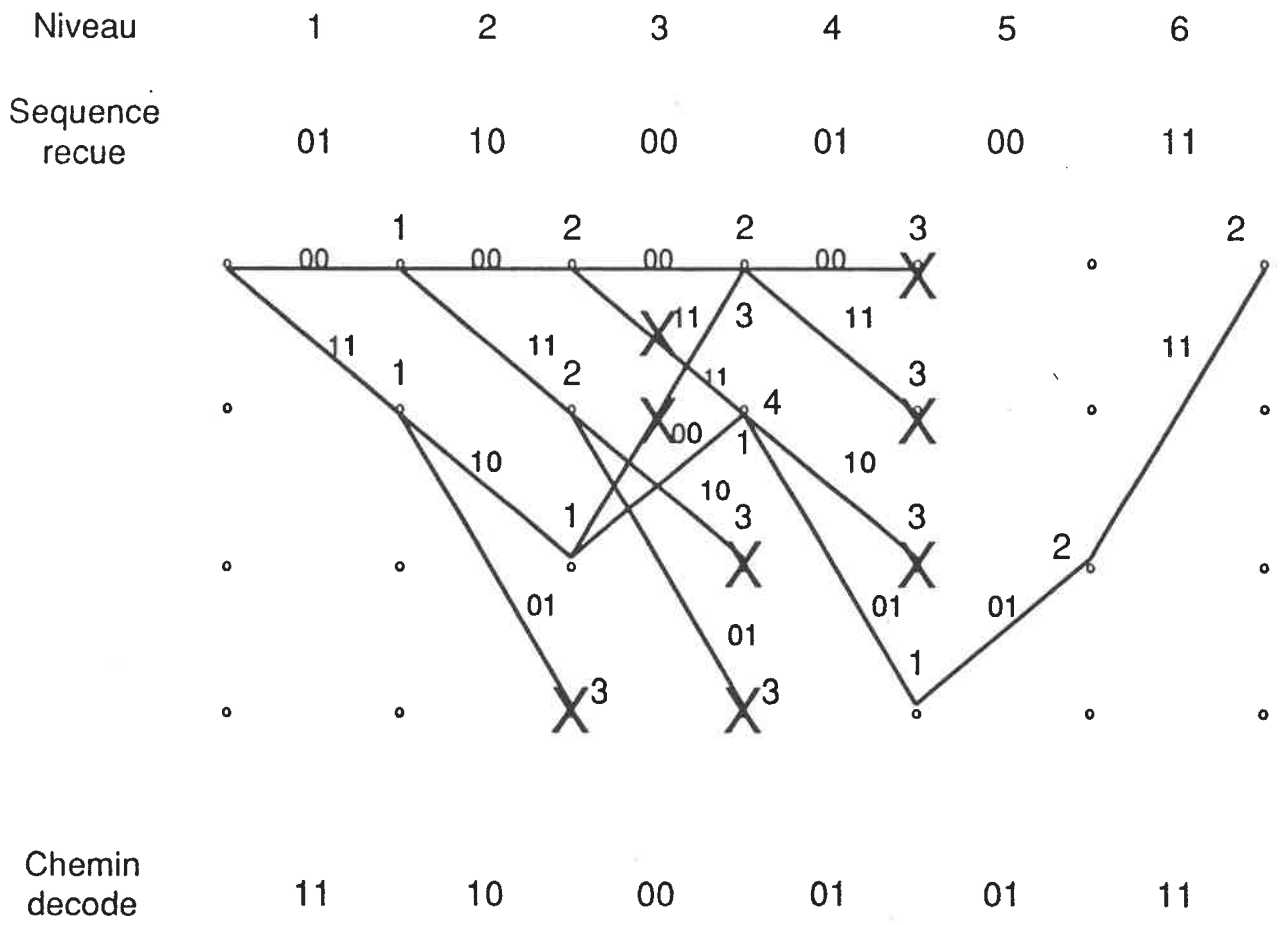

Figure 4.2: Exemple de décodage par l'Algorithme de Viterbi Adaptatif (Seuil de rejet=1)

a beaucoup de chemins à une distance inférieure ou égale à MIN + S et le nombre de survivants devient élevé. Dans ce cas, l'effort de décodage est plus grand.

Quand le bruit dans le canal cesse, la distance du chemin correct n'augmente plus, contrairement à celle des chemins incorrects. Ces derniers vont être éliminés et graduellement, le nombre de survivants redescend à N_{min}. L'avantage de cet algorithme, qui s'adapte au canal, est clair: la quantité de travail fourni dépend de la qualité du canal. Cela nous permet d'utiliser un code ayant une plus grande longueur de contrainte parce que même si l'effort de calcul est momentanément élevé, il diminue dès que la salve de bruit se termine. Il n'est donc pas nécessaire d'avoir un décodeur pouvant accomplir constamment l'effort de calcul maximal; nous pouvons nous contenter d'un décodeur moins complexe, faisant moins de calculs que le maximum requis pendant les salves de bruit, mais capable d'accomplir assez rapidement tous les calculs non faits quand le canal redevient bon et l'effort de calcul, faible. Comme on va le voir au chapitre suivant, ce caractère adaptatif du décodeur lui permet de donner une bonne performance, proche de celle d'un décodeur de Viterbi, à un effort de calcul moindre en moyenne.

L'avantage de l'adaptation de l'effort de calcul au bruit vient d'être décrit plus haut (voir également section 4.1, prop. (c)). Malheureusement, la variabilité de cet effort entraîne aussi des inconvénients, qui sont maintenant considérés.

Le nombre d'opérations à effectuer pour décoder un bit varie à chaque niveau du treillis et nous devons décider quel doit être le gain de vitesse ou la capacité de traitement de ce décodeur, c'est-à-dire, quel est le nombre d'opérations que le décodeur peut réaliser pendant le temps d'interarrivée de deux bits d'information. Supposons que le décodeur puisse accomplir G opérations pendant le temps d'interarrivée de deux bits, une opération consistant à prolonger un chemin, calculer les

deux métriques des successeurs et placer ces successeurs dans les cases.

Si G est égal à N_{max}, le nombre maximal de survivants, un tampon n'est pas nécessaire à l'entrée du décodeur, mais le décodeur est inactif la plupart du temps. Cela équivaut à un gaspillage et ne correspond pas à l'objectif d'efficacité visé (voir prop. (c)). Par contre, si G est inférieur ou égal au nombre moyen de survivants, cela équivaut à avoir un taux d'arrivée supérieur au taux de service ou taux de sortie. Il est évident que ceci est inacceptable car la file d'attente dans le tampon d'entrée va tendre vers l'infini. G doit donc être entre le nombre moyen de survivants et N_{max}.

Dans ce cas, s'il y a du bruit, le nombre d'opérations à effectuer est supérieur à G. Le décodeur va accumuler un certain retard: il n'a pas le temps de traiter les bits qui arrivent et ceux-ci sont stockés dans le tampon d'entrée en attendant. Dès que le bruit cesse (toujours en supposant que le chemin correct n'est pas perdu), le nombre d'opérations requises diminue graduellement jusqu'à N_{min}. Comme G est supérieur à N_{min}, le décodeur peut traiter plusieurs bits (G/N_{min} bits, plus exactement) avant qu'un autre bit arrive. Si G/N_{min} est suffisamment grand, il peut donc vider le tampon d'entrée et éviter tout débordement. Nous allons considérer le cas où le chemin correct est perdu dans la section 4.5, quand l'effet de N_{max} sur la performance est étudié. Ainsi, il y a un échange possible entre le gain de vitesse et la taille du tampon.

Pour qu'une file d'attente soit stable, il faut que le facteur d'utilisation ρ, qui est le produit du temps moyen de service par le taux d'arrivée, soit inférieur à 1 [18]. Le taux d'arrivée λ est constant: nous supposons qu'une branche arrive à chaque unité de temps, où une unité de temps est le temps d'interarrivée de deux branches. Donc, $\lambda = 1$. Le temps de service t est égal à n_s/G, où n_s est le nombre dynamique de survivants. Soit $P(t)$ la probabilité que le temps de service soit égal à t. Supposons que nous réalisons une simulation en utilisant n bits d'information et que nous

obtenons $x^{(n_s)}$ fois un certain nombre de survivants n_s. Alors, $P(t) = x^{(n_s)}/n$ et $t = n_s/G$. Comme n_s peut varier de 1 à N_{max}, ρ est donné par la relation suivante:

$$\rho = \lambda \bar{t} = \lambda \sum_{t} tP(t)$$

D'où

$$\rho = \lambda \sum_{n_s=1}^{N_{max}} \frac{n_s}{G} \frac{x^{(n_s)}}{n} \tag{4.1}$$

Pour que ρ soit inférieure à 1, G doit être suffisamment grand. Les résultats de trois simulations réalisées avec un rapport signal-à-bruit égal à 4.61 dB, 5.5 dB et 6.0 dB sont donnés à l'Annexe B. Le gain de vitesse G du décodeur est égal à 64 et la longueur de contrainte du code est K=8. Le nombre de survivants n_s varie de 1 à $2^{K-1} = 128$, tel qu'indiqué à l'Annexe B. On compte combien de fois n_s est apparu pendant le décodage des 2×10^6 bits d'information et le total $x^{(n_s)}$ est utilisé pour calculer ρ à l'aide de l'équation 4.1. Les valeurs obtenues pour ρ sont reportées dans le tableau 4.1.

	E_b/N_0		
	4.61 dB	5.5 dB	6.0 dB
ρ	0.571	0.406	0.343

Tableau 4.1: Valeur de ρ pour un gain de vitesse G=64 et un code ayant K=8

On peut constater que le facteur d'utilisation ρ diminue quand le rapport de signal-à-bruit augmente. Ceci est normal car le décodeur étant adaptatif, il accomplit moins de travail lorsque le canal est bon. Dans les exemples du tableau 4.1, où le nombre de survivants a été obtenu par simulation, ρ est inférieur à 1. On peut en conclure que la valeur de G est satisfaisante et que la file d'attente est stable. Par conséquent, la taille maximale de la file d'attente devrait être modeste. Ceci est vérifié à la section 5.3.

	Longueur de l'historique(bits)	Nombre de bits en erreur	Probabilité d'erreur
	14	498	4.98×10^{-4}
	21	116	1.16×10^{-4}
	28	80	8.0×10^{-5}
	35	82	8.2×10^{-5}
E_b/N_0=6.0 dB	42	82	8.2×10^{-5}
1000000 bits	56	82	8.2×10^{-5}
	70	82	8.2×10^{-5}
	98	82	8.2×10^{-5}
	200	82	8.2×10^{-5}
	28	407	2.03×10^{-3}
E_b/N_0=4.61 dB	42	330	1.65×10^{-3}
200000 bits	200	320	1.60×10^{-3}

Tableau 4.2: Influence de la longueur de l'historique sur la performance pour un code ayant K=7

4.5 Influence des paramètres

Nous allons considérer l'influence sur la performance des paramètres choisis par l'utilisateur: la longueur de l'historique du chemin, le nombre maximal de survivants et le nombre de cases. La valeur à donner à ces paramètres pour obtenir la meilleure performance possible est également déterminée.

4.5.1 Longueur de l'historique du chemin

Si le chemin correct fait partie des survivants, il a exactement la même métrique en utilisant l'AV ou l'AVA. La longueur de l'historique du chemin ou longueur de récupération peut donc être choisie égale à celle de l'AV. Tel qu'indiqué au chapitre 2, Heller et Jacobs [12] ont observé que si la longueur de l'historique est supérieure à quatre ou cinq fois la longueur de contrainte K, la dégradation de la

	Longueur de l'historique(bits)	Nombre de bits en erreur	Probabilité d'erreur
E_b/N_0=6.0 dB	48	56	5.6×10^{-5}
1000000 bits	200	52	5.2×10^{-5}
E_b/N_0=5.0 dB	48	102	3.4×10^{-4}
300000 bits	200	102	3.4×10^{-4}
E_b/N_0=4.61 dB	48	177	8.85×10^{-4}
200000 bits	200	180	9.0×10^{-4}

Tableau 4.3: Influence de la longueur de l'historique sur la performance pour un code ayant K=8

performance de l'algorithme de Viterbi tronqué est négligeable par rapport à celle de l'algorithme non tronqué.

Les tableaux 4.2 et 4.3 montrent qu'une longueur de récupération M égale à six fois la longueur de contrainte K est suffisante pour les rapports de signal-à-bruit qui nous intéressent (entre 4.61 et 6.0 dB) et que la performance n'est pas améliorée même si M augmente. Dans toutes les simulations qui vont être faites par la suite, la longueur de l'historique du chemin est choisie égale à six fois K.

4.5.2 Nombre maximal de survivants

Pour avoir la meilleure performance d'erreur possible, le nombre maximal de survivants N_{max} devrait être égal à celui de l'AV, 2^{K-1}. Cependant, si K est grand, 2^{K-1} peut représenter un nombre trop grand: la mémoire requise et l'effort de calcul à fournir deviennent irréalisables matériellement. La grande variabilité de l'effort de calcul (le nombre de survivants varie entre N_{min} et 2^{K-1}) peut entrainer des débordements dans le tampon d'entrée, sauf si le gain de vitesse G est beaucoup plus grand que N_{min}, c'est-à-dire, si le décodeur est très puissant (voir section 4.4).

Un décodeur, aussi puissant et complexe, est cependant difficile, sinon impossible à réaliser.

Pour éviter ces inconvénients, le nombre de survivants doit donc être limité si K est grand. Malheureusement, cela entraine d'autres inconvénients. Si N_{max} est inférieur à 2^{K-1} et si N_{max} chemins ont une meilleure métrique que le chemin correct, il se peut que le chemin correct soit rejeté . De plus, si N_{max} est beaucoup plus petit que 2^{K-1} et le chemin correct est perdu, le décodeur va examiner les chemins possibles de façon aléatoire: il va toujours prolonger les meilleurs chemins, mais comme ils ont tous sensiblement la même métrique, aucun ne va se distinguer et l'opération de décodage équivaut à choisir les survivants au hasard (décodage aléatoire). Seul le bruit, donc le hasard, peut lui faire retrouver le chemin correct. Cette situation est à éviter.

Par contre, si N_{max} n'est pas limité et le chemin correct est perdu, il redeviendra rapidement l'un des survivants et il sera le meilleur peu après la fin des transitions dans le canal. En effet, si le chemin correct est perdu, tous les chemins ont à peu près la même métrique (voir section 4.4). Le nombre de survivants augmente donc à chaque extension (double s'il n'y a pas de convergence). Par conséquent, on obtient 2^{K-1} survivants après quelques branches (K-1 branches dans le pire cas) et forcément, l'un d'eux est le chemin correct. S'il n'y a plus de bruit dans le canal, la métrique de ce chemin devient meilleure que celles des autres, ou plutôt, les métriques des chemins incorrects vont se détériorer tandis que celle du chemin correct ne bouge pas. Par conséquent, si N_{max} est égal à 2^{K-1}, cet algorithme peut retrouver tout seul le chemin correct perdu: on n'a pas besoin de diviser l'information en petits blocs car, contrairement aux algorithmes de décodage séquentiel, tel l'AMC ou l'AMP, l'AVA peut se resynchroniser. Ainsi, même si le chemin correct est perdu, le décodage ne devient pas aléatoire jusqu'à la fin de

la séquence d'information, mais seulement jusqu'à ce que le chemin correct soit retrouvé. Le nombre de bits en erreur est donc limité à la période de resynchronisation.

La probabilité de resynchronisation, notée P_{resync}, est donnée par:

$$P_{resync} = \frac{N_{max}}{2^{K-1}} \tag{4.2}$$

Démonstration

Tel qu'indiqué plus haut, quand le chemin correct est perdu, le décodeur choisit les survivants au hasard. P_{resync}, la probabilité de choisir le chemin correct, est

$$P_{resync} = \frac{\binom{2^{K-1}-1}{N_{max}-1}}{\binom{2^{K-1}}{N_{max}}} \tag{4.3}$$

Le dénominateur est le nombre total de combinaisons de 2^{K-1} éléments quand on en prend N_{max} au hasard. Le numérateur est le nombre de combinaisons contenant le chemin correct: on prend le chemin correct et ensuite, on choisit les autres $N_{max}-1$ au hasard parmi les $2^{K-1}-1$ chemins restant. Le dénominateur peut s'écrire:

$$\begin{aligned} \binom{2^{K-1}}{N_{max}} &= \frac{(2^{K-1})!}{N_{max}!(2^{K-1}-N_{max})!} && (4.4) \\ &= \frac{2^{K-1}(2^{K-1}-1)!}{N_{max}(N_{max}-1)![(2^{K-1}-1)-(N_{max}-1)]!} \\ &= \frac{2^{K-1}}{N_{max}}\binom{2^{K-1}-1}{N_{max}-1} && (4.5) \end{aligned}$$

En substituant l'équation 4.5 dans 4.3, on obtient l'équation 4.2.

La probabilité de retrouver le chemin correct est très faible si N_{max} est beaucoup plus petit que 2^{K-1} et c'est ce qui passe avec l'algorithme M (voir chapitre 3): une fois que le chemin correct est perdu, il est presque impossible de le retrouver et le nombre de bits en erreur est évidemment très grand. Par contre, l'AVA est capable de se resynchroniser, sans utiliser des blocs. Cette resynchronisation est garantie

seulement si N_{max} est égal à 2^{K-1}. Evidemment, le nombre de cases et le seuil de rejet ne doivent pas être trop petits. L'influence du nombre de cases et la valeur du seuil à choisir pour avoir la meilleure performance possible sont étudiées dans la section suivante.

	Eb/No=4.61dB, 4 cases, 10^6 bits			
	K=9			K=8
Nmax	256	200	128	128
N. bits err.	413	459	1041	669
N.evenem.erreur	66	67	68	104
N.surv. moy.	39.24	39.00	36.98	36.5
Queue max.	218	186	367	77
tamp. d'ent. moy.	7.4	6.7	3.7	2.2

Tableau 4.4: Influence de N_{max} sur la probabilité d'erreur

Le tableau 4.4 montre l'influence de N_{max} sur la probabilité d'erreur. On voit que pour K=9, le nombre d'événements-erreurs est sensiblement le même mais le nombre de bits erronés est très différent. La définition d'un événement-erreur a été donnée au chapitre 2. En divisant le nombre d'erreurs par le nombre d'événements, un événement a en moyenne 6.2 erreurs si $N_{max} = 256$ et 15.3 si $N_{max} = 128$. Toutefois, la moyenne ne donne pas vraiment une bonne indication de ce qui se passe: à une ou deux exceptions près, on a exactement les mêmes événements-erreurs mais certains événements ont beaucoup plus d'erreurs si N_{max} est plus petit. Par exemple, si $N_{max} = 128$, l'événement-erreur 26 a 170 bits en erreur, alors que le même événement a seulement 12 bits en erreur si $N_{max} = 256$. Cela confirme ce que nous avons dit précédemment: si le chemin correct est perdu et si $N_{max}/256$ est petit, c'est-à-dire, si la probabilité de resynchronisation est petite (0.5, par exemple), l'algorithme va mettre beaucoup de temps pour retrouver le chemin correct, d'où

le nombre élevé de bits en erreur.

Comme le nombre d'événements-erreurs est pratiquement le même, nous pouvons en déduire que la probabilité de perdre le chemin correct à cause de N_{max} est assez faible. Cependant, comme nous l'avons déjà dit, si le chemin correct est perdu, il en résulte de nombreux bits incorrects. Ainsi, pour K=9, si $N_{max} = 128$, l'événement-erreur 12 commençant au bit 161561 cause 59 bits erronés alors qu'il n'y en a aucun si $N_{max} = 256$. Si $N_{max} = 200$, nous avons aussi un événement-erreur, mais le nombre de bits en erreur est seulement de 24 au lieu de 59: la resynchronisation est plus rapide.

Le nombre moyen de survivants et le nombre moyen de bits dans le tampon d'entrée sont plus petits si N_{max} est limité, mais cela ne se traduit pas toujours par un tampon d'entrée plus petit: la queue maximale dans le tampon est de 367 bits d'information si $N_{max} = 128$ alors qu'elle est plus petite pour les deux autres cas. La procédure utilisée dans les simulations pour déterminer la taille de la file d'attente dans le tampon d'entrée est explicitée dans l'Annexe B. Pour éviter un débordement, il faut donc un tampon plus grand si $N_{max} = 128$. Donc, si N_{max} est plus grand, le nombre moyen de bits dans le tampon d'entrée est plus élevé, mais il est moins variable.

Nous remarquons aussi que si N_{max} est égal à 128, un code K=8 a une probabilité d'erreur par bit plus faible qu'un code K=9, même si le nombre d'événements-erreurs est plus grand. Chaque fois que le chemin correct est perdu, le nombre de bits en erreur est plus élevé avec le code K=9, qu'il ne l'est avec le code K=8. En effet, pour le premier code, la probabilité de resynchronisation est égale à 0.5 et pour le deuxième, avec K=8, elle est égale à 1. La période de resynchronisation et de décodage aléatoire est plus longue dans le premier cas et par conséquent, il y a plus de bits incorrects.

Utiliser un code plus puissant permet d'obtenir moins d'événements-erreurs, mais chaque événement contient plus de bits en erreur. Donc, si on veut minimiser la probabilité d'erreur *par bit* et si le nombre maximal de survivants est limité à N_{max}, il vaut mieux utiliser un code dont la longueur de contrainte satisfait $2^{K-1} \leq N_{max}$ plutôt qu'un code ayant un plus grand K. Par exemple, si $N_{max} = 128$, le code ayant K=8 va donner une probabilité d'erreur par bit plus faible que le code ayant K=9. C'est donc le code à choisir si le codage est utilisé pour corriger des erreurs.

Par contre, si on veut demander une retransmission en cas d'erreur (technique ARQ: Automatic Repeat Request) et si c'est la probabilité d'événement-erreur que l'on veut minimiser, il est préférable de choisir un code ayant une grande longueur de contrainte. Contrairement à l'algorithme de Viterbi, la détection d'erreur ne nécessite pas d'effort supplémentaire avec l'AVA: le nombre de survivants et leur métrique sont disponibles à chaque niveau du treillis et permet de déterminer la qualité du canal. Normalement, s'il n'y a pas de bruit dans le canal, la distance de Hamming d'un seul chemin, le chemin correct, ne change pas au niveau suivant . Ainsi, si aucun noeud ou plus d'un noeud ont une distance inchangée, cela veut dire qu'il y a eu une transition dans le canal. De plus, la perte du chemin correct est aussi facilement détectable: le nombre de survivants devient élevé et l'écart entre les métriques de ces survivants est faible. Dans ce cas, une retransmission du bloc d'information est demandée.

Dans cette section, nous avons étudié l'influence sur la performance d'une limitation du nombre maximal de survivants N_{max}. Nous avons vu que pour un nombre maximal de survivants donné, la probabilité d'événement-erreur décroît si on utilise un code plus puissant, mais la probabilité d'erreur par bit augmente si $2^{K-1} \geq N_{max}$. Nous allons maintenant considérer les autres paramètres.

4.5.3 Nombre de cases et seuil

Le seuil influence directement le nombre de survivants et la probabilité d'erreur. Il est évident que si S est grand, le nombre de survivants à chaque niveau du treillis est élevé, et par conséquent, la probabilité que le chemin correct soit parmi les survivants est plus forte. Cependant, un nombre plus grand de survivants entraine un temps de décodage plus long avec accumulation possible au tampon d'entrée. S est donc choisi le plus petit possible pour donner la probabilité d'erreur désirée.

Le nombre de cases influence aussi la performance. Il est préférable d'avoir n cases MIN + P1, MIN + P2, $\cdots$, MIN + n plutôt qu'une seule case MIN + S, même si S = n, c'est-à-dire, même si dans les deux cas, on rejette les noeuds dont la distance est supérieure à MIN + S. En effet, dans le premier cas, le meilleur chemin au niveau précédent est favorisé. De plus, si le nombre de survivants est limité à N_{max}, le premier cas garantit que les noeuds choisis sont les meilleurs puisqu'on commence par prendre les noeuds dans MIN + P1. Dans le deuxième cas, ce n'est pas certain. Pour cette raison, on choisira $Pj = Pi+1$, i,j=1,$\cdots$,n et on sera sûr que les noeuds que l'on a pris ont une distance inférieure ou au moins égale à celle des noeuds que l'on va prendre après. Seul le premier cas permet de réaliser l'avantage inhérent de l'AVA décrit dans la section 4.2 (le meilleur chemin au niveau précédent est favorisé en cas d'égalité).

En résumé, pour obtenir la meilleure performance possible si le seuil de rejet est S, nous allons utiliser S cases: MIN + 1, MIN + 2, $\cdots$, MIN + S. Ceci est confirmé par les résultats de simulation du tableau 4.5.

Ce tableau montre l'amélioration substantielle obtenue en utilisant seulement deux cases au lieu d'une seule. En choisissant P1=1, un des deux successeurs du meilleur noeud va entrer dans la case MIN + 1 et est favorisé par rapport aux

	500000 bits, E_b/N_0 = 6.0dB, S=4					
	K=8			K=9		
N.cases	1	2	4	1	2	4
Pi(i=1,...,n)	4	1,4	1,2,3,4	4	1,4	1,2,3,4
N.bits incor.	1448	13	8	380	0	0
N.even.err.	269	4	3	66	0	0
N.surv.moy.	22.06	22.07	21.98	21.04	21.04	20.98

Tableau 4.5: Influence du nombre de cases sur la probabilité d'erreur

autres. Le nombre de bits en erreur passe de 1448 à 13 pour le code ayant K=8. Si quatre cases sont utilisées, la performance est encore améliorée et le nombre de bits incorrects devient 8.

Nous avons déterminé que le nombre optimal de cases pour obtenir la meilleure performance est égal à S, le seuil de rejet. Il reste maintenant à déterminer la valeur de S. Le tableau 4.6 donne la performance du code ayant K=7 pour un rapport de signal-à-bruit E_b/N_0 = 4.61 dB, quand le seuil varie de 1 à 6. Le nombre de cases utilisées dans chaque cas est, bien sûr, le nombre optimal, c'est-à-dire, S. Nous remarquons que si le seuil est trop petit, S=1 ou 2, par exemple, le nombre de bits en erreur est très élevé car le chemin correct est perdu plus souvent. La resynchronisation prend beaucoup de temps et le décodage s'effectue de façon aléatoire pendant une longue période. Si S=1, on obtient une probabilité d'erreur par bit de 0.4344 alors que la probabilité de transition du canal est seulement de 4.5% environ pour le rapport de signal-à-bruit utilisé. La performance est donc très mauvaise bien que l'effort de décodage soit plus faible que dans les exemples ayant un seuil plus grand.

Le tableau 4.6 montre que la probabilité d'erreur est minimale si le seuil est

égal à 4. Si ce seuil est augmenté, la performance n'est pas améliorée, mais le nombre moyen de survivants est plus grand. 4 est donc la valeur optimale de S pour ce code. Un seuil plus grand ne fait qu'augmenter le temps de décodage et la complexité sans améliorer la performance. Le pouvoir de correction du code ayant K=7 et $d_{free} = 10$ est:

$$t = \left\lfloor \frac{d_{free} - 1}{2} \right\rfloor = \left\lfloor \frac{10 - 1}{2} \right\rfloor = 4$$

Un seuil de rejet égal au pouvoir de correction permet donc d'obtenir la meilleure performance. Cette observation a été confirmée par toutes les simulations qui ont été réalisées avec différents codes et rapports de signal-à-bruit.

Seuil S	Prob. d'erreur par bit	N.bits inc./ N.bits	Prob. d'ev. erreur (Nombre)	N.surv.
1	0.4344	$43444/10^5$	$7.4 \times 10^{-4}(74)$	3.79
2	1.831×10^{-2}	$18314/10^6$	$5.25 \times 10^{-4}(525)$	5.38
3	1.618×10^{-3}	$1618/10^6$	$2.45 \times 10^{-4}(245)$	13.07
4	1.238×10^{-3}	$1238/10^6$	$2.24 \times 10^{-4}(224)$	29.08
5	1.239×10^{-3}	$1239/10^6$	$2.25 \times 10^{-4}(225)$	46.83
6	1.239×10^{-3}	$1239/10^6$	$2.25 \times 10^{-4}(225)$	57.60

Tableau 4.6: Influence du seuil sur la probabilité d'erreur

K=7, R=1/2, $E_b/N_0 = 4.61$ dB

Expérimentalement, nous avons constaté dans les simulations réalisées que si $S = t = \lfloor (d_{free} - 1)/2 \rfloor$, le pouvoir de correction du code, la performance de cet algorithme est très proche de celle de l'AV. Il n'y a aucun avantage à l'augmenter. Une analyse plus détaillée de la probabilité d'erreur et une comparaison des résultats de simulation de l'AV et de l'AVA sont faites dans le chapitre suivant. Nous allons

aussi montrer pourquoi un seuil de rejet égal au pouvoir de correction donne une performance quasi-optimale.

Nous remarquons aussi que si S passe de 2 à 3, c'est-à-dire si on veut corriger trois symboles en erreur au lieu de deux, le nombre moyen de survivants augmente de 5.38 à 13.07. Ceci correspond à un rapport de 13.07/5.38 = 2.429. De même, si le seuil de rejet passe de 3 à 4, le nombre de survivants est multiplié par 2.225. Anderson [19] a montré que si le taux de codage est $R = 1/2$, le nombre de chemins d'un algorithme de décodage "breadth-first" — un tel algorithme ne permet pas les retours en arrière (voir [16]) — augmente selon 2.414^t, où t est le nombre d'erreurs à corriger. Les résultats de nos simulations confirment donc les travaux d'Anderson. Notons que la relation n'est pas valide pour S=1, 5 et 6, car dans ces cas, S ne correspond pas au nombre d'erreurs corrigées. En effet, le pouvoir de correction du code est égal à 4, donc le décodeur ne peut corriger que quatre erreurs au maximum. De plus, pour S=1, le chemin correct est perdu tellement souvent (car il y a eu plus d'une transition) que le décodeur n'est même pas capable de corriger les erreurs simples.

Les propriétés et l'opération de l'Algorithme de Viterbi Adaptatif ont été décrites dans ce chapitre. L'influence des paramètres sur la performance a aussi été étudiée et nous en avons déduit les valeurs permettant d'optimiser la performance. La probabilité d'erreur, l'effort de calcul moyen et la dynamique du décodage de cet algorithme sont considérées dans le chapitre 5.

Chapitre 5

Performance de l'algorithme de Viterbi adaptatif

Le chapitre précédent a permis de mieux comprendre l'AVA, l'algorithme de Viterbi adaptatif. Nous avons étudié ses propriétés et son fonctionnement. Dans ce chapitre, la performance est analysée et vérifiée par simulation sur l'ordinateur. La probabilité d'erreur et l'effort de calcul sont ensuite comparés avec ceux de l'AV. La dynamique du décodage et surtout le tampon d'entrée du décodeur sont également considérés.

5.1 Analyse de la probabilité d'erreur

Nous allons maintenant déterminer la probabilité d'erreur ou plutôt, la probabilité d'événement-erreur de l'AVA. On peut classer tous les événements-erreurs possibles dans deux catégories:

(a) les erreurs causées par le dépassement du pouvoir de correction du code. Elles surviennent même si un algorithme optimal, tel l'AV, est utilisé, c'est-à-dire,

même si l'état correct n'est pas perdu.

(b) les erreurs résultant de la perte de l'état correct et qui auraient pu être corrigées si l'état correct n'était pas perdu, c'est-à-dire, si l'AV était utilisé.

La deuxième condition dans la catégorie (b) permet d'exclure les erreurs qui n'auraient pas existé avec l'algorithme de Viterbi. Elle est très importante pour que les deux classes soient disjointes. En effet, il se peut que le bruit dans le canal provoque la perte de l'état correct mais *aussi* le dépassement du pouvoir de correction. Ce type d'erreurs entre donc dans la catégorie (a) et doit être exclu de la catégorie (b).

Soient

$P(E)$: Probabilité d'avoir un événement-erreur

P_V: Probabilité d'être dans la catégorie (a) (erreur de type Viterbi)

$P(E/V)$: Probabilité d'avoir un événement-erreur, sachant que nous sommes dans la catégorie (a): état correct non perdu (AV utilisé)

P_L: Probabilité d'être dans la catégorie (b), c'est-à-dire, perte de l'état correct et pas d'erreur avec l'AV

$P(E/L)$: Probabilité d'avoir un événement-erreur, sachant que nous sommes dans la catégorie (b): état correct perdu

$$P(E) = P(E/V)P_V + P(E/L)P_L \tag{5.1}$$

$P(E/V)$ est la probabilité d'erreur d'un algorithme optimal, tel l'AV. Notons

$$P(E/V) = P(E)_{opt}$$

Une erreur entre soit dans la catégorie (a), soit dans la catégorie (b) car les événements (a) et (b) sont mutuellement exclusifs. Par conséquent, nous pouvons

écrire:

$$P_V + P_L = 1$$

L'equation 5.1 devient donc

$$\begin{aligned} P(E) &= P(E)_{opt}[1 - P_L] + P(E/L)P_L & (5.2) \\ &= P(E)_{opt} + P_L[P(E/L) - P(E)_{opt}] & (5.3) \end{aligned}$$

Mais, si l'état correct est perdu, il est certain qu'une erreur va se produire. Donc

$$P(E/L) = 1$$

De plus, si E_b/N_0 est suffisamment grand

$$P(E)_{opt} \ll 1$$

Par conséquent, l'équation 5.3 peut s'écrire

$$P(E) \approx P(E)_{opt} + P_L \qquad (5.4)$$

Nous pouvons distinguer deux cas extrêmes:

1^{er} **cas:** $M = 2^{K-1}$ Si le nombre de survivants à chaque niveau du treillis est égal à 2^{K-1}, cela équivaut à utiliser l'AV. L'état correct ne peut pas être perdu, donc $P_L = 0$. L'équation 5.4 devient $P(E) = P(E)_{opt}$. Comme on s'y attendait, la probabilité d'événement-erreur de l'AVA est égale à celle d'un algorithme optimal dans ce cas.

2^{eme} **cas:** M **faible** Si le nombre de survivants est faible, l'état correct est souvent perdu. La probabilité P_L est beaucoup plus grande que $P(E)_{opt}$, donc

$$P(E) \approx P_L$$

Dans les autre cas, la probabilité d'erreur est donnée par l'équation 5.4.

La dégradation de l'AVA par rapport à l'AV est égale à P_L, la probabilité de perdre le chemin correct *sans* dépasser le pouvoir de correction du code. P_L représente la dégradation résultant de la conservation d'un nombre de survivants inférieur à 2^{K-1}. Inversement, P_L représente l'amélioration maximale possible de la performance d'erreur si le nombre moyen de survivants de l'AVA est augmenté jusqu'à 2^{K-1}. P_L est la probabilité de perdre l'état correct *et* de ne pas avoir d'erreur avec l'AV; c'est donc le produit de deux probabilités: la probabilité de perdre l'état correct, notée P_{perte}, et la probabilité que l'AV soit correct, notée $P_{correct}$

$$P_L = P_{perte} P_{correct} \tag{5.5}$$

Nous allons maintenant déterminer P_{perte} si le nombre maximal de survivants N_{max} est égal à 2^{K-1}. Soit S le seuil de rejet. L'état correct sera alors rejeté ou perdu s'il y a eu plus de S transitions dans le canal. En effet, supposant que la distance d'un chemin incorrect n'augmente pas, c'est-à-dire, les symboles reçus, ayant subi des transitions à cause du bruit, deviennent identiques aux symboles codés de ce chemin incorrect, la distance entre le chemin correct et la séquence reçue sera alors supérieure à MIN + S. Le chemin correct ne fait donc pas partie des survivants et il est perdu. Considérons un canal BSC. Si p est la probabilité de transition du canal, la probabilité de perdre le chemin correct, P_{perte}, est égale à la probabilité d'avoir S transitions ou plus (il n'est pas nécessaire d'avoir S transitions successives, mais ces transitions doivent être à l'intérieur d'une longueur d'historique)

$$P_{perte} = \sum_{i=S}^{\infty} \alpha_i p^i \tag{5.6}$$

où α_i est une constante qui représente toutes les combinaisons de i transitions provoquant la perte de l'état correct. Si le canal est assez bon, c'est-à-dire si p est

petit, la somme de l'équation 5.6 est dominée par le premier terme (i=S) et par conséquent

$$P_{perte} \approx \alpha_S p^S \tag{5.7}$$

Si N_{max} est inférieur à 2^{K-1}, il faut ajouter à P_{perte} la probabilité de perdre le chemin correct parce que N_{max} chemins ont une meilleure métrique que lui. Nous avons déjà vu à la section 4.5.2 que cette probabilité est faible (le problème majeur résultant de la limitation du nombre de chemins est la resynchronisation beaucoup plus longue du décodeur et non la perte du chemin correct). Ce cas n'est pas traité ici car nous supposons que les paramètres sont choisis de façon à optimiser la performance.

Si $S = t = \left\lfloor \frac{d_{free}-1}{2} \right\rfloor$, la probabilité que l'AV soit correct après S transitions ou plus dans le canal, c'est-à-dire la probabilité que la séquence transmise soit plus proche de la séquence reçue qu'un autre chemin, est très faible. t est le pouvoir de correction *garanti* du code; selon les transitions, il est possible mais peu probable qu'un décodeur optimal corrige plus de t transitions. Cela dépend du spectre du code. Un tel exemple est illustré à la figure 5.1(a), où un code ayant une longueur de contrainte K=3 est utilisé. Pour ce code, t=2. On voit que même si la séquence transmise a subi trois transitions, le décodeur de Viterbi ne rejette pas le chemin correct, alors qu'avec l'AVA, le chemin correct est perdu.

Cependant, en général, si le nombre i de transitions est supérieur à t, le nombre de séquences ayant subi i transitions qui ont pu être décodées sans erreur est très petit. Ceci explique pourquoi la valeur optimale du seuil de rejet trouvée expérimentalement à la section 4.5.3 est égale au pouvoir de correction. Seule une petite fraction des chemins ayant subi S transitions ou plus peut être décodée sans

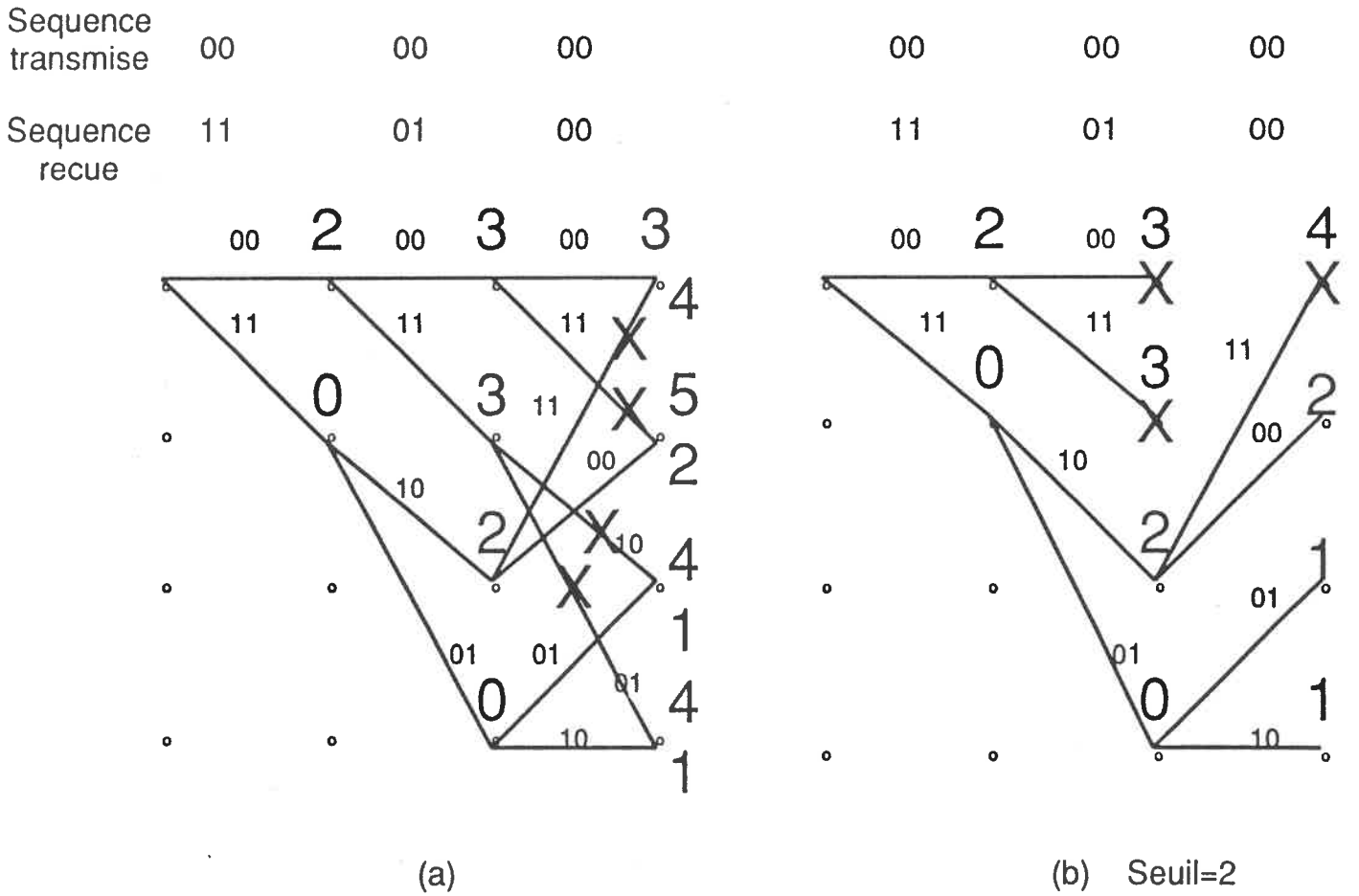

Figure 5.1: Exemple de décodage par l'AV et l'AVA: comparaison. (a) chemin correct non rejeté par l'AV (si aucune transition par la suite, ce chemin deviendra le meilleur après quelques branches). (b) chemin correct rejeté par l'AVA (événement-erreur)

erreur, d'où

$$P_{correct} = \sum_{i=S}^{\infty} \frac{n_i}{\alpha_i} \tag{5.8}$$

où n_i est le nombre de chemins ayant subi i transitions qui ont pu être décodés sans erreur et α_i, tel que défini précédemment, est le nombre de combinaisons de i transitions provoquant la perte de l'état correct. Il est évident que n_i tend vers zéro quand i, le nombre de transitions, grandit. La somme de l'équation 5.8 est donc dominée par le premier terme:

$$P_{correct} \approx \frac{n_S}{\alpha_S} \tag{5.9}$$

En substituant les équations 5.7 et 5.9 dans 5.5, nous obtenons

$$P_L \approx n_S p^S \tag{5.10}$$

Si p est petit, nous pouvons fixer $n_S = 1$ pour obtenir

$$P_L \approx p^S \tag{5.11}$$

En conclusion, si S=t, la performance de l'AVA est sensiblement la même que celle de l'AV: la dégradation, de l'ordre de p^S, est négligeable. L'AVA est donc quasi-optimal. Nous allons vérifier cette conclusion théorique par des simulations sur ordinateur.

5.2 Résultats de simulation

En ce qui concerne la performance, nous allons nous intéresser à la probabilité d'erreur et à la complexité, définie comme l'effort moyen de décodage, en nombre de survivants par bit décodé. Les générateurs des codes utilisés sont donnés dans l'Annexe A.

5.2.1 Probabilité d'erreur

Nous avons vu que si le seuil S est égal à t, le pouvoir de correction du code, la dégradation de l'AVA sur l'AV est minime ($\approx p^S$). On peut penser que la dégradation est plus importante si S, donc t, n'est pas très grand, c'est-à-dire, pour les codes dont la longueur de contrainte est petite (K=3 ou 5). Les simulations réalisées avec les mêmes séquences d'information et de bruit montrent que ce n'est pas le cas: la probabilité d'événement-erreur de l'AVA est très proche de celle de l'AV pour tous les rapports de signal-à-bruit considérés bien que le nombre de survivants à chaque niveau soit plus faible. Le nombre de cases utilisées et le seuil de rejet sont égaux au pouvoir de correction des codes (2 cases pour K=3 et 3 pour K=5) et le nombre maximal de survivants n'est pas limité ($N_{max} = 2^{K-1}$). Nous constatons même que dans certains cas, la performance est améliorée: l'AV produit plus d'événements-erreurs que l'AVA. Il peut sembler paradoxal d'obtenir une meilleure performance qu'un algorithme optimal, l'AV, en réduisant le nombre de survivants.

La raison de cette amélioration a déjà été donnée au chapitre 4: l'AVA possède un avantage intrinsèque sur l'AV en cas d'égalité (convergence de deux chemins au même noeud avec des métriques égales). Alors que l'AV choisit le survivant au hasard dans ce cas, l'AVA favorise le chemin qui a la plus grande probabilité d'être le chemin correct (celui qui était le meilleur au niveau précédent du treillis). Ainsi, l'AV fait une erreur dans 50 % des cas mais l'AVA parvient à rendre une décision correcte dans la plupart des cas et par conséquent, le nombre d'événements-erreurs peut être inférieur avec l'AVA. Nous n'avons pas tenu compte de ce phénomène dans l'analyse de la probabilité d'erreur car il est aléatoire et difficile à analyser. Nous le considérons juste comme un boni.

De plus, la resynchronisation est très rapide si la longueur de contrainte K

	Algorithme de Viterbi		Algorithme de Viterbi Adaptatif		
	Bits incor.	Ev. erreurs	Bits incor.	Ev. erreurs	
E_b/N_0 (N.bits)	Probab. (Nombre)	Probab. (Nombre)	Probab. (Nombre)	Probab. (Nombre)	N. surv.
3.0dB (50000)	3.40×10^{-2} (1700)	8.56×10^{-3} (428)	3.414×10^{-2} (1707)	8.72×10^{-3} (436)	3.03
3.5dB (50000)	2.04×10^{-2} (1020)	5.80×10^{-3} (290)	2.112×10^{-2} (1056)	5.96×10^{-3} (298)	2.90
4.61dB (200000)	5.56×10^{-3} (1112)	2.165×10^{-3} (433)	5.535×10^{-3} (1107)	2.095×10^{-3} (419)	2.61
5.5dB (500000)	1.422×10^{-3} (711)	6.5×10^{-4} (325)	1.420×10^{-3} (710)	6.34×10^{-4} (317)	2.41
6.0dB (800000)	5.725×10^{-4} (458)	2.84×10^{-4} (227)	6.312×10^{-4} (505)	3.012×10^{-4} (241)	2.32
7.0dB (2×10^6)	8.80×10^{-5} (176)	4.75×10^{-5} (95)	7.8×10^{-5} (156)	4.30×10^{-5} (86)	2.17

Tableau 5.1: Performance du code ayant K=3 (S=2)

est faible (nous supposons que les paramètres sont choisis de façon optimale: le nombre maximal de survivants $N_{max} = 2^{K-1}$). Le nombre d'erreurs (bits) causées par chaque événement-erreur est donc limité et la probabilité d'erreur par bit est aussi très proche de celle de l'AV. En comparaison, l'algorithme M de Lin [9] donne une bonne probabilité d'événement-erreur mais la probabilité d'erreur par bit est loin d'être satisfaisante (voir chapitre 3). En ce qui concerne la probabilité d'erreur, l'AVA satisfait donc l'objectif fixé au chapitre 4: obtenir une performance proche de celle de l'AV. Les tableaux 5.1 et 5.2 montrent aussi que le nombre moyen de calculs pour décoder un bit est inférieur à 2^{K-1}, satisfaisant une autre des propriétés désirées.

Les résultats précédents nous ont permis de vérifier que si les paramètres sont choisis de façon optimale (voir section 4.5), la performance d'erreur de l'AVA est

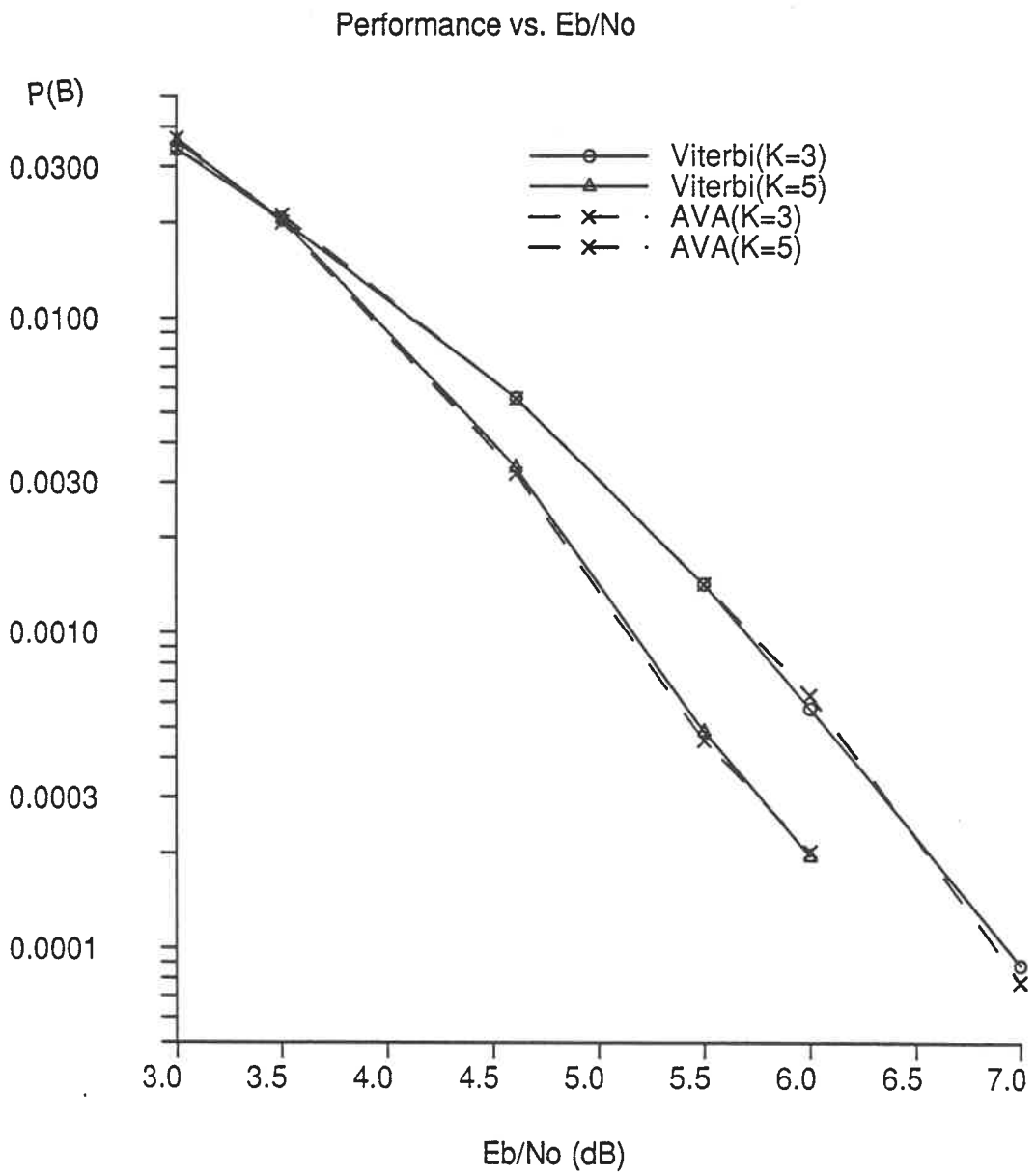

Figure 5.2: Performance des codes ayant K=3 et K=5

Pour K=3, $N_{max} = 4$ et $S = 2$.
Pour K=5, $N_{max} = 16$ et $S = 3$.

	Algorithme de Viterbi		Algorithme de Viterbi Adaptatif		
	Bits incor.	Ev. erreurs	Bits incor.	Ev. erreurs	
E_b/N_0 (N.bits)	Probab. (Nombre)	Probab. (Nombre)	Probab. (Nombre)	Probab. (Nombre)	N. surv.
3.0dB (50000)	3.602×10^{-2} (1801)	5.28×10^{-3} (264)	3.682×10^{-2} (1841)	5.14×10^{-3} (257)	12.31
3.5dB (50000)	2.044×10^{-2} (1022)	3.48×10^{-3} (174)	2.002×10^{-2} (1001)	3.36×10^{-3} (168)	11.61
4.61dB (200000)	3.33×10^{-3} (666)	7.35×10^{-4} (147)	3.19×10^{-3} (638)	7.10×10^{-4} (142)	10.04
5.5dB (1500000)	4.84×10^{-4} (726)	1.273×10^{-4} (191)	4.54×10^{-4} (681)	1.18×10^{-4} (177)	8.99
6.0dB (3×10^6)	1.947×10^{-4} (584)	4.867×10^{-5} (146)	2.02×10^{-4} (606)	5.167×10^{-5} (155)	8.52

Tableau 5.2: Performance du code ayant K=5 (S=3)

presque similaire à celle de l'AV. Toutefois, sur le plan pratique, ces codes, ayant une petite longueur de contrainte (K=3 et 5), sont moins intéressants. Nous allons maintenant considérer des codes ayant une longueur de contrainte supérieure ou égale à 7 (K=7 représente actuellement la longueur de contrainte maximale des décodeurs de Viterbi disponibles sur le marché).

La figure 5.3 montre que pour K=7 et K=8, la performance de l'AV est très proche de celle de l'AVA si on utilise quatre cases (les séquences d'information et de bruit des deux décodeurs sont identiques). Pour ces deux codes, la distance libre d_{free} est égale à 10. Le pouvoir de correction est donc

$$t = \left\lfloor \frac{10-1}{2} \right\rfloor = 4$$

Par conséquent, nos simulations concordent encore une fois avec l'analyse faite plus haut: le nombre de cases pour obtenir une performance quasi-optimale est t. Pour

K=9, la performance de l'AVA est légèrement inférieure à celle de l'AV car nous avons utilisé quatre cases alors que le pouvoir de correction t de ce code ayant une distance libre 12 est égal à 5.

Le gain maximal, en dB, que l'AVA peut fournir sur un décodeur de Viterbi (K=7) est maintenant déterminé. La figure 5.3 compare la probabilité d'erreur de l'AV à celles de l'AVA (K entre 7 et 10, nombre de cases=4). Dans tous les cas, on suppose que le gain de vitesse est 64, c'est-à-dire, le décodeur peut accomplir 64 calculs pendant le temps d'interarrivée de deux bits d'information. Les tableaux 5.3, 5.4 et 5.5 montrent que la longueur maximale de la queue dans le tampon d'entrée n'est pas excessive. La dynamique du décodage est traitée dans la section suivante. Seule la performance d'erreur est considérée ici. On voit que pour une probabilité d'erreur proche de 10^{-5}, l'AVA (K=10) peut procurer approximativement un gain de 0.8 dB sur l'AV (K=7). Comme l'AV peut fournir un gain de 3 dB sur un système non codé (K=7, quantification dure, probabilité d'erreur=10^{-5}), l'AVA peut donc améliorer ce gain de 30% environ avec un effort de décodage moyen équivalent: les deux décodeurs peuvent accomplir 64 calculs pendant le temps d'interarrivée de deux bits.

5.2.2 Complexité

La complexité d'un décodeur dépend du nombre moyen de calculs à effectuer. Ce nombre, à son tour, dépend du nombre moyen de survivants, qui est maintenant considéré. La complexité de l'AVA est comparée à celle de l'AV aux figures 5.4 à 5.6 (E_b/N_0=4.61 dB, 5.5 dB et 6.0 dB). La figure 5.5 montre que pour quatre cases, le nombre moyen de survivants est de 30 approximativement si $E_b/N_0 = 5.5$ dB. Ce nombre ne varie que très légèrement avec la longueur de contrainte K. Ainsi, pour K=8, l'AVA nécessite à peu près quatre fois moins de survivants en moyenne

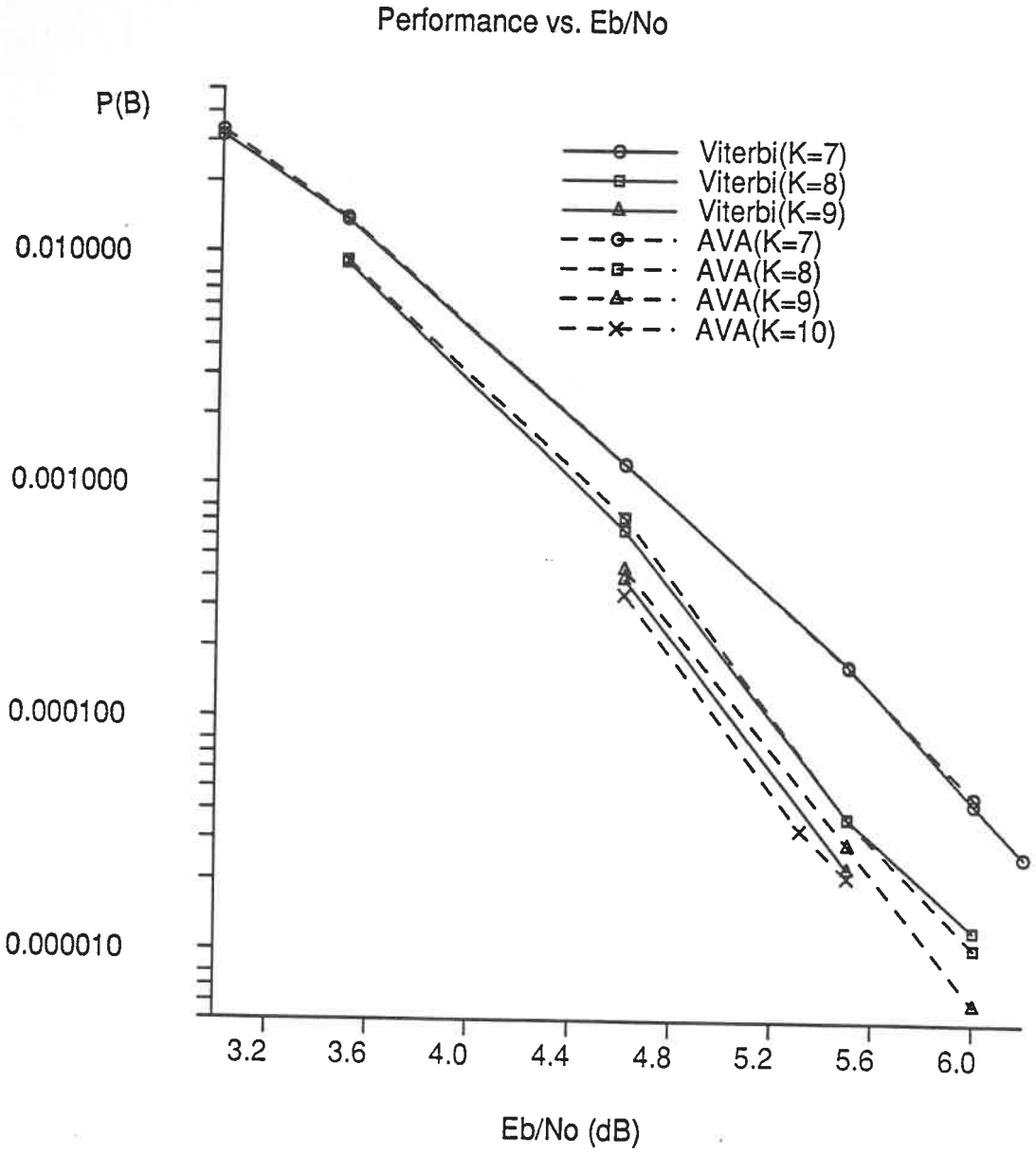

Figure 5.3: Comparaison de l'AVA et de l'AV

Pour l'AVA, $N_{max} = 2^{K-1}$ et $S = 4$.

Algorithme de Viterbi

K	N.surv.	P(B)	N.erreurs	N.bits
6	32	2.05×10^{-3}	1640	8×10^5
7	64	1.229×10^{-3}	1229	1×10^6
8	128	6.395×10^{-4}	1279	2×10^6
9	256	3.96×10^{-4}	1584	4×10^6

Algorithme de Viterbi Adaptatif

	K	N.surv.	P(B)	Queue max.	N.erreurs	N.bits
	7	29.08	1.238×10^{-3}	164(32)	1238	1×10^6
4 cases	8	36.56	7.27×10^{-4}	86	1454	2×10^6
	9	39.26	4.372×10^{-4}	218	1749	4×10^6
	10	48.28	3.394×10^{-4}	628	1697	5×10^6
	11	49.40	3.448×10^{-4}	1508	1724	5×10^6
	12	58.44	3.653×10^{-4}	4215	548	1.5×10^6

Tableau 5.3: Performance d'erreur à Eb/No = 4.61 dB

Dans les tableaux 5.3 à 5.5, la taille maximale de la queue ou file d'attente a été obtenue pour un gain de vitesse G=64 calculs/temps d'interarrivée de deux branches. Lorsqu'un G différent a été utilisé, il est indiqué entre parenthèses après la taille de la queue. Par exemple, pour K=7, G=32.

Algorithme de Viterbi

K	N.surv.	P(B)	N.erreurs	N.bits
4	8	9.93×10^{-4}	993	1×10^6
5	16	4.84×10^{-4}	726	1.5×10^6
6	32	2.84×10^{-4}	568	2×10^6
7	64	1.71×10^{-4}	856	5×10^6
8	128	3.80×10^{-5}	152	4×10^6
9	256	2.287×10^{-5}	343	15×10^6

Algorithme de Viterbi Adaptatif

	K	N.surv.	P(B)	Queue max.	N.erreurs	N.bits
	7	9.53	2.445×10^{-4}	30(32)	489	2×10^6
3 cases	8	10.57	1.357×10^{-4}	41(64)	543	4×10^6
	9	9.80	4.023×10^{-4}	561	6034	15×10^6
	7	23.83	1.686×10^{-4}	61	843	5×10^6
	8	26.01	3.750×10^{-5}	46	150	4×10^6
4 cases	9	25.57	2.90×10^{-5}	143	435	15×10^6
	10	29.47	2.115×10^{-5}	456	846	4×10^7
	11	29.12	2.144×10^{-5}	2049	1072	5×10^7
	8	57.15	3.75×10^{-5}	159	150	4×10^6
5 cases	9	62.82	2.22×10^{-5}	1495	333	15×10^6
	10	75.16	1.32×10^{-5}	191(128)*	368	27.8×10^6

Tableau 5.4: Performance d'erreur à Eb/No = 5.5 dB

* Cette taille maximale de la file d'attente a été obtenue pour un bloc d'information de 4×10^6 bits.

Algorithme de Viterbi

K	N.surv.	P(B)	N.erreurs	N.bits
6	32	9.275×10^{-5}	371	4×10^6
7	64	4.40×10^{-5}	220	5×10^6
8	128	1.26×10^{-5}	63	5×10^6

Algorithme de Viterbi Adaptatif

	K	N.surv.	P(B)	Queue max.	N.erreurs	N.bits
	7	8.21	7.35×10^{-5}	22(32)	294	4×10^6
3 cases	8	9.11	5.992×10^{-5}	347(32)	719	12×10^6
	9	8.22	1.434×10^{-4}	1011(32)	1434	10×10^6
	7	21.69	4.72×10^{-5}	41(32)	236	5×10^6
4 cases	8	21.98	1.064×10^{-5}	57	266	25×10^6
	9	20.98	6.05×10^{-6}	137	242	40×10^6

Tableau 5.5: Performance d'erreur à Eb/No = 6.0 dB

(26.0 au lieu de 128) pour une probabilité d'erreur égale. L'amélioration est encore plus spectaculaire pour l'AVA (K=10), dont la performance est similaire et même légèrement supérieure à celle de l'AV (K=9) mais requiert seulement 29.5 survivants en moyenne, soit 8 fois moins que l'AV (256 survivants). Ces simulations ont été réalisées avec $N_{max} = 2^{K-1}$.

De plus, le caractère adaptatif de l'AVA lui permet d'être encore plus efficace si le canal est bon: si le rapport signal-à-bruit grandit, le nombre moyen de calculs diminue avec l'AVA alors qu'il reste constant avec l'AV. La figure 5.7 illustre la diminution du nombre moyen de survivants, obtenu par simulation, en fonction du rapport E_b/N_0. Par conséquent, plus le rapport E_b/N_0 est élevé, plus il est avantageux d'utiliser l'AVA au lieu de l'AV.

Tel qu'indiqué dans la section 4.3, la seule différence entre l'AVA et l'AV réside dans la sélection des survivants: avec l'AV, les distances des chemins convergents sont comparées, tandis qu'avec l'AVA, la distance de chaque chemin prolongé est comparée à un seuil. Si le nombre de survivants est le même, la complexité additionnelle de l'AVA sur l'AV est minime. Comme le nombre moyen de survivants de l'AVA est plus petit, il peut utiliser un décodeur moins complexe que l'AV et donner la même probabilité d'erreur ou encore, garder la même complexité que le décodeur de Viterbi mais utiliser un code plus puissant, donc obtenir une probabilité d'erreur plus faible.

Cependant, un tampon d'entrée est nécessaire et un problème important reste à considérer. Le nombre moyen de survivants donne une bonne idée de l'effort de décodage. Mais, comme ce nombre varie dynamiquement pendant le décodage, nous devons étudier la dynamique, c'est-à-dire, la façon dont l'effort de calcul varie avec le bruit et la façon dont le tampon d'entrée se remplit et se vide. Le problème majeur du décodage séquentiel, le débordement du tampon d'entrée, doit bien sûr

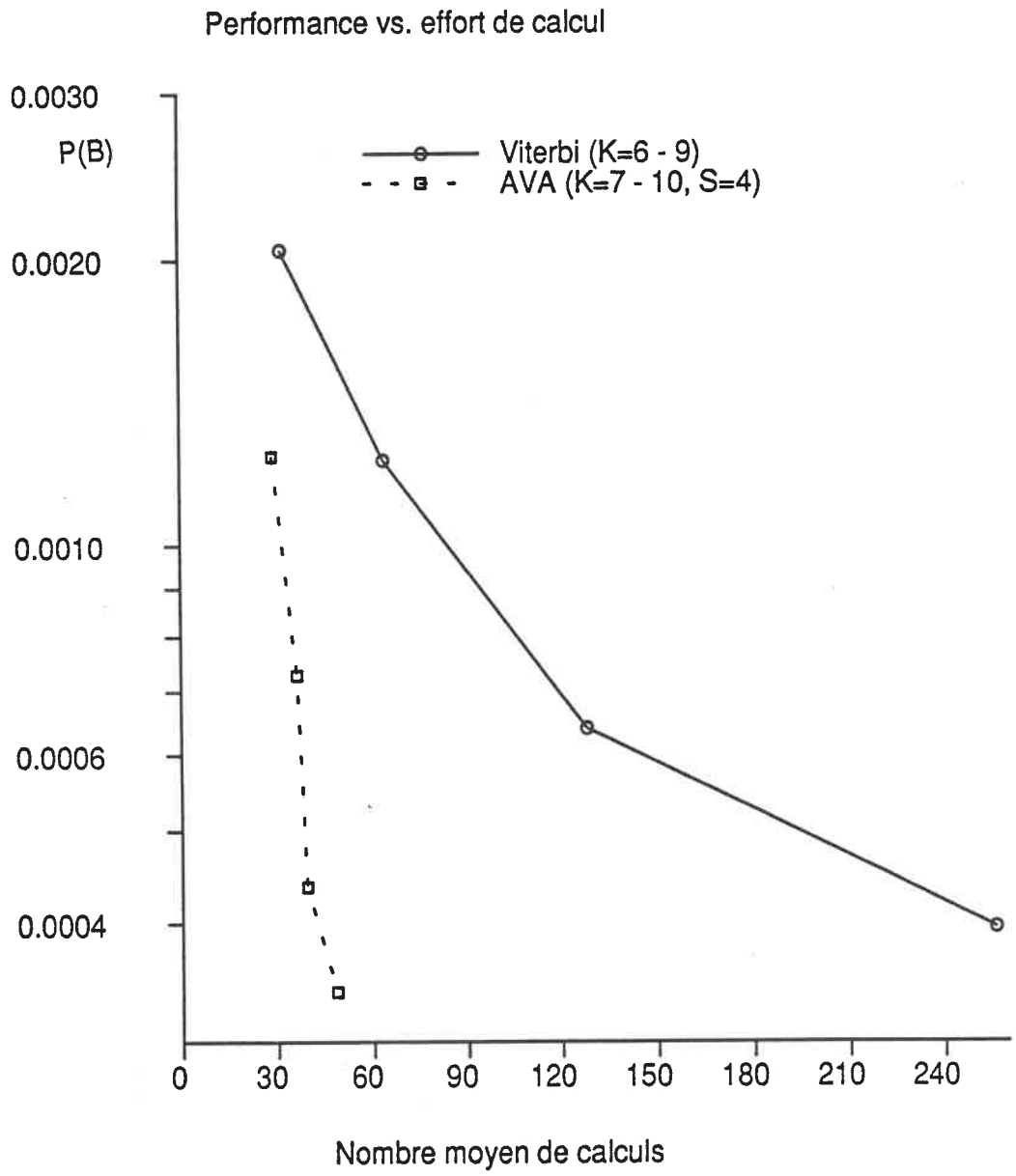

Figure 5.4: P(B) en fonction du nombre de calculs à 4.61 dB

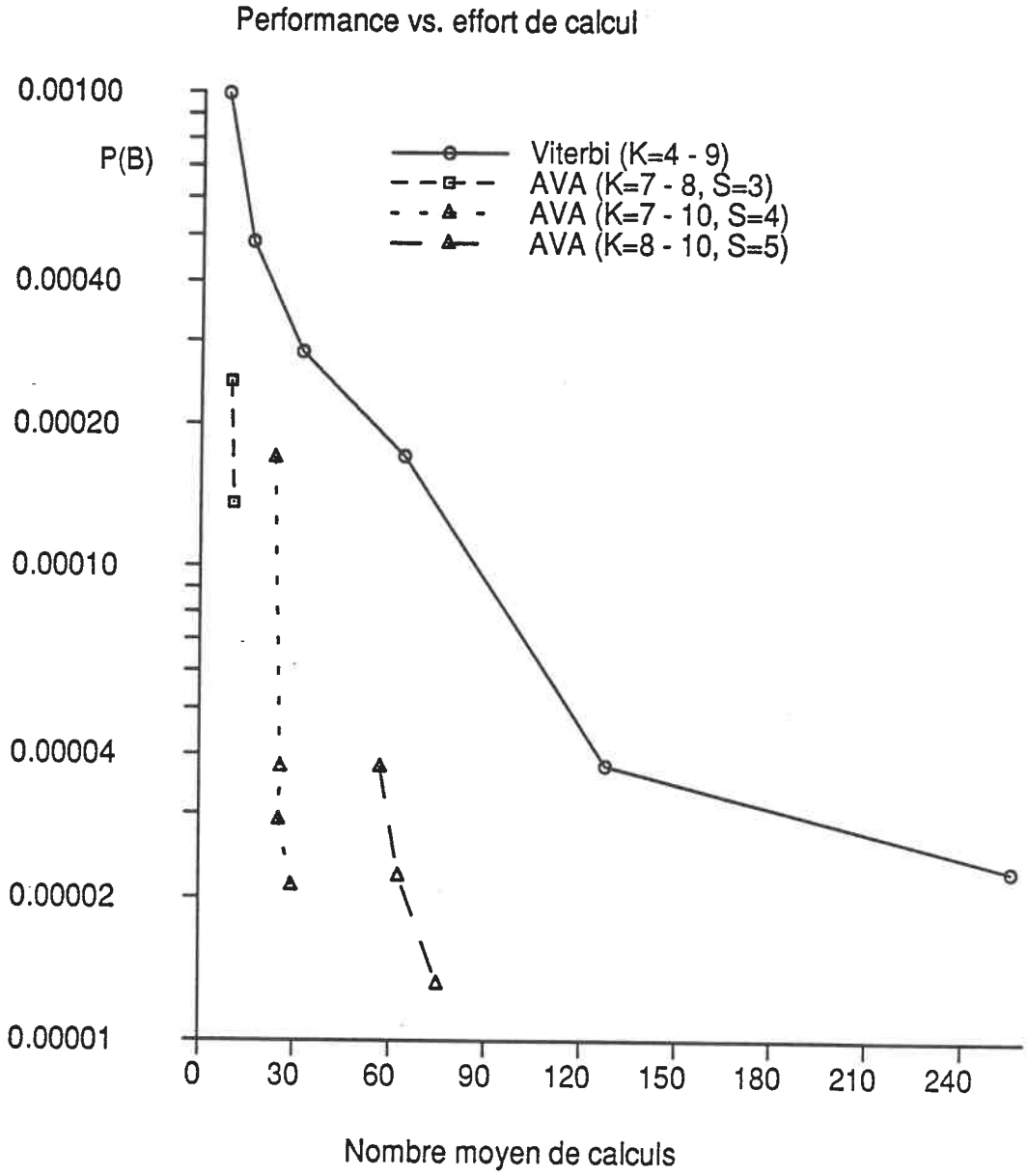

Figure 5.5: P(B) en fonction du nombre de calculs à 5.5 dB

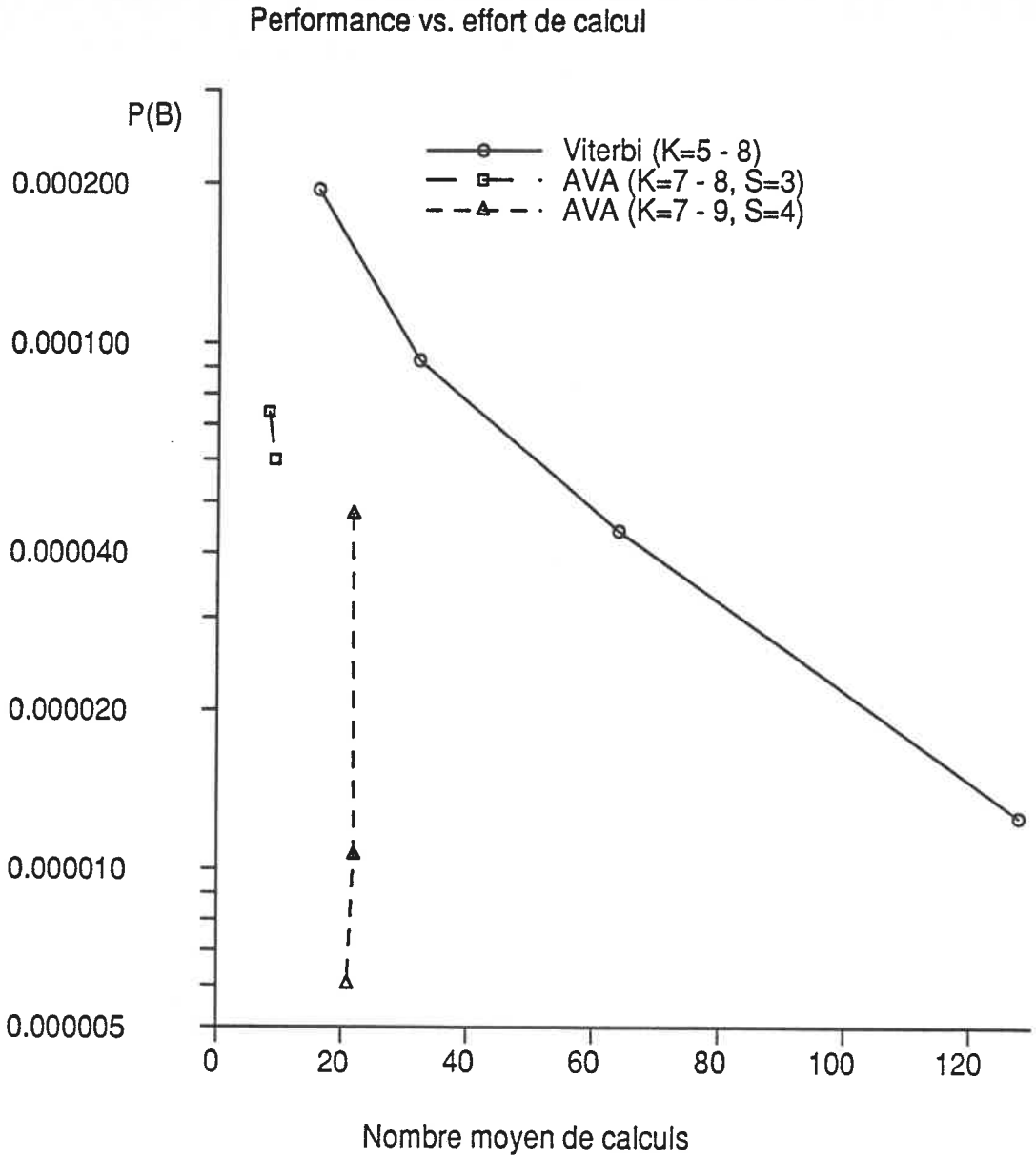

Figure 5.6: P(B) en fonction du nombre de calculs à 6.0 dB

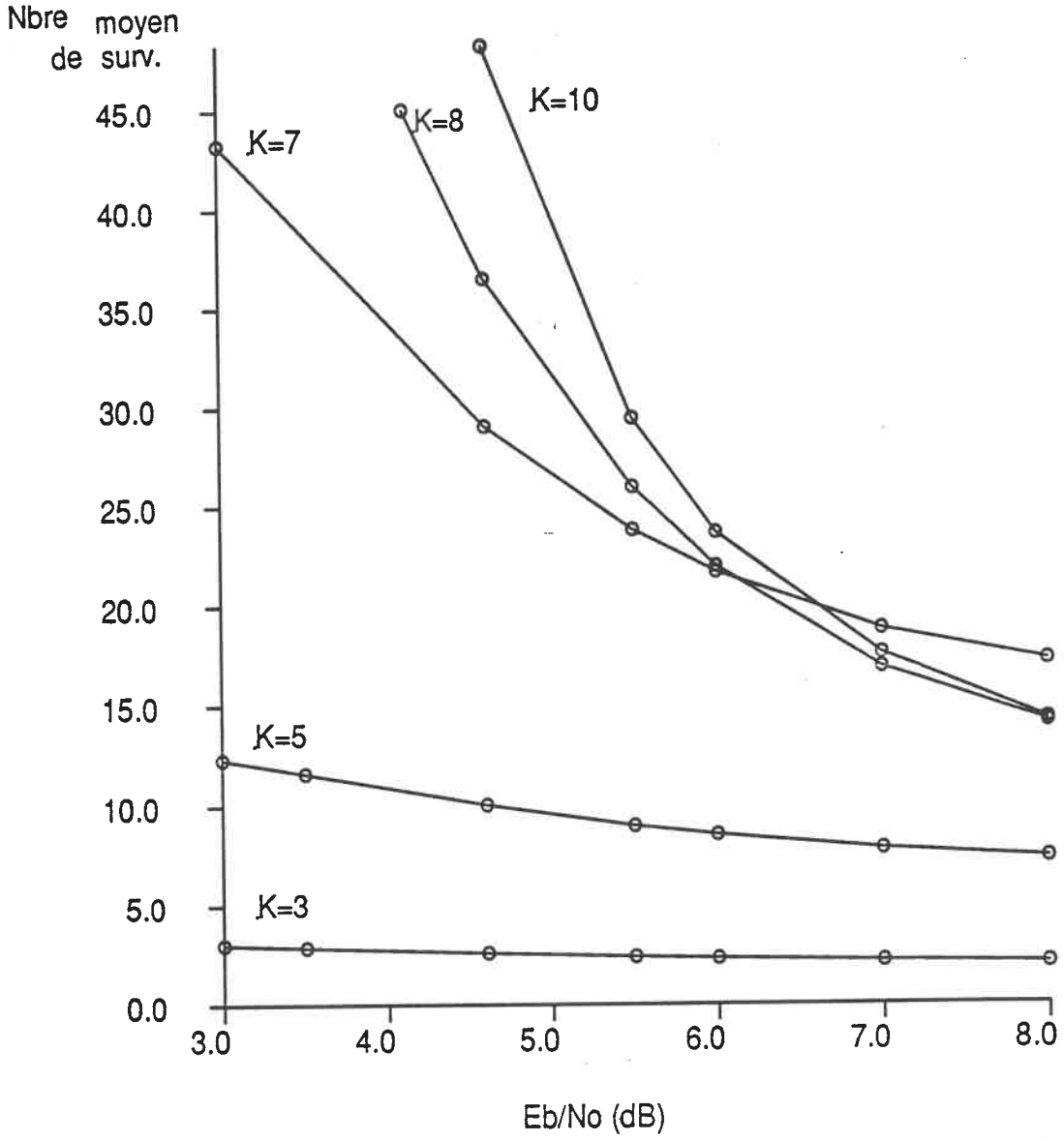

Figure 5.7: Nombre de survivants en fonction de E_b/N_0

$N_{max} = 2^{K-1}$; $S = 4$, sauf pour $K = 3$ et $K = 5$, où $S = 2$ et 3, respectivement.

être évité.

5.3 Dynamique du décodage

5.3.1 Nombre de survivants en fonction du bruit

L'effort moyen de calcul a été considéré dans la section précédente. Le comportement instantané du décodeur si une transition survient dans le canal est étudié ici. Nous savons déjà que le nombre de survivants augmente si les symboles reçus subissent des transitions (voir section 4.4). Des simulations sont exécutées en utilisant le code ayant une longueur de contrainte K=8. Les paramètres sont choisis de façon optimale, c'est-à-dire, le nombre maximal de survivants est égal à 128 et 4 cases sont utilisées. Comme exemple, le nombre dynamique de survivants est observé après une transition à la branche 100. Il est évident, et ceci a été vérifié par plusieurs simulations, que le nombre de survivants varie de la même façon si c'est une autre branche qui est affectée par une transition. Le canal est calme (pas de bruit) avant et après la transition. N_{min}, qui a été défini à la section 4.4, est le nombre de survivants pour le code et le seuil de rejet considérés, lorsque le canal est calme. Ici, $N_{min} = 12$. La figure 5.8 montre que le nombre de survivants augmente graduellement de N_{min} à 34 et redescend ensuite à N_{min}. S'il y a deux ou trois transitions, le comportement est identique mais le nombre maximal de survivants est plus grand et il faut plus de temps avant que ce nombre retourne à N_{min}.

Ces simulations montrent que le nombre de survivants dépend du bruit (ce qui est évident car l'AVA s'adapte à la qualité du canal) mais aussi du nombre de survivants au niveau précédent du treillis.

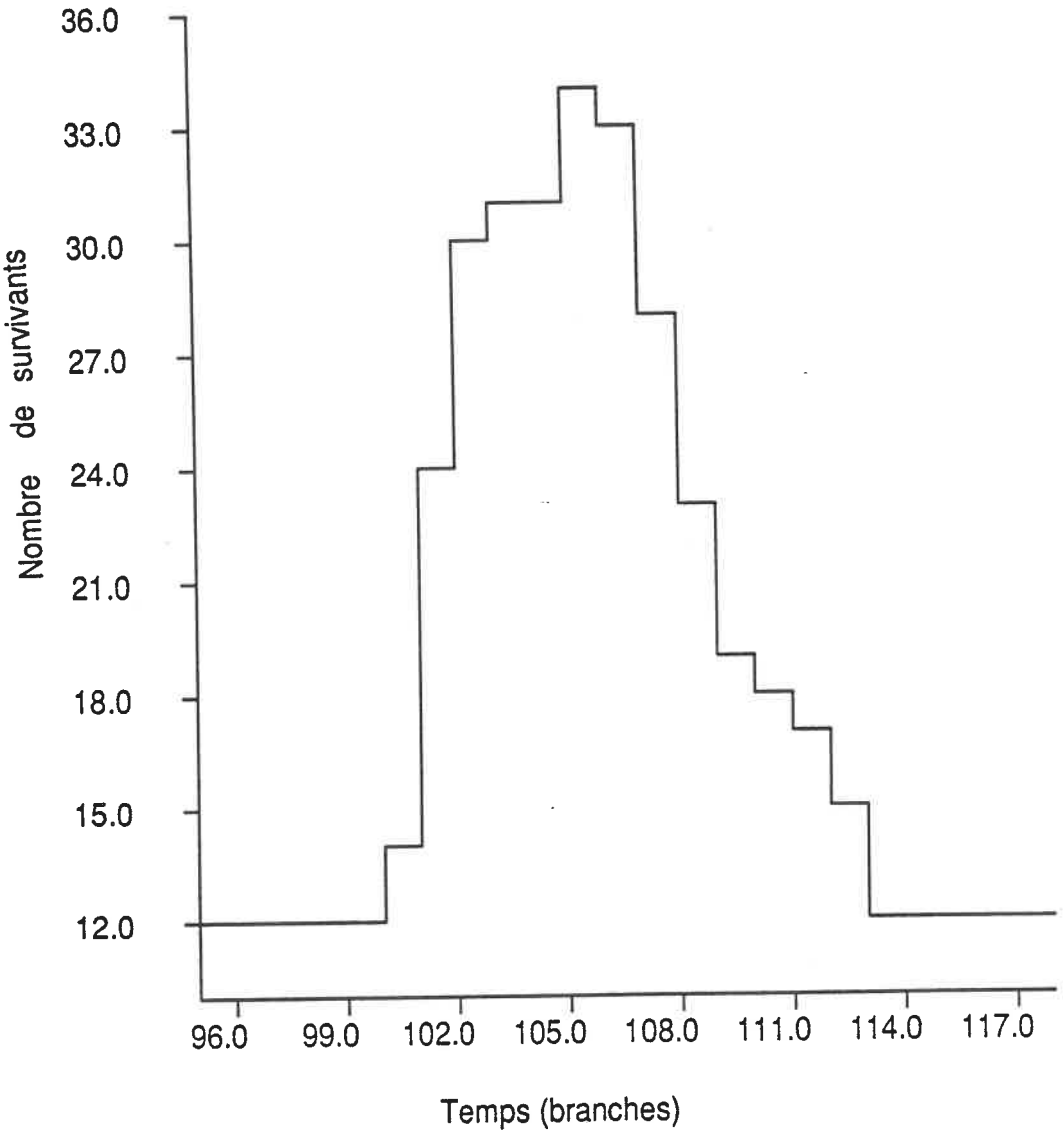

Figure 5.8: Nombre dynamique de survivants en fonction du temps
La branche 100 est affectée par une transition, le canal est calme (pas de bruit) avant et après cette transition.

5.3.2 Distribution du nombre de calculs

Comme nous l'avons mentionné au chapitre 2, l'effort de calcul du décodage séquentiel a une distribution de type Pareto [1], [15]. Un raisonnement simple permet de montrer que l'effort de calcul de l'AVA a une distribution exponentielle. Nous avons vu au chapitre 2 que la distribution de Pareto est le résultat de l'effet combiné de deux comportements exponentiels. Ici, la probabilité d'avoir une chute de la métrique du chemin correct supérieure à d décroît encore exponentiellement avec d pour les mêmes raisons (canal sans mémoire). Par contre, le nombre de calculs dus à la chute d ne croît plus exponentiellement avec d: les retours en arrière sont interdits et le nombre de survivants est limité. Dans le pire des cas, le nombre des survivants est 2^{K-1}, une constante. L'effet combiné des deux comportements (l'un décroît exponentiellement, l'autre est borné par une constante) donne donc une distribution exponentielle.

La figure 5.9 montre $P(C > N)$ en fonction de N, où C est le nombre de calculs. Les données utilisées pour tracer ces courbes sont les résultats de simulations de l'Annexe B. En utilisant une échelle logarithmique pour l'axe vertical et une échelle linéaire pour l'axe horizontal, on voit que $P(C > N)$ varie linéairement avec N, si N est compris entre 35 et 120 à peu près. Nous allons vérifier la nature exponentielle de la distribution de l'effort de calcul.

La fonction de répartition d'une variable aléatoire exponentielle C, dont la moyenne est égale à $1/\lambda$, est donnée par [20]

$$F_C(N) = P(C \leq N) = 1 - e^{-\lambda N} \qquad N \geq 0 \tag{5.12}$$

Il s'ensuit que

$$P(C > N) = 1 - P(C \leq N) = e^{-\lambda N} \tag{5.13}$$

La fonction $e^{-\lambda N}$ a été tracée sur la figure 5.9 pour différentes valeurs de λ, proches

de l'inverse du nombre moyen de survivants obtenu par simulation (voir Annexe B, pour les résultats des simulations). Par exemple, si $E_b/N_0 = 4.61$ dB, le nombre moyen de survivants est égal à 36.56. On remarque que $e^{-N/36}$ suit de très près $P(C > N)$, dont les valeurs ont été obtenues par simulation. Nous avons donc vérifié que l'effort de calcul décroît exponentiellement avec N.

5.3.3 Tampon d'entrée

Dans le cas de l'AVA, le tampon d'entrée est non vide seulement si le nombre de calculs est supérieur au gain de vitesse. Comme exemple, supposons que le décodeur puisse traiter 64 survivants pendant le temps d'interarrivée de deux branches d'information, qui est considéré comme l'unité de temps. Ainsi, si le nombre de survivants est 128, le décodeur va prendre deux unités de temps pour les traiter. En général, si le gain de vitesse est 64, le nombre d'unités de temps nécessaires est $C/64$, où C est le nombre de calculs ou survivants. Ici, comme il faut deux unités de temps, une branche est arrivée pendant que le décodeur est en train de traiter la branche précédente. Cette branche est stockée dans le tampon d'entrée. Soit q la taille de la file d'attente dans le tampon d'entrée. La probabilité que q excède une certaine taille T_q est donc égale à la probabilité que le nombre de calculs C dépasse un certain nombre N_c, N_c dépendant de T_q et du gain de vitesse

$$P(q > T_q) = P(C > N_c) \tag{5.14}$$

Nous avons déduit à la section 5.3.2 que $P(C > N_c)$ décroît exponentiellement. $P(q > T_q)$ a donc une décroissance exponentielle également. Par conséquent, si T_q est grand, la probabilité que la taille de la queue excède T_q est faible. Nous n'avons donc pas besoin d'un tampon d'entrée très grand et le débordement du tampon d'entrée, qui se produit fréquemment en décodage séquentiel, ne constitue

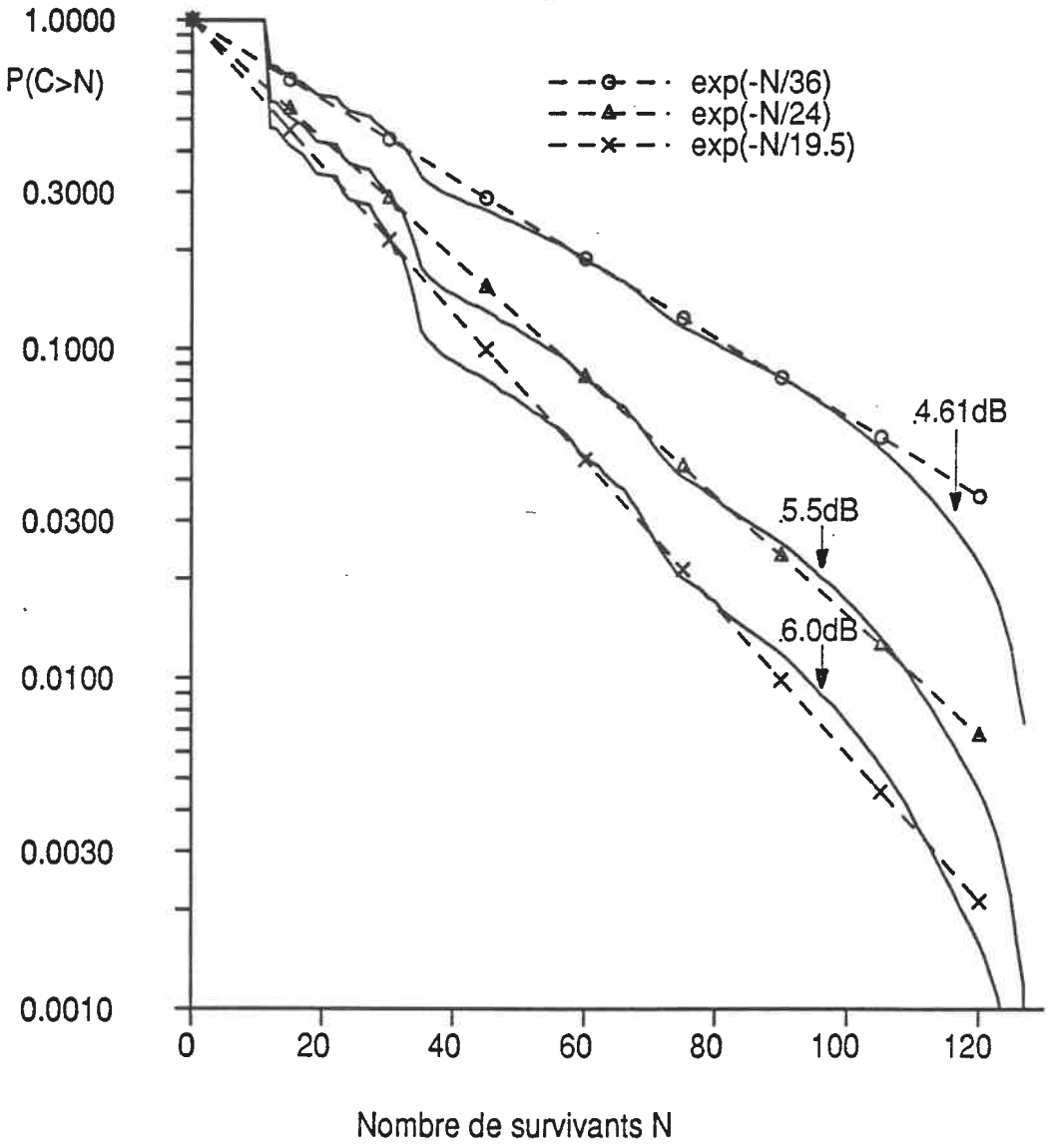

Figure 5.9: $P(C > N)$ en fonction de N

Longueur de contrainte du code K=8, $N_{max} = 128$, S=4.

pas un problème pour l'AVA, à condition que le gain de vitesse soit choisi de façon appropriée (voir aussi section 4.4). Comme on pouvait le prévoir, la taille du tampon peut être réduite si le gain de vitesse du décodeur est augmenté.

La figure 5.10 représente $P(q > T_q)$, observée par simulation, en fonction de la taille de la queue. Elle permet de vérifier que la file d'attente décroît exponentiellement car nous obtenons des droites (à l'exception de l'extrémité de la courbe correspondant à $E_b/N_0 = 5.5$ dB, ce qui est sans doute causé par un nombre insuffisant de bits lors de la simulation). Nous remarquons aussi que plus le rapport signal-à-bruit est élevé, plus la pente est forte: la file d'attente décroît plus rapidement si le canal est bon, comme prévu.

Les données de la figure 5.10 proviennent des mêmes simulations que celles qui ont été réalisées pour calculer le facteur d'utilisation ρ du chapitre 4, ainsi que la fontion de répartition de la section précédente. Ces données peuvent être trouvées dans l'Annexe B. Comme ρ est inférieur à 1, nous avons conclu, au chapitre 4, que la file d'attente est stable et que sa taille maximale ne doit pas être très élevée. La taille maximale obtenue est inférieure à 90 branches, ce qui est très modeste. Par conséquent, nous avons vérifié la conclusion basée sur le calcul de ρ: le gain de vitesse est suffisant et la file d'attente est bien stable.

Le tableau 5.6 récapitule les résultats de toutes les simulations effectuées avec un seuil S=4 et donne la taille maximale de la queue du tampon d'entrée. Certaines simulations ont été faites avec un bloc de 40×10^6 bits d'information. Si un bloc plus petit est utilisé, il est peu probable que la queue excède la taille maximale obtenue. Cette valeur donne donc une bonne idée du tampon d'entrée nécessaire. Le tableau montre que la taille du tampon n'est pas excessive et dépasse rarement 1000 branches, ce qui est très raisonnable. Si la longueur de contrainte du code est supérieure à 10, la taille maximale de la file d'attente commence à être importante.

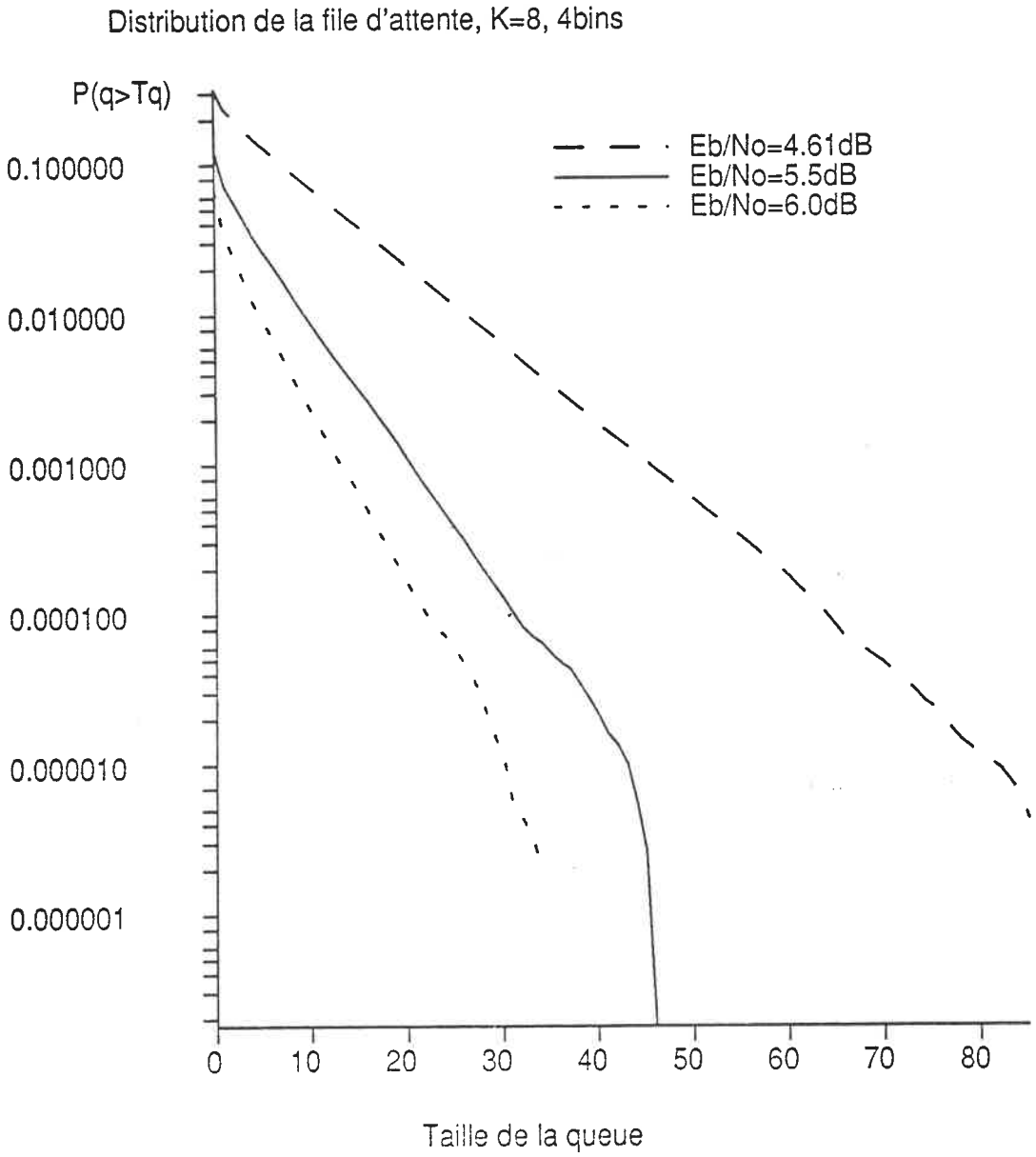

Figure 5.10: $P(q > T_q)$ vs. taille de la queue

Longueur de contrainte du code K=8, $N_{max} = 128$, S=4.

4 cases, gain de vitesse = 64, $N_{max} = 2^{K-1}$

E_b/N_0	4.61 dB		5.5 dB		6.0 dB	
K	Queue max	N. bits	Queue max	N. bits	Queue max	N. bits
8	86	2×10^6	46	4×10^6	57	25×10^6
9	218	4×10^6	143	15×10^6	137	40×10^6
10	628	5×10^6	456	40×10^6		
11	1508	5×10^6	2049	50×10^6		
12	4215	1.5×10^6				

Tableau 5.6: Taille maximale (en branches) de la file d'attente

Toutefois, nous avons vu à la section précédente que la performance d'erreur de ces codes est moins bonne que celle du code ayant K=10 si le nombre de cases est égal à 4. Par conséquent, ces codes ne devraient pas être choisis sauf si le nombre de cases et le gain de vitesse sont augmentés.

La dynamique de la file d'attente est maintenant examinée pour un décodeur ayant un gain de vitesse égal à 64. La figure 5.11 donne un exemple du comportement de la file d'attente en fonction du temps ainsi que le nombre instantané de survivants. La file d'attente apparaît sous la forme de pics disjoints. Cela signifie que le tampon d'entrée se vide régulièrement: même si le décodeur accumule un retard, il arrive à le rattraper rapidement. Les branches ne s'accumulent pas de façon excessive dans le tampon et le gain de vitesse est donc satisfaisant. Nous venons de vérifier encore une fois qu'un décodeur, utilisant un code dont la longueur de contrainte K=8 et pouvant accomplir 64 calculs en moyenne pendant le temps d'interarrivée de deux branches, a une file d'attente stable lorsque le rapport de signal-à-bruit est égal à 6.0 dB. Ce résultat n'est pas surprenant car nous avons déjà vu, au chapitre 4, que le facteur d'utilisation ρ est égal à 0.3434 dans ce cas. Comme ρ est inférieur à 1, il est normal d'avoir une file d'attente stable.

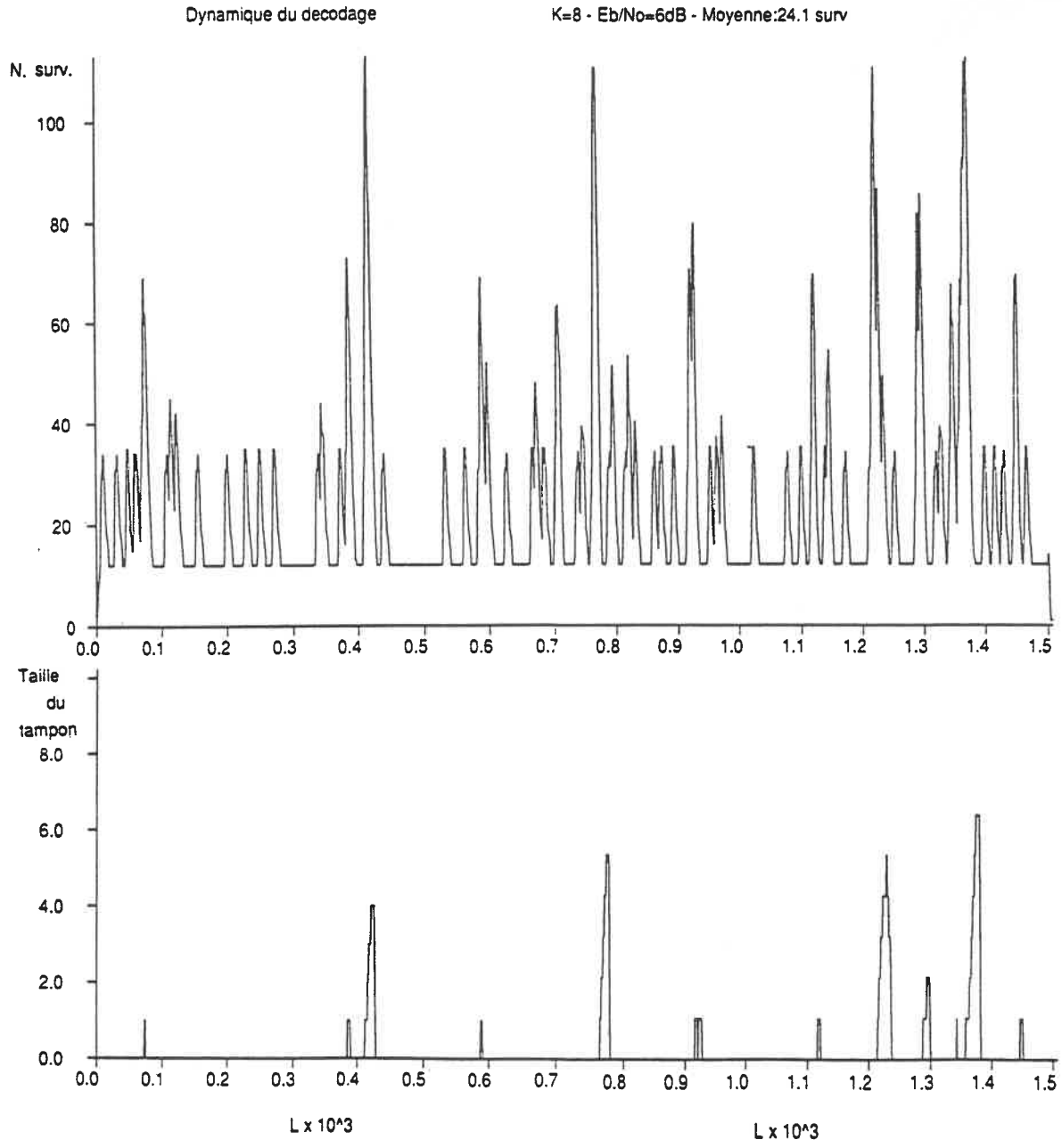

Figure 5.11: Dynamique de la file d'attente

G=64 calculs/temps d'interarrivée de deux branches

Une approche théorique de l'analyse de la file d'attente a été tentée mais n'a pas donné de résultats concluants. Cette analyse est difficile car si nous connaissons la loi d'arrivée dans le tampon d'entrée (elle est déterministe, le temps d'interarrivée entre deux bits étant constant), la loi de service est, en revanche, inconnue. Nous ne savons pas exactement quel est le temps de service, c'est-à-dire, le temps nécessaire pour décoder un bit. Nous savons seulement qu'il dépend du bruit dans le canal et du nombre de survivants au niveau précédent du treillis. Le problème de l'analyse reste donc entier.

En conclusion, nous avons montré que la fonction de répartition de la file d'attente au tampon d'entrée suit une loi exponentielle. L'AVA présente donc un avantage considérable sur le décodage séquentiel, dont la fonction de répartition suit asymptotiquement une loi de type Pareto, car une loi exponentielle décroît beaucoup plus rapidement. En pratique, un décodeur AVA peut être réalisé avec un tampon de taille modeste si le gain de vitesse est suffisant, comme dans les exemples précédents.

5.4 Choix du code

La performance de l'AVA a été analysée et vérifiée par simulation dans les sections précédentes. Il a été montré que si les paramètres sont choisis de façon adéquate, une probabilité d'erreur quasi-optimale est obtenue. De plus, la variabilité de l'effort de calcul ne crée pas de problème si le gain de vitesse est satisfaisant. Nous allons maintenant examiner comment concevoir un système de codage. Quel code doit-on utiliser pour obtenir une probabilité d'erreur $P(B)$ pour un rapport de signal-à-bruit donné et un effort de calcul minimal?

Les figures 5.12 à 5.14 montrent la performance d'erreur de l'AV et l'AVA en

fonction de la longueur de contrainte K ($E_b/N_0 = 4.61$ dB, 5.5 dB et 6.0 dB). Quand K grandit, la probabilité d'erreur de l'AVA diminue d'abord mais elle augmente ensuite. En effet, nous avons vu à la section 4.5.3 que pour chaque K, il y a un nombre de cases optimal, égal au pouvoir de correction du code. Si ce nombre est insuffisant pour le code utilisé, la probabilité de perdre le chemin est grande et la probabilité d'erreur par bit est élevée. Ces figures permettent de déterminer le code à utiliser pour un nombre de cases donné (le nombre de cases donne une indication du nombre de survivants, donc de la complexité). Afin d'obtenir la meilleure performance posssible, le code à choisir est celui qui se trouve au creux de la courbe. Par exemple, si $E_b/N_0 = 5.5$ dB et on veut approximativement 30 survivants, donc 4 cases, le minimum de la courbe de 4 cases, à la figure 5.13, survient pour K=10. Ce code permet d'obtenir la probabilité d'erreur minimale.

Ces courbes permettent aussi de déterminer le nombre minimal de cases à utiliser si on veut obtenir une certaine probabilité d'erreur. Supposons que $E_b/N_0 = 5.5$ dB et que l'on désire avoir $P(B) \approx 2 \times 10^{-5}$. D'après la figure 5.13, deux codes donnent cette probabilité d'erreur: le code ayant K=10 et utilisant 4 cases ou le code ayant K=9 et utilisant 5 cases. Cependant, l'effort de calcul moyen est différent. Dans le premier cas, le nombre moyen de survivants est de 29.47 alors que dans l'autre cas, il est de 62.82, soit plus du double. Donc, si on veut minimiser l'effort de calcul, le premier code doit être choisi.

En conclusion, l'effort de calcul de l'AVA dépend du seuil de rejet choisi et ne varie que légèrement avec la longueur de contrainte K. Malheureusement, il n'est pas possible d'augmenter indéfiniment K pour obtenir une meilleure performance d'erreur. Pour un seuil donné (donc, un effort de calcul donné), il existe une longueur de contrainte K optimale. Si K est supérieur à cette valeur, la probabilité d'erreur par bit se détériore. On doit donc utiliser le code optimal tel que déterminé

par les figures 5.12 à 5.14.

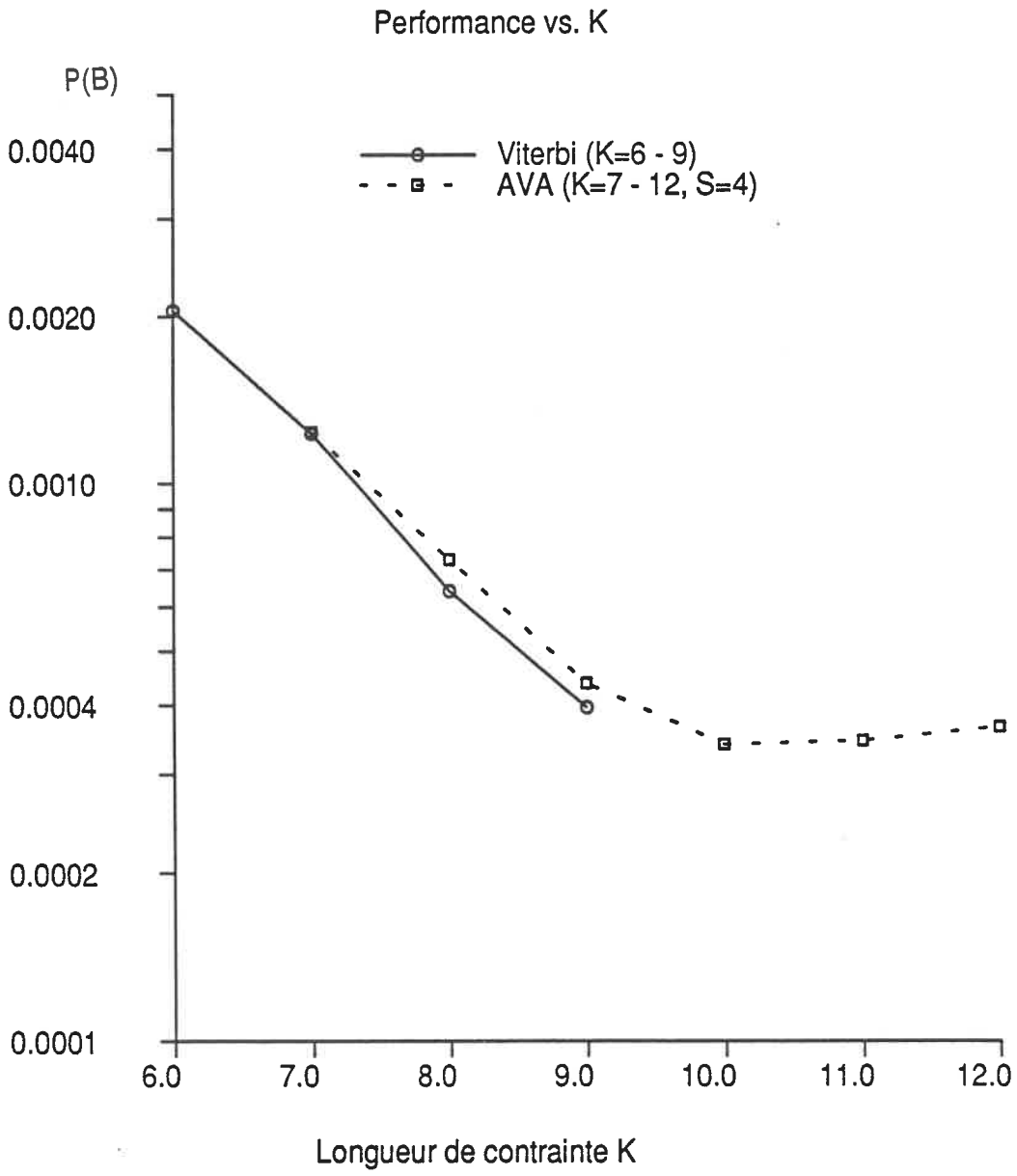

Figure 5.12: Choix du code optimal pour $E_b/N_0 = 4.61$ dB

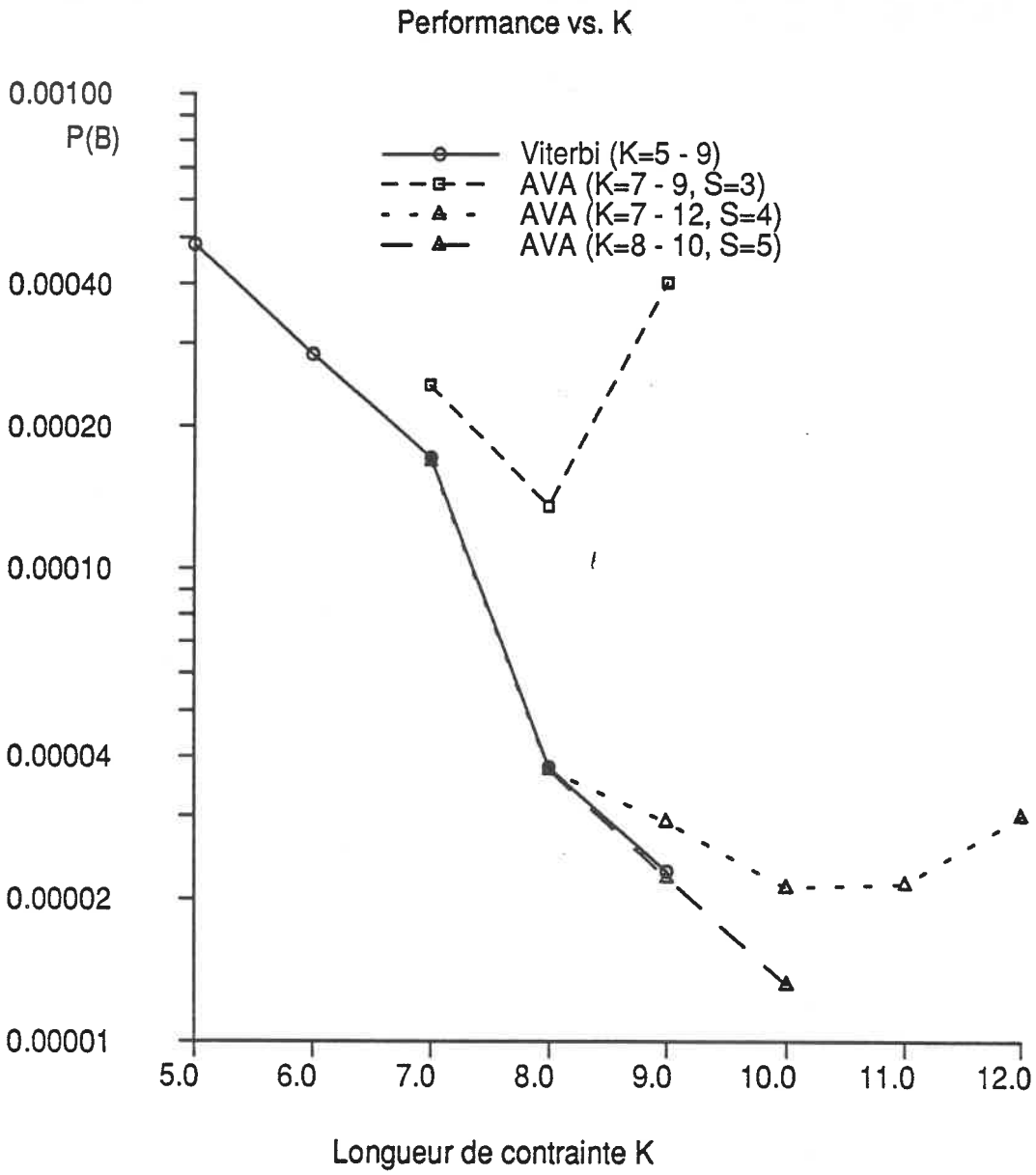

Figure 5.13: Choix du code optimal pour $E_b/N_0 = 5.5$ dB

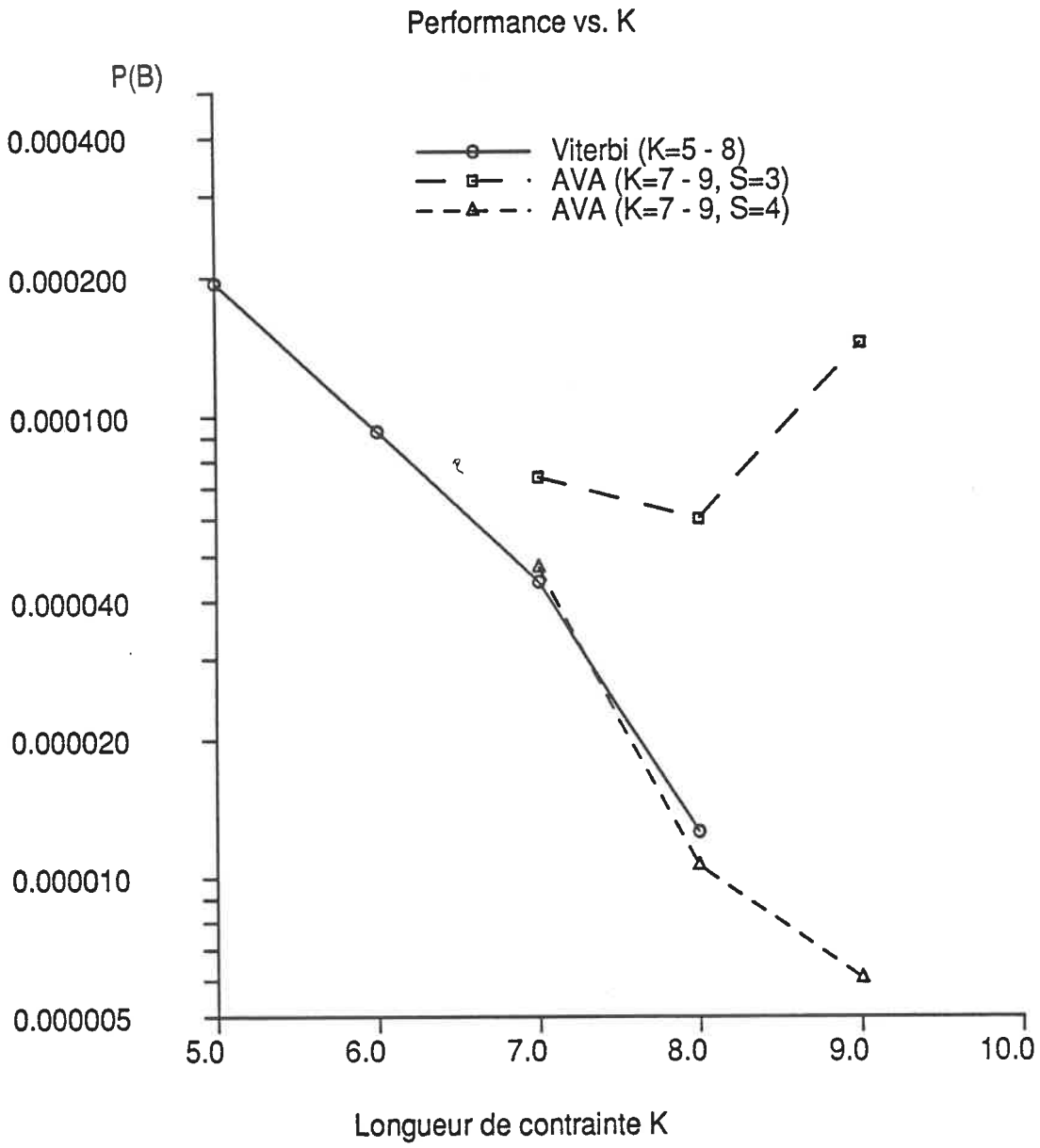

Figure 5.14: Choix du code optimal pour $E_b/N_0 = 6.0$ dB

Chapitre 6

Conclusion

Un nouvel algorithme de décodage probabiliste a été proposé dans ce mémoire. L'Algorithme de Viterbi Adaptatif (AVA), contrairement à l'algorithme de Viterbi, ne garde qu'un certain nombre des meilleurs noeuds à chaque niveau du treillis. Il en résulte un effort moyen de calcul plus faible et une efficacité supérieure à celle de l'AV. Comme l'effort de calcul de l'AVA ne dépend presque pas de la longueur de contrainte K, un code ayant un K plus élevé peut être utilisé. Cela n'augmente la complexité que légèrement. Ainsi, pour un effort de décodage moyen équivalent à celui d'un décodeur de Viterbi ayant K=7, c'est-à-dire pour un décodeur dont le gain de vitesse est 64, l'AVA permet d'utiliser un code ayant K=10. Les résultats des simulations confirment qu'il est possible d'obtenir ainsi une probabilité d'erreur inférieure à celle de l'AV pour une même complexité.

L'analyse de la performance d'erreur montre que la dégradation sur l'AV est minime. Des simulations réalisées avec le même code et dans les mêmes conditions (séquences d'information et de bruit identiques) nous ont permis de vérifier que la performance de l'AVA est très proche de celle de l'AV si les paramètres sont choisis de façon optimale. Dans certains cas, nous obtenons même une meilleure

performance car l'AVA présente un avantage intrinsèque sur l'AV en cas d'égalité des métriques.

De plus, l'effort de calcul, bien que variable, a une distribution exponentielle et non une distribution de type Pareto, comme le décodage séquentiel. Par conséquent, la taille du tampon d'entrée ne sera pas excessive et les simulations le confirment.

En conclusion, l'AVA permet donc d'obtenir une probabilité d'erreur faible et d'améliorer les méthodes classiques de décodage probabiliste. La complexité ne croît plus exponentiellement avec la longueur de contrainte et la probabilité de débordement du tampon d'entrée est maintenant très faible si le gain de vitesse est suffisant. L'AVA peut remplacer l'AV dans pratiquement tous les cas. Dans le pire cas, il va donner une performance similaire à celle de l'AV mais, la plupart du temps, comme l'effort moyen de décodage est réduit, un code plus puissant peut être utilisé et l'AVA permet d'obtenir une meilleure probabilité d'erreur pour une complexité équivalente.

L'analyse de la performance d'erreur, au chapitre 5, a montré que la dégradation de l'AVA sur l'AV est approximativement égale à la probabilité de perdre le chemin correct. Par conséquent, si nous parvenons à éviter la perte du chemin correct, la dégradation sera nulle et nous obtiendrons la performance d'un algorithme optimal. Pour améliorer l'AVA, nous devons donc étudier les moyens d'éliminer la perte du chemin correct. Ceci pourrait faire l'objet de recherches futures.

Une autre recommandation de recherche future concerne l'utilisation de codes perforés [21] et les applications de l'AVA, en particulier la modulation codée et l'égalisation pour un canal affecté par des interférences entre symboles (canal à largeur de bande limitée). Il serait intéressant d'étudier la performance de cet algorithme dans ces applications différentes du codage pour la correction d'erreurs.

Finalement, le tampon d'entrée mérite une analyse plus poussée. Nous avons juste indiqué que la taille de la file d'attente décroît exponentiellement. Les paramètres de cette distribution n'ont pas été examinés.

Bibliographie

[1] V. Bhargava, D. Haccoun, R. Matyas, P. Nuspl, *Digital Communications by Satellite*, John Wiley, New York, 1981.

[2] S. Lin et D. J. Costello, *Error Control Coding: Fundamentals and Applications*, Prentice-Hall, Englewood Cliffs, 1983.

[3] R. G. Gallager, *Information Theory and Reliable Communication*, John Wiley, New York, 1968.

[4] A. J. Viterbi, "Convolutional Codes and Their Performance in Communication Systems," *IEEE Trans. Commun. Technol.*, vol. COM-19, pp. 751–772, Oct. 1971.

[5] J. M. Wozencraft et J. M. Jacobs, *Principles of Communication Engineering*, John Wiley, New York, 1965.

[6] D. Haccoun, "Multiple-Path Stack Algorithms for Decoding Convolutional Codes," Ph. D. dissertation, Dept. of Elec. Eng., McGill Univ., Montreal, Juin 1974.

[7] D. Haccoun, M. Ferguson, "Generalized Stack Algorithm for the Decoding of Convolutional Codes," *IEEE Trans. Inform. Theory*, vol. IT-21, pp. 638–651, Nov. 1975.

[8] P. R. Chevillat, D. J. Costello, "A Multiple Stack Algorithm for Erasurefree Decoding of Convolutional Codes," *IEEE Trans. Commun.*, vol. COM-25, pp. 1460–1470, Déc. 1977.

[9] C.F. Lin et J. B. Anderson, "M-Algorithm Decoding of Channel Convolutional Codes," *Proc. Princeton Conference on Information Science and Systems*, pp. 362–366, Mars 1986.

[10] T. Aulin, "A New Trellis Decoding Algorithm — Analysis and Applications," Technical report Nr. 2, School of Elec. and Comp. Eng., Univ. of Technology, Gőteborg, Sweden, Déc. 1985.

[11] K. J. Larsen, "Short Convolutional Codes with Maximal Free Distances for Rate 1/2, 1/3, and 1/4," *IEEE Trans. Inform. Theory*, vol. IT-19, pp. 371–371, Mai 1973.

[12] A. J. Heller et I. M. Jacobs, "Viterbi Decoding for Satellite and Space Communication," *IEEE Trans. Commun. Technol.*, vol. COM-19, pp. 835–848, Oct. 1971.

[13] R. Smith, "Analyse des performances d'un décodeur convolutionnel basé sur l'algorithme de Viterbi tronqué", rapport de projet de Maitrise en ingénierie (M. Ing.), Dépt. de génie élect., Ecole Polytechnique, Montréal, Oct. 1987.

[14] G. D. Forney, Jr., "Convolutional Codes II: Maximum Likelihood Decoding," *Inf. Control*, 25, pp. 222–266, Juil. 1974.

[15] D. Haccoun, " Variabilité de calculs et débordements de décodeurs séquentiels à pile," *Traitement du signal*, vol. 3 — n^o 3, pp. 127–143, 1986.

[16] J. B. Anderson et S. Mohan, "Sequential Coding Algorithms: A Survey and Cost Analysis," *IEEE Trans. Commun.*, vol. COM-32, pp. 169–176, Fév. 1984.

[17] F. Jelinek et J. B. Anderson, "Instrumentable Tree Encoding of Information Sources," *IEEE Trans. Inform. Theory*, vol. IT-17, pp.118–119, Jan. 1971.

[18] L. Kleinrock, *Queueing Systems*, Vol. 1: Theory, John Wiley, New York, 1975.

[19] J. B. Anderson, "Limited Search Trellis Decoding of Convolutional Codes," *IEEE Trans. Inform. Theory*, vol. IT-35, pp. 944–955, Sep. 1989.

[20] S. M. Ross, *Introduction to Probability Models*, 4th edition, Academic Press, New York, 1989.

[21] D. Haccoun et G. Bégin, "High-Rate Punctured Convolutional Codes for Viterbi and Sequential Decoding," *IEEE Trans. Commun.*, vol. COM-37, pp. 1113–1125, Nov. 1989.

[22] J. Conan et D. Haccoun, "Reduced-State Viterbi Decoding of Convolutional Codes," *Proc. 14th Allerton Conference on Circuit and System Theory*, pp. 695–703, Oct. 1976, Monticello, Ill., USA.

Annexe A

Liste des codes convolutionnels

Dans le tableau A.1, les codes convolutionnels de taux R=1/2, qui ont une distance libre maximale, sont énumérés. Ces codes sont ceux qui ont été utilisés dans les simulations.

Les générateurs sont les coefficients des vecteurs de connexion

$$G_j = (g_{j1}, g_{j2}, ..., g_{jK}) \qquad j = 1, 2, ..., V$$

qui ont été introduits au chapitre 1. Ces générateurs sont exprimés en notation octale. Par exemple, 133 correspond à (1011011).

K	Generateurs		d_{free}
3	5	7	5
4	15	17	6
5	23	35	8
6	53	75	8
7	133	171	10
8	247	371	10
9	561	753	12
10	1167	1545	12
11	2335	3661	14
12	4335	5723	15
13	10533	17661	16
14	21675	27123	16

Tableau A.1: Codes convolutionnels de taux R=1/2, ayant une distance libre maximale
Référence: [11].

Annexe B

Résultats des simulations

A titre d'exemples, les données suivantes sont extraites des résultats de trois simulations et donnent la fréquence d'apparition de n_s, le nombre dynamique de survivants, ainsi que la taille de la file d'attente du tampon d'entrée. On considère que le canal est un BSC. Le code convolutionnel utilisé a une longueur de contrainte K=8 et un taux de codage R=1/2. Ses générateurs peuvent être trouvés dans l'Annexe A. Ces résultats ont servi pour calculer le facteur d'utilisation ρ (voir section 4.4) et pour tracer la fonction de distribution du nombre de calculs, ainsi que celle de la taille du tampon (voir section 5.3).

Les bits d'information sont générés aléatoirement: un générateur de nombre pseudo-aléatoire entre 0 et 1 est utilisé. Si le nombre généré est inférieur à 0.5, le bit d'information est un 0, sinon, c'est un 1.

Ce bit est encodé et on obtient deux symboles par branche. On génère un autre nombre aléatoire: s'il est inférieur à p, la probabilité de transition du canal, on suppose que le bruit affecte ce symbole. Dans ce cas, le symbole reçu est donc l'inverse du symbole transmis. Les symboles reçus sont décodés en suivant les étapes décrites au chapitre 4 (voir section 4.3 et figure 4.1). Finalement, le bit

d'information décodé est comparé au bit transmis, qui a été gardé en mémoire. S'il est différent, on conclut qu'il y a une erreur.

La taille de la file d'attente est examinée de la façon suivante: on suppose que le décodeur peut traiter G survivants pendant le temps d'interarrivée de deux branches. Ce temps est appelé un cycle. Au début de chaque cycle, une branche est reçue et le tampon d'entrée augmente d'une unité. Si le nombre de calculs (ou survivants) n_s nécessaires pour décoder cette branche est inférieur ou égal à G, le décodage est complété durant ce cycle et on diminue le tampon d'une unité. Lorsque le nombre dynamique de survivants n_s est supérieur à G, le décodeur ne peut traiter que G d'entre eux, les autres $(n_s - G)$ doivent être traités au cycle suivant. En supposant que le tampon d'entrée était vide au début du cycle, il contient maintenant une branche, à la fin du cycle. Le nombre de calculs requis $(n_s - G)$ est aussi gardé en mémoire. Quand la branche suivante arrive, elle doit attendre dans le tampon d'entrée, qui augmente d'une unité, que le décodage de la branche précédente soit complété. Si le décodage a pu être complété avant la fin du présent cycle, on diminue le tampon d'entrée et on commence le décodage de la branche suivante dans le tampon. Il est évident que la première branche arrivée est la première à être servie ("First-Come, First-Served"). Le même procédé est répété pour cette branche.

SIMULATION # 1

Longueur de contrainte du code: K = 8
Rapport de signal-a-bruit: Eb/No = 4.61 dB
Nombre de bits d'information = 2000000 bits
Parametres de simulation: Nmax = 128, Seuil=4
Gain de vitesse: G=64 calc./ tps d'interarrivee

Nombre de survivants	Nombre de fois	Probabilite
1	5	0.25000E-05
2	1	0.50000E-06
3	1	0.50000E-06
4	1	0.50000E-06
5	0	0.00000E+00
6	0	0.00000E+00
7	1	0.50000E-06
8	1	0.50000E-06
9	1	0.50000E-06
10	0	0.00000E+00
11	1	0.50000E-06
12	536950	0.26847E+00
13	18017	0.90085E-02
14	58930	0.29465E-01
15	45294	0.22647E-01
16	27252	0.13626E-01
17	29291	0.14646E-01
18	44491	0.22246E-01
19	47408	0.23704E-01
20	12931	0.64655E-02
21	8725	0.43625E-02
22	9270	0.46350E-02
23	65242	0.32621E-01
24	36334	0.18167E-01
25	13290	0.66450E-02
26	14589	0.72945E-02
27	12274	0.61370E-02
28	44550	0.22275E-01
29	38285	0.19142E-01
30	31711	0.15856E-01
31	54817	0.27408E-01
32	13568	0.67840E-02
33	53799	0.26899E-01
34	58091	0.29045E-01
35	55949	0.27974E-01
36	27163	0.13582E-01
37	18813	0.94065E-02
38	18595	0.92975E-02
39	13642	0.68210E-02
40	10538	0.52690E-02
41	11623	0.58115E-02
42	12757	0.63785E-02
43	8434	0.42170E-02
44	8804	0.44020E-02
45	10681	0.53405E-02
46	10113	0.50565E-02
47	14841	0.74205E-02
48	8025	0.40125E-02
49	8445	0.42225E-02
50	11222	0.56110E-02
51	9964	0.49820E-02
52	11282	0.56410E-02
53	8062	0.40310E-02
54	11034	0.55170E-02
55	8881	0.44405E-02

56	7581	0.37905E-02
57	9837	0.49185E-02
58	12532	0.62660E-02
59	14911	0.74555E-02
60	8313	0.41565E-02
61	9517	0.47585E-02
62	7108	0.35540E-02
63	11527	0.57635E-02
64	8671	0.43355E-02
65	8533	0.42665E-02
66	6157	0.30785E-02
67	12281	0.61405E-02
68	9399	0.46995E-02
69	12636	0.63180E-02
70	11298	0.56490E-02
71	10808	0.54040E-02
72	9185	0.45925E-02
73	8308	0.41540E-02
74	7867	0.39335E-02
75	6156	0.30780E-02
76	4718	0.23590E-02
77	4838	0.24190E-02
78	4735	0.23675E-02
79	5976	0.29880E-02
80	4570	0.22850E-02
81	6069	0.30345E-02
82	4596	0.22980E-02
83	4516	0.22580E-02
84	4415	0.22075E-02
85	4322	0.21610E-02
86	4242	0.21210E-02
87	3999	0.19995E-02
88	4057	0.20285E-02
89	4012	0.20060E-02
90	4087	0.20435E-02
91	4494	0.22470E-02
92	4439	0.22195E-02
93	4589	0.22945E-02
94	4755	0.23775E-02
95	4498	0.22490E-02
96	4738	0.23690E-02
97	3818	0.19090E-02
98	3865	0.19325E-02
99	4451	0.22255E-02
100	4058	0.20290E-02
101	4134	0.20670E-02
102	4133	0.20665E-02
103	4378	0.21890E-02
104	4078	0.20390E-02
105	4315	0.21575E-02
106	3752	0.18760E-02
107	4071	0.20355E-02
108	3889	0.19445E-02
109	3821	0.19105E-02
110	4312	0.21560E-02
111	4034	0.20170E-02
112	3466	0.17330E-02
113	3554	0.17770E-02
114	3582	0.17910E-02
115	3418	0.17090E-02
116	3423	0.17115E-02
117	3515	0.17575E-02
118	3367	0.16835E-02
119	3205	0.16025E-02
120	3397	0.16985E-02
121	3315	0.16575E-02

122	3849	0.19245E-02
123	3781	0.18905E-02
124	4770	0.23850E-02
125	4038	0.20190E-02
126	6211	0.31055E-02
127	3994	0.19970E-02
128	14734	0.73670E-02

Facteur d'utilisation rho = 0.571329
Nombre moyen de survivants = 36.5651
Variance = 864.648
Deviation standard = 29.4049

Taille max. de file d'attente	Nombre de fois	Probabilite
0	1363968	0.68198E+00
1	162455	0.81227E-01
2	68074	0.34037E-01
3	56168	0.28084E-01
4	48065	0.24032E-01
5	36949	0.18474E-01
6	30724	0.15362E-01
7	27699	0.13849E-01
8	25138	0.12569E-01
9	21346	0.10673E-01
10	17836	0.89180E-02
11	16167	0.80835E-02
12	14636	0.73180E-02
13	12798	0.63990E-02
14	10855	0.54275E-02
15	9799	0.48995E-02
16	8794	0.43970E-02
17	7747	0.38735E-02
18	6663	0.33315E-02
19	6133	0.30665E-02
20	5358	0.26790E-02
21	4866	0.24330E-02
22	4217	0.21085E-02
23	3853	0.19265E-02
24	3335	0.16675E-02
25	2992	0.14960E-02
26	2558	0.12790E-02
27	2331	0.11655E-02
28	2056	0.10280E-02
29	1892	0.94600E-03
30	1681	0.84050E-03
31	1510	0.75500E-03
32	1405	0.70250E-03
33	1163	0.58150E-03
34	1009	0.50450E-03
35	883	0.44150E-03
36	807	0.40350E-03
37	673	0.33650E-03
38	610	0.30500E-03
39	525	0.26250E-03
40	519	0.25950E-03
41	410	0.20500E-03
42	347	0.17350E-03
43	336	0.16800E-03
44	303	0.15150E-03
45	268	0.13400E-03
46	245	0.12250E-03
47	182	0.91000E-04

48	177	0.88500E-04
49	171	0.85500E-04
50	140	0.70000E-04
51	128	0.64000E-04
52	115	0.57500E-04
53	98	0.49000E-04
54	91	0.45500E-04
55	77	0.38500E-04
56	74	0.37000E-04
57	78	0.39000E-04
58	53	0.26500E-04
59	50	0.25000E-04
60	51	0.25500E-04
61	50	0.25000E-04
62	41	0.20500E-04
63	33	0.16500E-04
64	36	0.18000E-04
65	31	0.15500E-04
66	29	0.14500E-04
67	12	0.60000E-05
68	9	0.45000E-05
69	11	0.55000E-05
70	10	0.50000E-05
71	12	0.60000E-05
72	10	0.50000E-05
73	10	0.50000E-05
74	9	0.45000E-05
75	5	0.25000E-05
76	6	0.30000E-05
77	8	0.40000E-05
78	5	0.25000E-05
79	3	0.15000E-05
80	3	0.15000E-05
81	2	0.10000E-05
82	2	0.10000E-05
83	3	0.15000E-05
84	3	0.15000E-05
85	4	0.20000E-05
86	9	0.45000E-05

Taille moyenne de la file = 2.22832

SIMULATION # 2

Longueur de contrainte du code: K = 8
Rapport de signal-a-bruit: Eb/No = 5.5 dB
Nombre de bits d'information = 2000000 bits
Parametres de simulation: Nmax = 128, Seuil=4
Gain de vitesse: G=64 calc./ tps d'interarrivee

Nombre de survivants	Nombre de fois	Probabilite
1	5	0.25000E-05
2	2	0.10000E-05
3	0	0.00000E+00
4	1	0.50000E-06
5	1	0.50000E-06
6	0	0.00000E+00
7	1	0.50000E-06
8	0	0.00000E+00
9	1	0.50000E-06
10	0	0.00000E+00
11	1	0.50000E-06
12	864696	0.43235E+00
13	13815	0.69075E-02
14	59554	0.29777E-01
15	48478	0.24239E-01
16	27539	0.13769E-01
17	28758	0.14379E-01
18	48507	0.24254E-01
19	50875	0.25438E-01
20	8997	0.44985E-02
21	5968	0.29840E-02
22	6645	0.33225E-02
23	71291	0.35645E-01
24	34529	0.17265E-01
25	9406	0.47030E-02
26	10157	0.50785E-02
27	8632	0.43160E-02
28	47708	0.23854E-01
29	35900	0.17950E-01
30	30578	0.15289E-01
31	55195	0.27598E-01
32	9079	0.45395E-02
33	55325	0.27663E-01
34	57615	0.28808E-01
35	56435	0.28218E-01
36	19496	0.97480E-02
37	12994	0.64970E-02
38	13079	0.65395E-02
39	8821	0.44105E-02
40	6423	0.32115E-02
41	7532	0.37660E-02
42	8309	0.41545E-02
43	4856	0.24280E-02
44	5160	0.25800E-02
45	7127	0.35635E-02
46	6452	0.32260E-02
47	10094	0.50470E-02
48	4730	0.23650E-02
49	4946	0.24730E-02
50	7322	0.36610E-02
51	6369	0.31845E-02
52	6991	0.34955E-02
53	4567	0.22835E-02
54	6976	0.34880E-02
55	5203	0.26015E-02

56	4363	0.21815E-02
57	6128	0.30640E-02
58	8276	0.41380E-02
59	9610	0.48050E-02
60	4855	0.24275E-02
61	5760	0.28800E-02
62	3879	0.19395E-02
63	7294	0.36470E-02
64	5006	0.25030E-02
65	5061	0.25305E-02
66	3108	0.15540E-02
67	7908	0.39540E-02
68	5618	0.28090E-02
69	8068	0.40340E-02
70	6979	0.34895E-02
71	6365	0.31825E-02
72	5252	0.26260E-02
73	4484	0.22420E-02
74	4375	0.21875E-02
75	2821	0.14105E-02
76	2022	0.10110E-02
77	2160	0.10800E-02
78	2105	0.10525E-02
79	2928	0.14640E-02
80	1907	0.95350E-03
81	3239	0.16195E-02
82	1965	0.98250E-03
83	1920	0.96000E-03
84	1849	0.92450E-03
85	1763	0.88150E-03
86	1752	0.87600E-03
87	1652	0.82600E-03
88	1644	0.82200E-03
89	1598	0.79900E-03
90	1638	0.81900E-03
91	1810	0.90500E-03
92	1817	0.90850E-03
93	1890	0.94500E-03
94	1956	0.97800E-03
95	1698	0.84900E-03
96	1933	0.96650E-03
97	1330	0.66500E-03
98	1424	0.71200E-03
99	1778	0.88900E-03
100	1532	0.76600E-03
101	1623	0.81150E-03
102	1549	0.77450E-03
103	1673	0.83650E-03
104	1496	0.74800E-03
105	1553	0.77650E-03
106	1299	0.64950E-03
107	1456	0.72800E-03
108	1357	0.67850E-03
109	1295	0.64750E-03
110	1574	0.78700E-03
111	1364	0.68200E-03
112	1125	0.56250E-03
113	1149	0.57450E-03
114	1120	0.56000E-03
115	1027	0.51350E-03
116	922	0.46100E-03
117	992	0.49600E-03
118	864	0.43200E-03
119	837	0.41850E-03
120	822	0.41100E-03
121	809	0.40450E-03

122	957	0.47850E-03
123	958	0.47900E-03
124	1125	0.56250E-03
125	953	0.47650E-03
126	1280	0.64000E-03
127	802	0.40100E-03
128	2365	0.11825E-02

Facteur d'utilisation rho = 0.406233
Nombre moyen de survivants = 25.9989
Variance = 445.686
Deviation standard = 21.1113

Taille max. de file d'attente	Nombre de fois	Probabilite
0	1750447	0.87522E+00
1	104798	0.52399E-01
2	33209	0.16604E-01
3	25531	0.12765E-01
4	20389	0.10194E-01
5	13084	0.65420E-02
6	9605	0.48025E-02
7	8369	0.41845E-02
8	7100	0.35500E-02
9	5600	0.28000E-02
10	4081	0.20405E-02
11	3448	0.17240E-02
12	2710	0.13550E-02
13	2131	0.10655E-02
14	1697	0.84850E-03
15	1409	0.70450E-03
16	1154	0.57700E-03
17	999	0.49950E-03
18	783	0.39150E-03
19	675	0.33750E-03
20	582	0.29100E-03
21	447	0.22350E-03
22	337	0.16850E-03
23	260	0.13000E-03
24	226	0.11300E-03
25	172	0.86000E-04
26	133	0.66500E-04
27	129	0.64500E-04
28	101	0.50500E-04
29	76	0.38000E-04
30	62	0.31000E-04
31	55	0.27500E-04
32	41	0.20500E-04
33	23	0.11500E-04
34	14	0.70000E-05
35	19	0.95000E-05
36	15	0.75000E-05
37	9	0.45000E-05
38	18	0.90000E-05
39	14	0.70000E-05
40	12	0.60000E-05
41	11	0.55000E-05
42	5	0.25000E-05
43	7	0.35000E-05
44	9	0.45000E-05
45	6	0.30000E-05
46	5	0.25000E-05

Taille moyenne de la file = 0.465876

SIMULATION # 3
Longueur de contrainte du code: K = 8
Rapport de signal-a-bruit: Eb/No = 6.0 dB
Nombre de bits d'information = 2000000 bits
Parametres de simulation: Nmax = 128, Seuil=4
Gain de vitesse: G=64 calc./ tps d'interarrivee

Nombre de survivants	Nombre de fois	Probabilite
1	5	0.25000E-05
2	2	0.10000E-05
3	0	0.00000E+00
4	1	0.50000E-06
5	1	0.50000E-06
6	0	0.00000E+00
7	1	0.50000E-06
8	0	0.00000E+00
9	1	0.50000E-06
10	0	0.00000E+00
11	1	0.50000E-06
12	1057852	0.52893E+00
13	10535	0.52675E-02
14	54820	0.27410E-01
15	46257	0.23129E-01
16	25654	0.12827E-01
17	26328	0.13164E-01
18	46520	0.23260E-01
19	48499	0.24249E-01
20	6518	0.32590E-02
21	4159	0.20795E-02
22	4895	0.24475E-02
23	68866	0.34433E-01
24	30837	0.15418E-01
25	6791	0.33955E-02
26	7376	0.36880E-02
27	6271	0.31355E-02
28	45764	0.22882E-01
29	32006	0.16003E-01
30	27688	0.13844E-01
31	51262	0.25631E-01
32	6296	0.31480E-02
33	51665	0.25833E-01
34	53170	0.26585E-01
35	52397	0.26199E-01
36	14540	0.72700E-02
37	9455	0.47275E-02
38	9577	0.47885E-02
39	6060	0.30300E-02
40	4305	0.21525E-02
41	5254	0.26270E-02
42	5788	0.28940E-02
43	3189	0.15945E-02
44	3339	0.16695E-02
45	5052	0.25260E-02
46	4427	0.22135E-02
47	7209	0.36045E-02
48	3111	0.15555E-02
49	3230	0.16150E-02
50	5083	0.25415E-02
51	4389	0.21945E-02
52	4809	0.24045E-02
53	2909	0.14545E-02
54	4869	0.24345E-02
55	3398	0.16990E-02
56	2740	0.13700E-02

57	4244	0.21220E-02
58	5920	0.29600E-02
59	6844	0.34220E-02
60	3145	0.15725E-02
61	3918	0.19590E-02
62	2391	0.11955E-02
63	4952	0.24760E-02
64	3287	0.16435E-02
65	3324	0.16620E-02
66	1852	0.92600E-03
67	5583	0.27915E-02
68	3832	0.19160E-02
69	5682	0.28410E-02
70	4716	0.23580E-02
71	4155	0.20775E-02
72	3434	0.17170E-02
73	2889	0.14445E-02
74	2836	0.14180E-02
75	1661	0.83050E-03
76	1105	0.55250E-03
77	1172	0.58600E-03
78	1117	0.55850E-03
79	1717	0.85850E-03
80	946	0.47300E-03
81	2033	0.10165E-02
82	1100	0.55000E-03
83	1063	0.53150E-03
84	1009	0.50450E-03
85	935	0.46750E-03
86	867	0.43350E-03
87	893	0.44650E-03
88	867	0.43350E-03
89	826	0.41300E-03
90	843	0.42150E-03
91	986	0.49300E-03
92	952	0.47600E-03
93	1034	0.51700E-03
94	973	0.48650E-03
95	887	0.44350E-03
96	994	0.49700E-03
97	674	0.33700E-03
98	734	0.36700E-03
99	874	0.43700E-03
100	824	0.41200E-03
101	802	0.40100E-03
102	736	0.36800E-03
103	815	0.40750E-03
104	705	0.35250E-03
105	790	0.39500E-03
106	623	0.31150E-03
107	698	0.34900E-03
108	629	0.31450E-03
109	633	0.31650E-03
110	773	0.38650E-03
111	637	0.31850E-03
112	500	0.25000E-03
113	515	0.25750E-03
114	472	0.23600E-03
115	449	0.22450E-03
116	398	0.19900E-03
117	424	0.21200E-03
118	322	0.16100E-03
119	306	0.15300E-03
120	315	0.15750E-03
121	319	0.15950E-03
122	367	0.18350E-03

123	337	0.16850E-03
124	390	0.19500E-03
125	347	0.17350E-03
126	438	0.21900E-03
127	273	0.13650E-03
128	733	0.36650E-03

Facteur d'utilisation rho = 0.343396
Nombre moyen de survivants = 21.9773
Variance = 289.219
Deviation standard = 17.0064

Taille max. de file d'attente	Nombre de fois	Probabilite
0	1863490	0.93174E+00
1	69953	0.34976E-01
2	18169	0.90845E-02
3	13563	0.67815E-02
4	10583	0.52915E-02
5	5966	0.29830E-02
6	4027	0.20135E-02
7	3427	0.17135E-02
8	2686	0.13430E-02
9	2055	0.10275E-02
10	1395	0.69750E-03
11	1111	0.55550E-03
12	856	0.42800E-03
13	641	0.32050E-03
14	419	0.20950E-03
15	378	0.18900E-03
16	311	0.15550E-03
17	245	0.12250E-03
18	175	0.87500E-04
19	129	0.64500E-04
20	100	0.50000E-04
21	71	0.35500E-04
22	54	0.27000E-04
23	36	0.18000E-04
24	23	0.11500E-04
25	24	0.12000E-04
26	24	0.12000E-04
27	21	0.10500E-04
28	23	0.11500E-04
29	19	0.95000E-05
30	10	0.50000E-05
31	12	0.60000E-05
32	2	0.10000E-05
33	2	0.10000E-05
34	3	0.15000E-05
35	4	0.20000E-05

Taille moyenne de la file = 0.194209