# Décodage pondéré de codes linéaires en bloc : Application au code BCH étendu (32,21)

Fabrice Bellard et Vincent Huitric

4 avril 1997

## 1 Introduction

### 1.1 Généralités

Le décodage pondéré (*soft decoding*) apporte un gain non négligeable face au décodage ferme (*hard decoding*) : le gain asymptotique est de 2 dB sur le canal gaussien. Nous nous intéressons ici au décodage pondéré des codes linéaires en blocs aux faibles rapports signal sur bruit (SNR), et plus particulièrement au code BCH étendu par parité (32,21) utilisé dans la norme ERMES de radiomessagerie. Aux faibles SNR, les estimations asymptotiques sont de peu d'utilité, c'est pourquoi nous baserons toute notre étude sur la simulation. Toutes nos simulations sont effectuées sur le canal gaussien. Sur d'autres types de canaux, par exemple le canal de Rayleigh, nous nous attendons à de meilleurs gains de codage. Nous nous intéresserons particulièrement à la complexité des différentes méthodes présentées, dans l'espoir d'en réaliser une implémentation effective.

### 1.2 Mise en perspective

Bien que les codes convolutionels soient beaucoup plus efficaces pour le décodage pondéré, un grand nombre de contributions ont été faites pour les codes en blocs. L'algorithme de Chase [3] est un des premiers exemples de décodage pondéré sous-optimal. Il est basé sur la modification d'un nombre prédéterminé de symboles parmi les moins vraissemblables puis une recherche algébrique du mot de code le plus proche. En [2], les symboles les moins vraissemblables sont déterminés par seuillage.

L'utilisation d'un treillis basé sur le syndrome a été suggéré par Wolf [4]. Le temps de décodage pour un code linéaire de paramètres $(n, k)$ devient exponentiel en $n-k$ au lieu de $k$. Un décodage sous-optimal plus rapide en sélectionnant les états les plus vraissemblables est décrit en [5]. La structure algébrique du code peut être utilisée pour simplifier le treillis [6].

Pour les codes de très faibles ratios $R = k/n$, la transformation de Hadamard accélère la recherche exhaustive [7]. Pour certains codes, une approche basée sur l'analyse du syndrome est très efficace [8].

Un algorithme étonnament simple basé sur le tri selon la vraissemblance des symboles est décrit en [1]. Il est sous-optimal, mais donne pour des codes linéaires où $n \leq 64$ des performances quasi optimales pour un temps de calcul raisonnable.

Nous avons implémenté les algorithmes expliqués en [4], [5], et [1] qui sont satisfaisants dans tous les cas pratiques.

# 2 Description de quelques algorithmes de décodage

## 2.1 Décodage ferme

Le décodage ferme est le décodage le plus simple. Il s'agit d'estimer la valeur de chacun des bits séparément avec un détecteur à seuil puis d'appliquer un algorithme de décodage algébrique sur le mot estimé. L'algorithme en question peut être un algorithme de type Peterson-Gorenstein-Zierler ou Berlekamp-Massey dans le cas d'un code BCH. Dans le cas d'un code linéaire quelconque, on peut utiliser une table de correspondance syndrome-erreur.

Cette technique est bien sûr utilisée comme référence car le but de l'étude est aussi de prouver la supériorité du décodage pondéré.

## 2.2 Algorithme de Viterbi

Lorsqu'on transmet une séquence de $-1$ et de $1$ de longueur $n$ dans un canal, ce qui est reçue est une séquence de longueur $n$ de réels $(r_i)_{1,\dots,n}$. Lorsque ce canal est gaussien, on définit la distance de $r$ à un mot du code par

$$dist(r, c) = \sum_{i=1}^{n} |r_i - c_i|^2 \ .$$

Le but du décodage pondéré est d'optimiser la distance du mot reçu au mot du code retenu ; on espère ainsi tirer profit de toutes les informations disponibles.

On remarque d'abord que $dist(r, c) = \sum r_i^2 + \sum c_i^2 - 2 \sum r_i c_i$ . Or la norme de $r$ ne dépend pas de $c$ et celle de $c$ vaut identiquement $n$. Il est donc équivalent de minimiser $dist(r, c)$ sur $c$ et de maximiser sur $c$ la corrélation

$$corr(r, c) = \sum_{i=1}^{n} c_i r_i$$

L'algorithme le plus primitif voudrait que l'on évalue $corr(r, c)$ pour tous les mots du code afin de trouver celui qui atteint le minimum. Cette méthode est rapidement inutilisable car pour un code $(n, k)$ cela correspondrait à $2^k$ évaluations. L'idée de l'algorithme de Viterbi est de dire qu'en fait on cherche un chemin optimal sur un graphe et d'en diminuer le nombre d'arêtes.

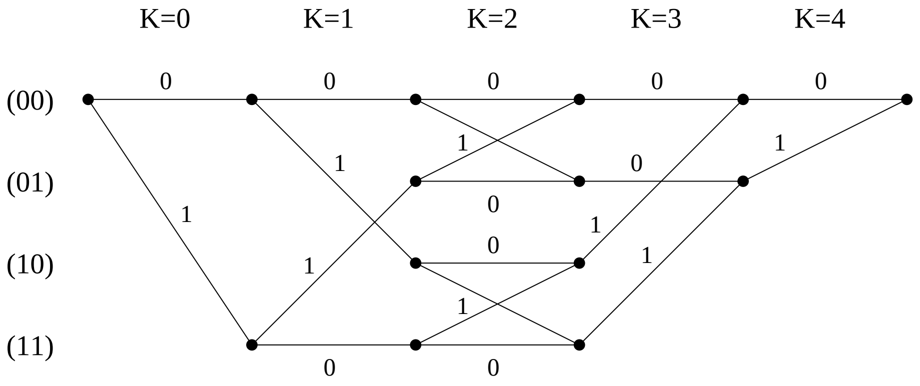
On construit donc un graphe orienté ayant pour source A (aucun bit de $c$ n'est encore déterminé) et pour puit B (tous les bits sont déterminés) tous les chemins ayant la même longueur $n$ puisque chaque étape correspond à la détermination d'un bit de $c$ de plus. L'ensemble des mots du code correspond donc à l'ensemble des chemins de A vers B. Ainsi de tout noeud du graphe ne peut partir qu'un ou deux arcs correspondant à un choix impératif ou libre du bit de $c$ en question. Un tel graphe est appelé un treillis. Si l'on affecte comme valeur aux arcs, soit $r_i$ s'il correspond au choix $c_i = 1$ soit $-r_i$ dans le cas contraire, la détermination du mot du code $c$ optimisant $dist(r, c)$ est équivalente à la détermination du chemin de poids maximum de A vers B.

Pour intégrer la contrainte de l'appartenance de $c$ au code dans le treillis, on le construit de la manière suivante : à chaque étape les noeuds sont indexés par le syndrome partiel, c'est-à-dire par le syndrome du code choisi par le chemin venant de A complété par des zéros. On remarque que cette construction est cohérente puisque si on "fusionne" deux chemins en les faisant passer par le même noeud, c'est que leur comportements futurs (en matière de syndrome) sont identiques. Et cela en vertue du caratère linéaire du syndrome. On élimine ensuite tous les chemins qui finissent par un syndrome non nul qui correspondent à des mots qui n'appartienent pas au code et on a déterminé un treillis correspondant au code. La complexité de la résolution du problème du chemin optimal est évaluée à $N = (2k - n + 2)2^{n-k+1}$. Cette complexité est obtenue en prenant en compte trois phases : une phase de croissance à partir de A, une phase de maintient à $2^{n-k}$ noeuds et une phase de décroissance jusqu'à B.

Par exemple, pour la matrice de parité

$$H = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

tirée de [4], on obtient le treillis suivant :




### 2.2.1 Viterbi tronqué

L'algorithme de Viterbi, puisque exact, présente l'inconvénient d'être encore raisonnablement lent. Le problème devient particulièrement crucial lorsque $n$ devient grand. Il est naturel de tranformer l'algorithme de Viterbi en un algorithme heuristique afin de gagner du temps. La méthode la plus simple qui vienne à l'idée est de limiter le nombre de noeuds à chaque étape et de considérer comme impossibles ceux qui dépasseraient ce quota [5]. La sélection ne se fait pas complètement au hasard puisqu'on ne retient que les noeuds offrant la plus grande valeur de corrélation partielle.

### 2.2.2 Mélange décodage ferme et Viterbi

Une autre méthode consiste en suposer qu'un certain nombre de bits sont surs et de n'appliquer l'algorithme de Viterbi qu'aux bits non retenus. Le critère de sélection étant bien entendu $|r_i| > Seuil$.

### 2.3 Effacement des $K$ bits les moins surs

Cette méthode est tres simple, il s'agit au début de procéder comme pour la méthode précédente. Pour un paramètre $K$, on sélectionne $n-K$ bits *a priori* surs et $K$ bits variables. Puis on fait varier les $K$ bits non retenus et pour chaque combinaison, on opère un décodage ferme sur le bloc. A la fin on retient le mot ayant donné la meilleure corrélation. L'étude complète de la méthode se trouve en [3] ou en [2] avec une petite nuance.

### 2.4 Tri, systématisation, et décodage ferme partiel

Cette méthode est un peu plus technique mais donne des résultats exceptionnels, elle est décrite dans [1]. On commence par sélectionner $k$ bits de telle sorte que les $k$ colonnes correspondantes de la matrice génératrice soient indépendantes et aussi sures que possible. En fait cette opération est faite en deux étapes, la première étant d'ordonner les bits par certitude décroissante, et la deuxième de les réordonner afin de rendre les $k$ premières colonnes indépendantes. En pratique, ce deuxième réordonnancement modifie peu le tri initial. Ici $k$ est bien la dimension du code et pas un paramètre de la méthode.

Une fois ces $k$ bits placés en tête, on détermine une nouvelle matrice génératrice systématique $G'$. On fixe par décision ferme les $k$ premiers bits $(j_i)_{1\ldots k}$ correspondant aux $k$ premières colonnes de $G'$. Ensuite on fait parcourir à $(e_i)_{1\ldots k}$ l'ensemble des vecteurs de poids inférieur à $l$ et on calcule le mot du code $c$ correspondant à $j+e$ soit $c=(j+e)G'$. Finalement on retient le mot $c$ donnant la meilleure correlation.

$$G' = \left( \begin{array}{c|c} I_k & P' \end{array} \right)$$

$$\left( \begin{array}{ccc} c_1, & \ldots, & c_n \end{array} \right) = \left( \begin{array}{ccc} j_1+e_1, & \ldots, & j_k+e_k \end{array} \right) \left( \begin{array}{c|c} I_k & P' \end{array} \right)$$

$$corr(r, c_{sub-opt}) = \max_{poids(e')\leq l} corr(r, c')$$

Du point de vue de la complexité, le tri est en $O(n \ln n)$, la systématisation qui consiste à exécuter un pivot de Gauss est en $O(nk)$ et enfin l'introduction des erreurs de poids $\leq l$ prend $O(k^l(n-k))$ opérations.

## 3 Résultats

### 3.1 Méthodologie

Les simulations ont été effectuées sur le canal gaussien. On a mesuré la probabilité d'erreur $P_e$ au niveau des bits pour un SNR $\frac{E_b}{N_0}$ prenant en compte l'énergie supplémentaire introduite par la redondance du code. Les codes utilisés ont des rendements proches de $\frac{1}{2}$ ou $\frac{2}{3}$, d'où les limites théoriques de transmission sans erreur de 0 dB et 0.6 dB respectivement, d'après le second théorème de Shannon. On notera que le rendement du code détermine directement l'encombrement spectral de la transmission.

Nous avons transmis des messages binaires générés aléatoirement. On simule le bruit avec un générateur gaussien décrit dans [9]. Pour garantir une précision relative de $\epsilon$ sur $P_e$, on attend la détection de $\frac{1}{\epsilon^2}$ erreurs. Nous avons choisi une précision de 10 % pour garantir un temps de calcul acceptable. Les courbes n'ont volontairement pas été lissées. Pour une étude asymptotique des méthodes, on se reportera aux articles cités. On a inclus sur chaque courbe les performances sans codage (modulation MDP-2 ou *BPSK*).

Lors du décodage pondéré nous avons quantifié les valeurs reçues sur une dizaine de bits. Nous avons vérifié, et c'est sans surprise, qu'une quantification peu précise donne des résultats très proches (trois ou quatre bits auraient suffi en fait).

Les algorithmes ont été optimisés ce qui nous permet de donner des estimations de vitesse. Elles restent très approximatives et ne présument en rien du temps requis par une implémentation *hardware* où le paraléllisme est plus important. Nous avons normalisé les temps de décodage par rapport au temps pris pour le décodage algébrique.

## 3.2 Code BCH (32,21)

### 3.2.1 Viterbi

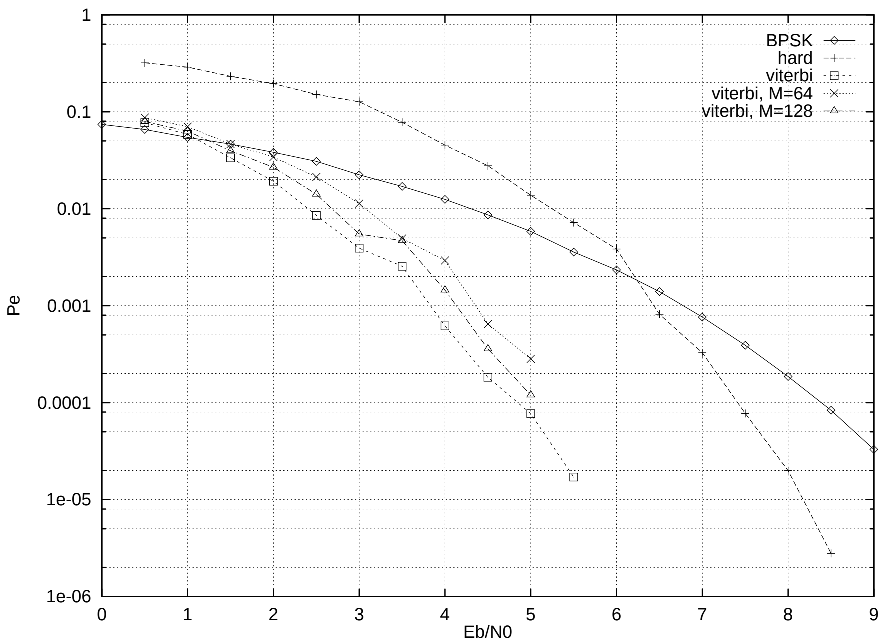



FIG. 1 – *Décodage pondéré du code BCH (32,21) - Algorithme de Viterbi*

L'algorithme de Viterbi est recommendé pour ce code car $n-k < k$. La figure 1 résume nos résultats. On obtient 3413 opérations par bit qu'on peut réduire à 1094 en diminuant la complexité du treillis par des méthodes algébriques [6]. Ce temps de calcul reste important. La méthode de base, qui se prête bien à l'intégration du fait de sa structure répétitive, reste préférable. Nous obtenons une vitesse normalisée de $\frac{1}{40}$ avec notre implémentation. Avec un algorithme de Viterbi tronqué à $M = 128$ états, on obtient un décodage sous-optimal (perte de 0.25 dB pour $P_e = 10^{-4}$) un peu plus rapide mais dans un faible rapport du fait de la logique supplémentaire nécessaire pour le tri des états.

Le gain du décodage pondéré est de 2.5 dB à $P_e = 10^{-4}$ face au décodage ferme, ce qui n'est pas négligeable.

### 3.2.2 Effacement

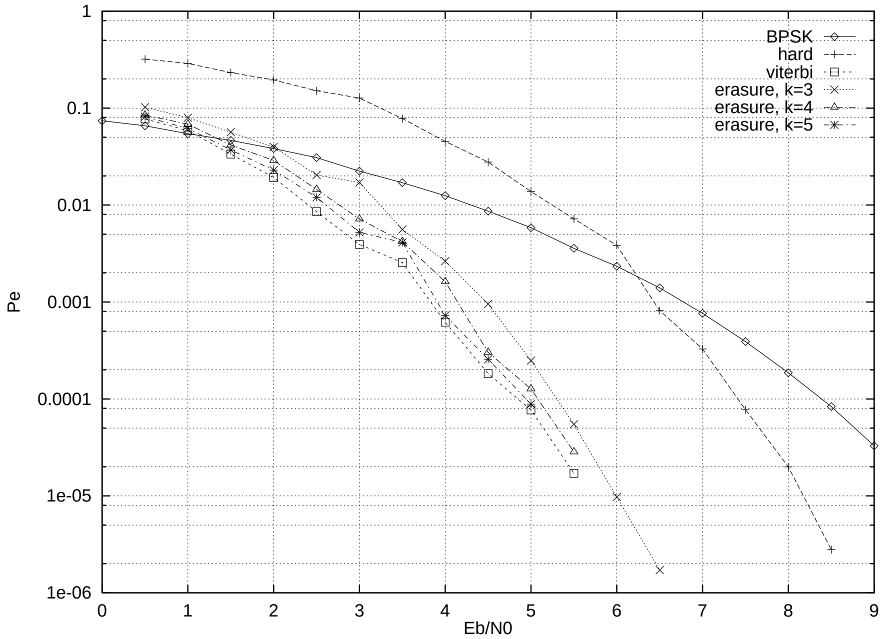



FIG. 2 – *Décodage pondéré du code BCH (32,21) - Algorithme d'effacement*

La figure 2 résume les résultats. La modification des $k = 4$ derniers bits donne un décodage quasi-optimal (perte de 0.25 dB pour $P_e = 10^{-4}$). La complexité est très faible (16 décodages algébriques sont nécessaires, plus quelques dizaines d'additions). La vitesse normalisée de notre implémentation est de 0.4. C'est la méthode de choix si on dispose déjà d'un décodeur algébrique.

### 3.2.3 Tri et systématisation

L'ordre 0 donne déjà un décodage meilleur que le décodage ferme à faible SNR. L'ordre 1, qui nécessite à peine plus d'opérations, donne un décodage très proche de l'optimal (voir la figure 3). La vitesse normalisée est de 0.2. On notera que l'implémentation *hardware* d'un tel algorithme est plus complexe que les précédents à cause de sa structure peu régulière, par exemple pour le pivot de Gauss. En revanche, une implémentation *software* efficace pour un débit modéré est tout-à-fait envisageable.

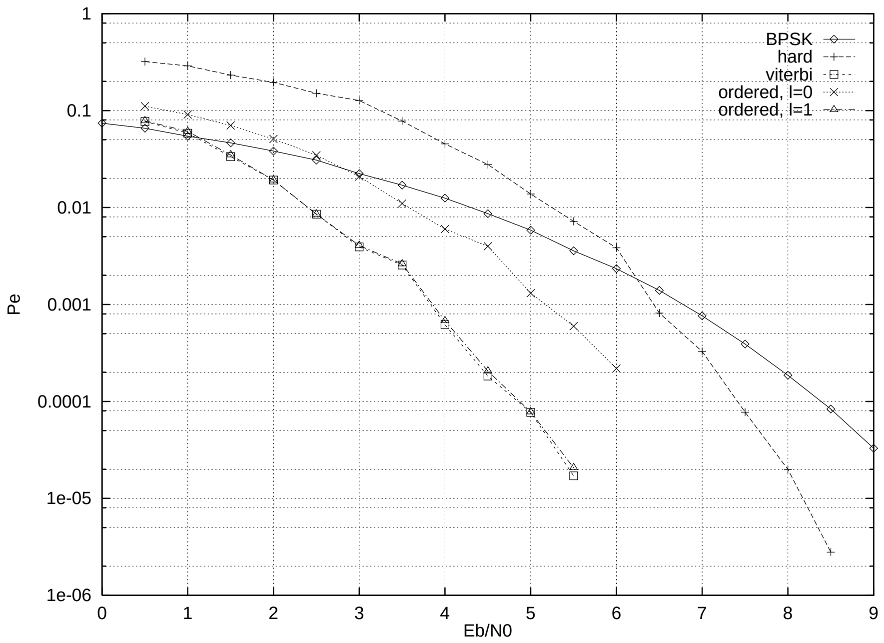



FIG. 3 – *Décodage pondéré du code BCH (32,21) - Algorithme de tri et systématisation*

## 3.3 Code de Golay étendu (24,12)

La figure 4 donne les résultats de nos simulations sur le code de Golay étendu (24,12) qui est décrit dans de nombreux articles. L'algorithme par effacement avec $k = 3$ est proche de l'optimal, et avec $k = 4$ on s'en rapproche encore. Si on dispose d'un algorithme algébrique efficace, par exemple avec une table syndrome-erreur, la vitesse est très importante. Le décodage par tri et systématisation donne des performances indiscernables de l'optimal.

## 3.4 Code BCH (64,36)

Pour de meilleurs performances, au prix d'un temps de calcul plus important, on peut utiliser des codes linéaires plus longs. La figure 5 donne les résultats sur le code BCH (64,36). Le décodage par tri et systématisation donne des performances optimales à l'ordre 2.

## 3.5 Code BCH (128,64)

Le code BCH (128,64) est présenté (figure 6) pour confirmer les résultats de [1]. Les temps de calcul commencent à être prohibitifs pour les ordres 3 ou 4.

## 3.6 Codes linéaires aléatoires (64,32)

On notera qu'à faible SNR la distance minimale du code a peu d'incidence sur le gain de codage. Comme l'algorithme par tri et systématisation fonctionne

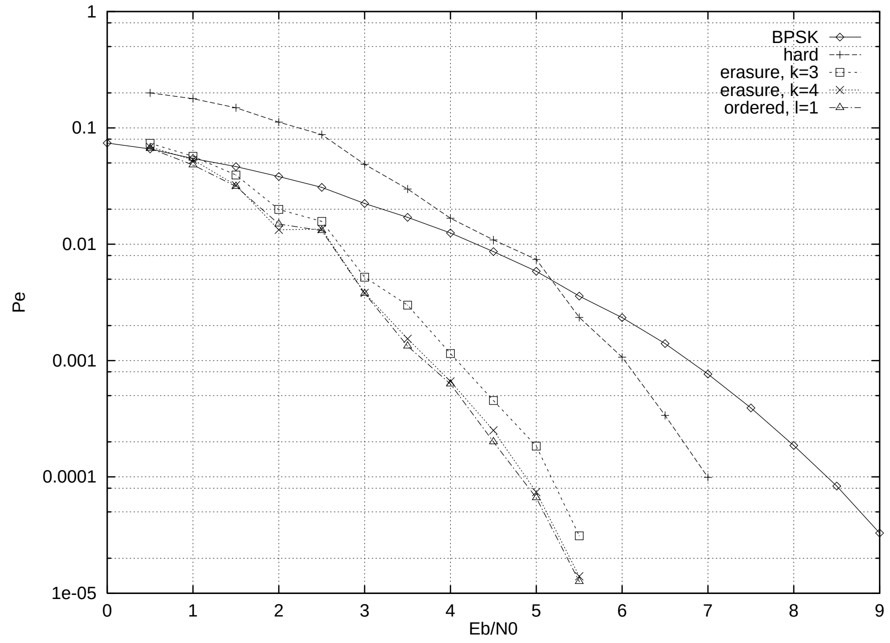



Fig. 4 – *Décodage pondéré du code de Golay étendu (24,12)*

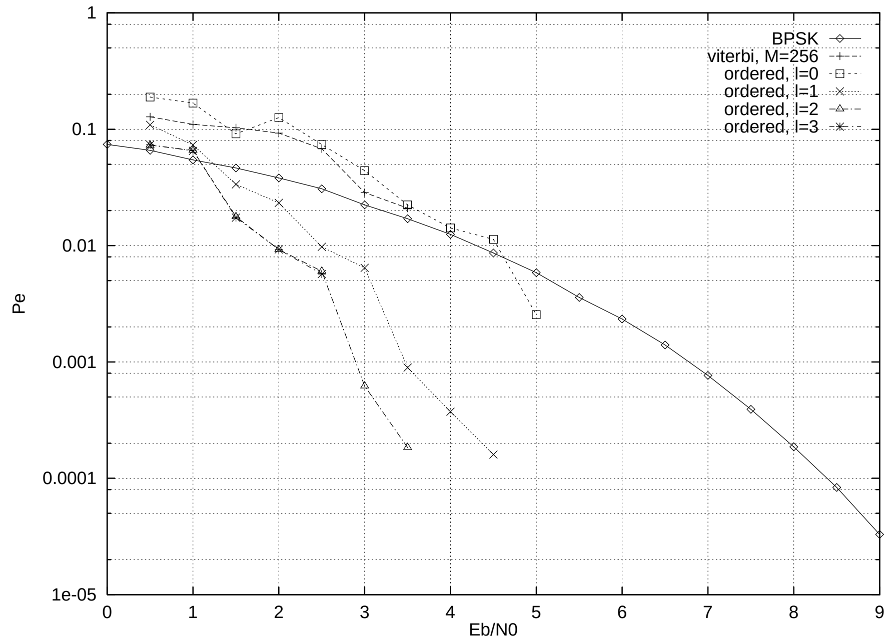



Fig. 5 – *Décodage pondéré du code BCH (64,36) - Algorithme de tri et systématisation*

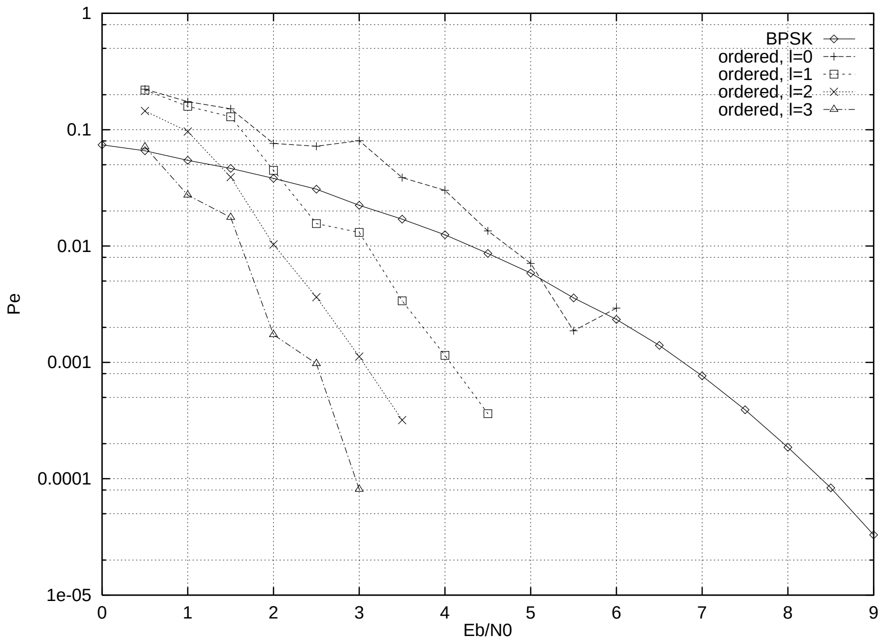



FIG. 6 – *Décodage pondéré du code BCH (128,64) - Algorithme de tri et systématisation*

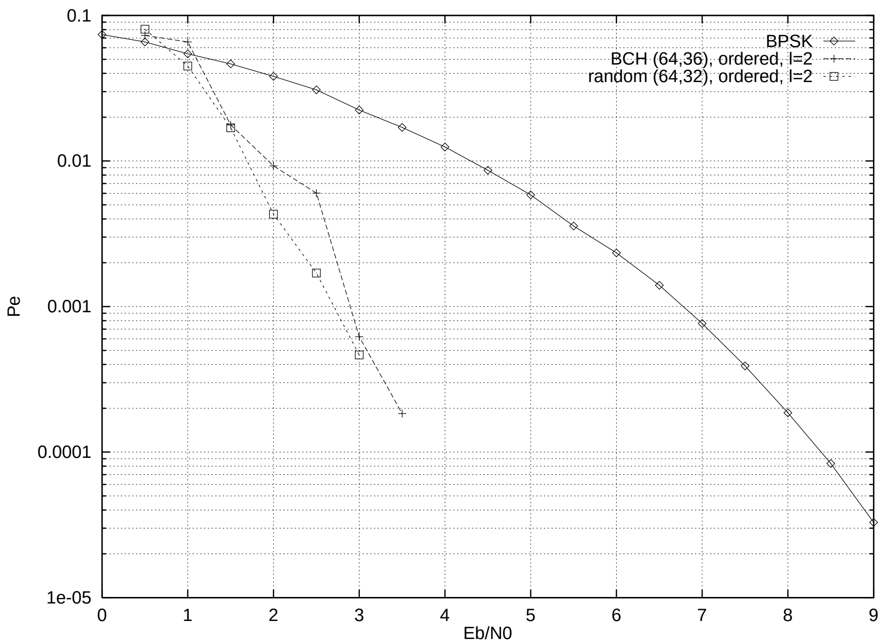



FIG. 7 – *Décodage pondéré d'un code linéaire aléatoire (64,32) - Algorithme de tri et systématisation*

sur tout code linéaire, nous l'avons appliqué à un code où la matrice G, systématique, a été générée de façon aléatoire (figure 7). Le gain de codage est comparable à celui du code BCH (64,36) considéré (les ratios $R$ étant quasiment égaux).

# 4 Conclusion

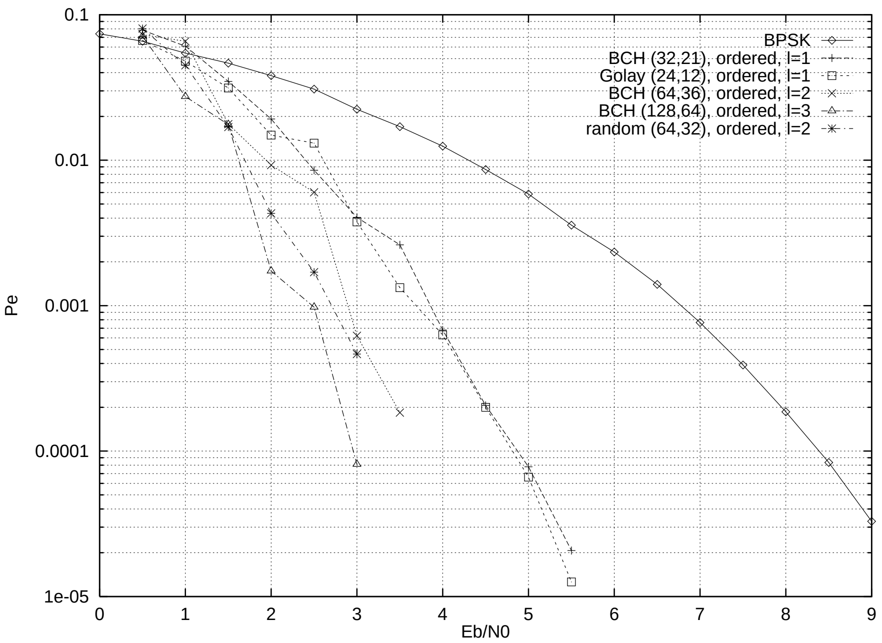



FIG. 8 – *Comparaison de quelques codes linéaires en décodage pondéré*

Nous considérons que le problème du décodage pondéré du code BCH (32,21) est en pratique résolu, ainsi que celui des codes linéaires de longueur comparable. On appliquera avec succès les algorithmes sous-optimaux fonctionnant par effacement ou tri et systématisation. La figure 8 résume les performances possibles par décodage pondéré de quelques codes. Nous n'avons pas étudié les performances des décodeurs aux SNR importants, mais les études théoriques montrent qu'ils se comportent correctement. De plus, pour les applications considérées, on se contente d'une probabilité d'erreur moyenne ($P_e = 10^{-4}$ par exemple) et le système fonctionne en général avec un SNR faible. Il serait intéressant de faire des simulations sur un canal différent du canal gaussien pour vérifier la faible sensiblité des algorithmes sous-optimaux à la métrique choisie. Mais les codes convolutionels aux structures de treillis simples sont encore le meilleur choix quand on peut choisir le code à employer.

# Références


[1] Marc P. C. FOSSORIER and Shu LIN, "Soft-Decision Decoding of Linear

Block Codes Based on Ordered Statistics ", *IEEE Trans. Inform. Theory* , vol. IT-41, no. 5, Sept. 1995.

[2] Hatsukazu TANAKA and Koji KAKIGAHARA, "Simplified Correlation Decoding by Selecting Possible Codewords Using Erasure Information", *IEEE Trans. Inform. Theory* , vol. IT-29, no. 5, Sept. 1983.

[3] D. CHASE, "A class of algorithms for decoding block codes with channel measurement information", *IEEE Trans. Inform. Theory* , vol. IT-18, pp. 170-182, Jan. 1972.

[4] Jack K. WOLF, "Efficient Maximum Likelihood Decoding of Linear Block Codes Using a Trellis", *IEEE Trans. Inform. Theory* , vol. IT-24, no. 1, Jan. 1978.

[5] K.R. MATIS and James W. MODESTINO, "Reduced-Search Soft-Decision Trellis Decoding of Linear Block Codes", *IEEE Trans. Inform. Theory* , vol. IT-28, no. 2, March 1982.

[6] Alexander VARDY and Yair BE'ERY, "Maximum-Likelihood Soft Decision of BCH Codes", *IEEE Trans. Inform. Theory* , vol. IT-40, no. 2, March 1993.

[7] Yair BE'ERY and Jakov SNYDERS, "Optimal Soft Decision Block Decoders Based on Fast Hadamard Transform", *IEEE Trans. Inform. Theory* , vol. IT-32, no. 3, May 1986.

[8] Jakov SNYDERS and Yair BE'ERY, "Maximum Likelihood Soft Decoding of Binary Block Codes and Decoders for the Golay Codes", *IEEE Trans. Inform. Theory* , vol. IT-35, no. 5, Sept. 1989.

[9] D. E. KNUTH, *The art of computer programming*, vol. 2, Addison-Wesley, 1973.