Etude du décodage des codes de Reed-Muller et application à la cryptographie.

Bassem Sakkour

THÈSE

présentée à

L'ECOLE POLYTECHNIQUE

pour obtenir le titre de

DOCTEUR EN SCIENCES

Spécialité :

Mathématiques et Informatique

par

BASSEM SAKKOUR

Sujet de la thèse :

Étude et amélioration du décodage des codes de Reed-Muller d'ordre deux.

Thèse Soutenue le 06 Avril 2007 devant le jury composé de :

Président	:	François Morain
Responsable	:	Pascale Charpin
Directeur	:	Pierre Loidreau
Rapporteurs	:	Sami Harari
		Grigory Kabatianskiy
Examinateurs	:	Thierry Berger
		Nicolas Sendrier

École Doctorale
de l'École Polytechnique
91128 Palaiseau

École Nationale Supérieure
de Techniques Avancées
32 Bd Victor, 75015 Paris

Remerciements

Ma plus grande reconnaissance revient à l'équipe de recherche du laboratoire UMA de l'ENSTA au sein duquel je me suis initié à la recherche.

Je tiens à remercier mon directeur de thèse Pierre Loidreau, tout d'abord pour m'avoir proposé le stage initial de DEA qui m'a permis d'effectuer cette thèse, ensuite, pour ses remarques constructives, ses conseils qui m'ont aidé depuis mes premiers pas dans la recherche, et toutes les discussions que nous avons eues tout au long de cette recherche. Je remercie Pascale Charpin d'avoir accepté d'être ma responsable de thèse et de son soutien.

Je remercie mes rapporteurs Grigory Kabatyanskiy pour toutes les discussions enrichissantes que j'ai eu le plaisir d'avoir avec lui, et Sami Harari pour ses conseils et son accueil à Toulon. Je remercie vivement Nicolas Sendrier, Thierry Berger et François Morain, mon professeur de DEA, d'être les examinateurs de ma thèse.

Les discussions enrichissantes que j'ai eu le plaisir d'avoir avec Jean-Pierre Tillich ont été déterminantes dans ma formation, je lui en suis très reconnaissant.

C'est avec joie que je vous exprime ma reconnaissance à tous les thésards et membres du Laboratoire UMA-ENSTA. Fabrice, François vous étiez de merveilleux amis et collègues de bureau. Guillaume et Colin d'excellent amis. Je remercie aussi tous les membres du Laboratoire que je n'ai pas nommé explicitement. Je remercie tous les membres du projet CODES de l'INRIA, permanents et thésards que j'ai côtoyé pendant ces trois ans.

Mes remerciements vont particulièrement à Manal, qui a quitté son travail pour me suivre ici en France, sans son soutien je n'aurais pas osé prendre ce chemin. Un grand merci à ma famille au grand complet et plus particulièrement à mon grand frère Nassar.

Enfin, j'ai une pensée émue pour mon père qui aurait souhaité voir ce moment, je lui dédie du fond du coeur ce mémoire.

Table des matières

Introduction

Ce manuscrit est le fruit de trois années de recherche au sein du Laboratoire de Mathématiques Appliquées de l'ENSTA sur le thème de recherche "Étude du décodage des codes de Reed-Muller". L'objectif de ma recherche est d'étudier les algorithmes de décodage des codes de Reed-Muller d'ordre-2 existants, notamment l'algorithme de décodage dû à Sidel'nikov et Pershakov qui utilise les propriétés structurelles des codes de Reed-Muller pour décoder significativement plus loin que la capacité de correction. L'idée est de transformer le problème du décodage d'un mot dans un code de Reed-Muller d'ordre deux en plusieurs problèmes de décodage de mots dans des codes de Reed-Muller d'ordre un.

Si pour les codes de Reed-Muller du premier ordre il existe des algorithmes parvenant au décodage par maximum de vraisemblance en complexité polylogarithmique en la longueur du code, en ce qui concerne les codes d'ordre deux, les meilleurs algorithmes de décodage connus sont loin d'approcher la borne du décodage à maximum de vraisemblance dont un équivalent asymptotique a été donné par Helleseth, Kløve et Levenshtein en 2003 [53]. L'avènement d'un tel algorithme constituerait un progrès majeur. En effet, cela ouvrirait des perspectives intéressantes dans le cadre de l'utilisation de ces codes dans des applications cryptographiques. Par exemple, on pourrait songer à tenter de décoder des combinaisons linéaires de sortie de systèmes de chiffrement à clé secrète et ainsi en trouver de bonnes approximations quadratiques, comme on le fait déjà dans le cas d'approximations linéaires. Du point de vue de la cryptographie à clé publique, cette famille se retrouve également au cœur d'un système de chiffrement à clé publique similaire à celui de McEliece.

Ce document est structuré en six chapitres. Dans le premier chapitre, nous présentons quelques généralités concernant la théorie des codes correcteurs d'erreurs, nous décrivons la problématique liée à la transmission de l'information à travers un canal bruité et à la corrections d'erreurs. Nous introduisons le groupe d'automorphisme d'un code linéaire, nous parlerons ensuite du problème de décodage en définissant les types de décodage et la borne de décodage asymptotique d'un algorithme de décodage.

Le second chapitre est consacré à l'étude des codes de Reed-Muller. Nous présentons en détail une étude des propriétés de ces codes ainsi que leurs groupes d'automorphisme. Dans ce chapitre j'ai utilisé un seul point de vue pour décrire ces codes en utilisant les fonctions booléennes, et j'ai donné de nouvelles démonstrations aux deux théorèmes qui déterminent le groupe d'automorphismes de ces codes.

Dans le chapitre 3, nous décrivons le problème de décodage optimal "Décodage à maximum de vraisemblance" et l'algorithme "FHT" qui le fait pour les codes de premier ordre. J'expose ensuite tous les algorithmes de décodages des codes de Reed-Muller binaires connus jusqu'à présent. Nous commençons par la définition de la dérivée discrète d'une fonction, que j'utilise pour décrire plusieurs algorithmes de décodage, principalement le premier algorithme conçu pour décoder ces codes "Décodage par vote majoritaire : MVD". J'établis la borne de décodage asymptotique pour l'algorithme MVD pour les codes de Reed-Muller d'ordre deux.

Le chapitre 4, quant à lui, présente l'algorithme de Sidel'nikov et Pershakov. Nous décrivons d'abord l'idée de l'algorithme de décodage et nous présentons en détail l'algorithme. Ensuite, nous ferons l'analyse détaillée de la capacité de correction. Enfin, nous présentons deux variantes de l'algorithme qui ont la même capacité de correction asymptotiquement : la première est une simplification que nous proposons, tandis que la deuxième, qui est proposée par les auteurs, est une paramétrisation de l'algorithme pour pouvoir décoder plus loin mais qui rend l'algorithme plus complexe.

Le chapitre 5 apporte un résultat sur la capacité de correction des algorithmes de décodage des codes d'ordre deux. Nous présentons une étude sur le décodage souple et nous définissons un code non binaire qui nous servira pour commenter l'algorithme de Sidel'nikov et Pershakov et d'introduire la modification que nous apportons sur cet algorithme. Nous exposons notre algorithme que nous analysons ensuite pour montrer l'effet de la modification sur la capacité de correction de l'algorithme. Enfin, nous donnons une preuve que les algorithmes SP et le nôtre font du décodage borné (jusqu'à la capacité de correction).

Le sixième chapitre expose les résultats de simulation des algorithmes étudiés pour les petites et moyennes longueurs des codes de Reed-Muller. Ces résultats montrent que l'algorithme que j'ai proposé décode beaucoup plus loin en pratique que tous les autres algorithmes.

L'annexe A présente une brève introduction aux probabilités. Le but est de parler des lois de probabilités de l'approximation gaussienne. Nous aurons souvent besoin d'utiliser cette approximation pour borner la probabilité que le poids de l'erreur dépasse une certaine borne. Les inégalités connues pour borner une variable aléatoire suivant la loi normale qui sont souvent utilisées dans l'étude des différents algorithmes étudiées dans ce document.

L'annexe B contient un préalable mathématique sur la définition des groupes et les actions sur un groupe.

Chapitre 1

Introduction à la théorie algébrique des codes

Contenu

Nous allons dans ce chapitre rappeler un ensemble de définitions et propriétés concernant les codes, et plus particulièrement les codes linéaires. Nous allons parler plus en détail des groupes d'automorphismes des codes linéaires qui nous sont utiles pour la suite. Les principales références utilisées concernant la théorie algébrique des codes sont dans [31], [34], [40] et [41].

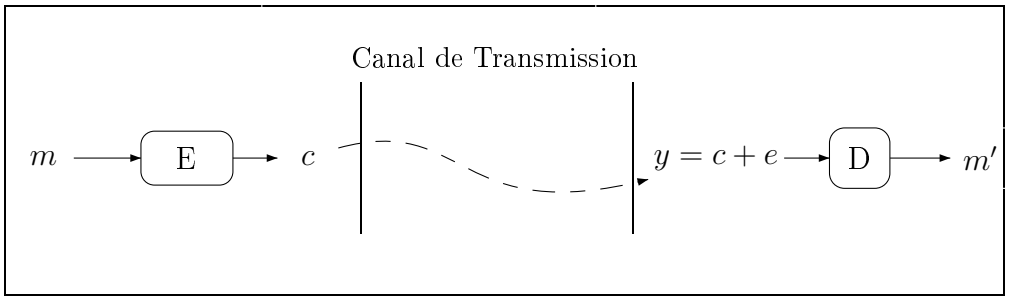


FIG. 1.1 – Principe général du processus d'encodage et décodage

1.1 Introduction et Définitions

Coder l'information est né bien avant l'apparition de l'homme, la nature utilise 4 molécules pour coder l'hérédité génétique, et les protéines utilisent 20 acides aminés pour coder leurs fonctions dans les cellules. Pour écrire, les chinois utilisent plus de 80,000 caractères pour coder leur langage, heureusement pour nous que les Phéniciens et les Grecs ont découvert qu'un alphabet de 23 caractères peuvent coder les sons élémentaires. George BOOLE (1815-1864) utilisait seulement deux caractères pour coder les opérations logiques. Peu de temps après, John von NEUMANN (1903-1957) développa le concept de programmation utilisant aussi un système binaire pour coder toute information. Pour Shannon (1948) l'unité est le bit et un système contient m-bits d'information s'il peut contenir 2^m caractères. Transmettre de l'information de manière sûre est pratiquement impossible. Dans la nature les informations ne sont pas binaires mais comprises dans des intervalles (seuil supérieur et seuil inférieur) comme les couleurs, les sons, etc., donc des erreurs sur l'information ne la modifiera pas nécessairement.

La théorie des codes est la discipline des mathématiques appliquées dont le sujet est la transmission fiable d'informations sur un canal de transmission bruité, en utilisant des objets combinatoires et algorithmiques appelés *codes correcteurs d'erreurs*. Cette théorie a de multiples champs d'applications, comme par exemple l'enregistrement des disques compacts, la transmission d'information sur les réseaux ou encore dans les communications par satellites pour n'en nommer que quelques-uns. Pour introduire le sujet, il est d'abord nécessaire de préciser les notions de base du codage.

Dans un processus de communication, nous avons trois entités impliquées : l'émetteur, le récepteur et le canal de transmission (cf. figure 1.1). L'objectif de l'émetteur est de communiquer au récepteur un message, m, c'est une suite de longueur k de symboles d'un alphabet fini $\mathcal{X}$. Le canal de transmission bruité est capable de communiquer des suites arbitrairement longues de symboles d'un alphabet $\mathcal{Y}$ pas nécessairement fini. Alors l'espace des messages à coder est $\mathcal{X}^k$, l'ensemble des suites de symboles de longueur k.

L'émetteur utilise une *fonction de codage*, E, pour coder les messages avant transmission, c'est une fonction injective : $E : \mathcal{X}^k \to \mathcal{Y}^n$. L'image $\mathcal{C} = E(m), m \in \mathcal{X}^k$ est appelé le *code*. Le taux k/n, noté en général R, est appelé le taux de transmission ou rendement du code, et c'est le premier paramètre fondamental d'un code, en théorie des codes.

En ce qui concerne le canal de transmission, il "bruite" les messages transmis. Ce bruit peut être vu comme une application de l'espace ambiant dans lui même.

Notations générales : Dans tous les chapitres de ce document : $\mathbb{F}_2$ est le corps à deux éléments, $+$ désigne l'addition dans $\mathbb{F}_2$ (sauf mention contraire) ; $\mathbb{F}_2^m$ est le $\mathbb{F}_2$-espace vectoriel de dimension m, l'addition dans $\mathbb{F}_2^m$ est notée simplement $+$, $x, y \in \mathbb{F}_2^m, x + y = (x_1 + y_1, \ldots, x_m + y_m)$.
On trouvera un index des notations à la page 133. Nous utiliserons fréquemment des notions de probabilité, l'annexe-A contient les rappels de probabilité nécessaires à la bonne compréhension des chapitres de ce document.

1.2 Le canal de transmission

La description générale est $Y = X + Z$ (X étant la source, Z le bruit et Y le signal sortant du canal). La variable aléatoire X représentera la distribution d'entrées et la variable aléatoire Y représentera la distribution de sorties. Le canal de transmission produit un vecteur de bruit $e \in \mathcal{Y}^n$ (voir 1.1), tel que un message m codé en $c = E(m)$ transmis sur le canal, sera reçu comme $y = c + e$. Le mot c sera le mot fourni en entrée et y représentera la chaîne en sortie. Les probabilités seront notées de la façon suivante :

$$\begin{array}{lcl} \Pr[c] & = & \Pr[X = c] \\ \Pr[y] & = & \Pr[Y = y] \\ \Pr[c|y] & = & \Pr[X = c|Y = y] \\ \Pr[y|c] & = & \Pr[Y = y|X = c] \end{array}$$

1.2.1 Le canal binaire symétrique

C'est un des cas les plus intéressants où l'alphabet est l'espace à deux éléments $\mathcal{X} = \mathcal{Y} = \{0, 1\}$. Il s'agit d'une transmission de symboles binaires, c'est pourquoi le canal est dit binaire. Dans ce modèle, chaque bit de donnée transmis est inversé avec une probabilité p et est transmis sans erreur avec la probabilité complémentaire de $1 - p$, donc le risque de mal transmettre "0" ou "1" est le même d'où le terme de canal symétrique.

Cette association peut se représenter par le diagramme (1.2). Pour le canal binaire symétrique, on pose p la probabilité de transmission incorrecte, sur le canal, d'un bit.

$$p = \Pr[0|1] = \Pr[1|0]$$

Si la transmission est indépendante, le canal est dit sans mémoire.

Proposition 1.2.1 *Si on transmet n bits sur un canal binaire symétrique, sans mémoire, la probabilité d'avoir j bits mal transmis est :*

$$\binom{n}{j} p^j (1 - p)^{n-j}$$

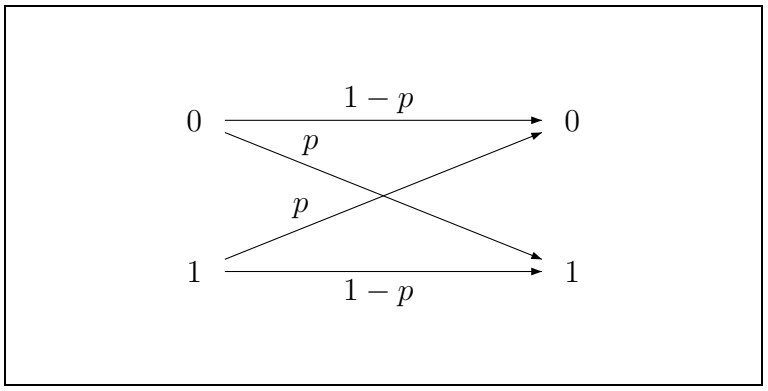


FIG. 1.2 – Diagramme de transition pour un canal binaire symétrique

Ainsi le poids de l'erreur sur un mot de longueur n, suit la loi binomiale.

1.2.2 Le canal additif gaussien blanc $AWGN$

Le bruit est blanc, gaussien de moyenne nulle et pollue les signaux de manière additive (Additive White Gaussian Noise) notée $AWGN$, dont la fonction densité de probabilité[1] $g(t)$ est une gaussienne $\mathcal{N}(0, \sigma^2)$.

$$g(t) = \frac{1}{\sigma\sqrt{2\,\pi}}\ \mathrm{e}^{-\frac{1}{2}\left(\frac{t}{\sigma}\right)^2}$$

Notons qu'une variable aléatoire ayant une fonction densité de probabilité g signifie qu'en tirant un grand nombre de fois cette variable, l'histogramme des valeurs obtenues a un profil en cloche (de la loi normale).
La source peut être discrète ou continue, le bruit est toujours supposé continu. Nous avons donc la situation suivante :

- q symboles discrets en entrée $X = \{x_1, \ldots, x_q\}$
- $Y \in \mathbb{R}$
- Z est la variable aléatoire caractérisant le bruit gaussien. Ainsi :

$$\Pr\left[y|x_k\right] = \frac{1}{\sigma\sqrt{2\pi}}\ \mathrm{e}^{-\frac{(y-x_k)^2}{2\sigma^2}}$$

Pour une transmission binaire l'alphabet d'entrée est $\{0, 1\}$, ces symboles sont transmis comme $\{+1, -1\}$, nous avons :

- $X = +1 \xrightarrow{\text{Transmis}} Y = 1 + Z$, à la sortie nous observons Y avec une densité de probabilité $f_1(t) = g(t-1) = \frac{1}{\sigma\sqrt{2\,\pi}}\ \mathrm{e}^{-\frac{(t-1)^2}{2\sigma^2}}$.
- $X = -1 \xrightarrow{\text{Transmis}} Y = -1 + Z$, à la sortie nous observons Y avec une densité de probabilité $f_{-1}(t) = g(t+1) = \frac{1}{\sigma\sqrt{2\,\pi}}\ \mathrm{e}^{-\frac{(t+1)^2}{2\sigma^2}}$.

[1] Voir l'annexe A

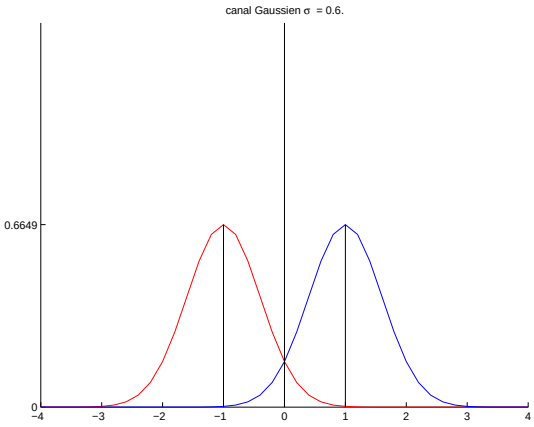


FIG. 1.3 – Canal Gaussien

Cela signifie qu'en transmettant $+1$ nous recevons à l'entrée $y \in \mathbb{R}$ la variable aléatoire observée à la sortie a comme densité de probabilité $g(t-1)$, ainsi si on veut calculer avec quelle probabilité nous observons une valeur positive :

$$\Pr[Y > 0 | X = +1] = \int_0^\infty g(t-1)\, dt$$

Nous remarquons que pour calculer la probabilité absolue d'observer une valeur positive à la sortie :

$$\Pr[Y > 0] = \Pr[Y > 0 \,|\, X = +1]\ \Pr[X = +1] + \Pr[Y > 0 \,|\, X = -1]\ \Pr[X = -1]$$

1.2.2.1 Lien entre BSC et $AWGN$

Prenons un canal $AWGN$ de paramètre $\mathcal{N}(0, \sigma^2)$. Nous voulons calculer le paramètre p du canal BSC équivalent (même niveau du bruit). Rappelons que p est la probabilité de mal transmettre un symbole de l'alphabet d'entrée (voir figure-1.2). Pour le canal $AWGN$ nous transmettons un symbole de $\{+1, -1\}$, et nous recevons une valeur réelle y à la sortie. cette valeur réelle sera arrondie à $\{+1, -1\}$ dans un canal BSC. De cette manière nous avons :

$$\begin{aligned}
p &= \Pr[Y > 0 | X = -1] \\
&= \Pr[Y < 0 | X = +1] \\
&= \int_{-\infty}^{0} \frac{1}{\sigma\sqrt{2\,\pi}}\ \mathrm{e}^{-\frac{1}{2}\left(\frac{t-1}{\sigma}\right)^2}\, dt \\
&= \int_{1/\sigma}^{\infty} \frac{1}{\sqrt{2\,\pi}}\ \mathrm{e}^{-\frac{u^2}{2}}\, du,\ u = \frac{1-t}{\sigma} \\
&= \int_{0}^{\infty} \frac{1}{\sqrt{2\,\pi}}\ \mathrm{e}^{-\frac{u^2}{2}}\, du - \int_{0}^{1/\sigma} \frac{1}{\sqrt{2\,\pi}}\ \mathrm{e}^{-\frac{u^2}{2}}\, du \\
&= \frac{1}{2} - \int_{0}^{1/\sigma} \frac{1}{\sqrt{2\,\pi}}\ \mathrm{e}^{-\frac{u^2}{2}}\, du
\end{aligned}$$

Soit $p = \frac{1}{2}(1-\epsilon)$, avec $\epsilon \ll 1$, donc nous avons l'approximation suivante :

$$\begin{aligned}
\epsilon &= 2 \int_{0}^{1/\sigma} \frac{1}{\sqrt{2\,\pi}}\ \mathrm{e}^{-\frac{u^2}{2}}\, du \\
&\approx \frac{2}{\sigma\ \sqrt{2\,\pi}}
\end{aligned}$$

AWGN	BSC
Loi normale $\mathcal{N}(0,\sigma^2)$	Loi Bernoulli $B(p),\ p=\frac{1}{2}(1-\epsilon)$
σ	$\epsilon = 2\ \int_0^{1/\sigma} \frac{1}{\sqrt{2\,\pi}}\ \mathrm{e}^{-\frac{u^2}{2}}\ du$
$\sigma = \frac{1}{\epsilon}\sqrt{\frac{2}{\pi}}$	$\epsilon \ll 1$

FIG. 1.4 – Comparer un algorithme sur les deux canaux

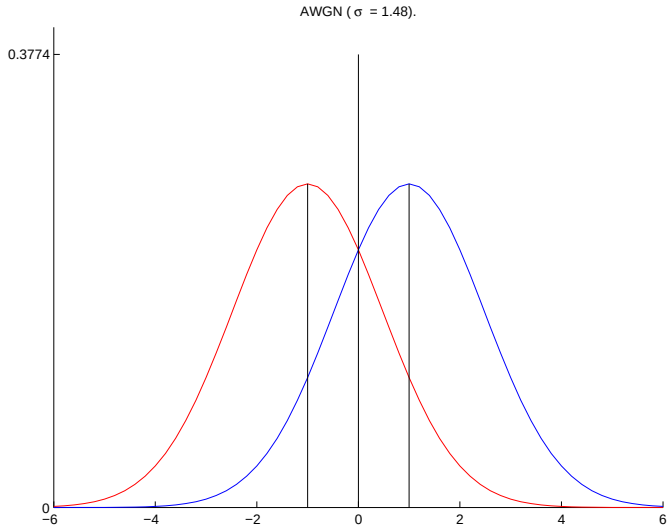


FIG. 1.5 – Si le signal reçu après transmission sur un canal AWGN ($\sigma = 1.48$) est arrondie à ± 1, La sortie est équivalent à un canal BSC avec $p = \frac{1}{4}$.

Le canal BSC nous intéresse pour les applications cryptographiques des algorithmes de décodage, et le canal $AWGN$ pour des applications dans les télécommunications.

1.3 les codes linéaires

Si on exige une structure de corps fini pour les alphabet $\mathcal{X}$ et $\mathcal{Y}$, ceci induit une structure d'espace vectoriel sur $\mathcal{Y}^n$. Nous allons considérer les codes linéaires, qui sont l'image par une application linéaire de $\mathcal{X}^k$ dans $\mathcal{Y}^n$. Donc un code linéaire est un sous-espace vectoriel sur le corps $\mathcal{Y}$.

On spécifiera un code linéaire $\mathcal{C}$ par sa matrice génératrice (base de $\mathcal{C}$), ce qui est une manière compacte de décrire un ensemble à priori de taille q^k, tel que q est le nombre de symboles de l'alphabet $\mathcal{Y}$. Tout espace de ce type est muni de la métrique de **Hamming**, cette dernière étant définie par la distance suivante :

Définition 1.3.1 *Soient u et v deux vecteurs de cet espace, la distance de Hamming entre deux mots u et v est le cardinal de l'ensemble des positions où les mots sont différents :*

$$d(u,v) = \#\{i, u_i \neq v_i\}$$

De plus, à tout vecteur u on associe son poids (de Hamming), $wt(u)$, qui est défini comme la distance entre u et le vecteur *nul*, c'est aussi le cardinal de l'ensemble des symboles non nuls.

Définition 1.3.2 *Soit $u \in \mathbb{F}_2^n$, le poids de Hamming est défini comme :*

$$\begin{array}{rcl} wt(u) & = & d(u,0) = \#\{i, u_i \neq 0\} \\ & = & \sum_{i=1}^{n} u_i \end{array}$$

où la somme sur est effectuée dans $\mathbb{N}$.

Exemple 1.3.1

$$\begin{array}{rclrcl} d(10011010, 11100010) & = & 4 & wt(10011010) & = & 4 \\ d(21101020, 10102010) & = & 4 & wt(21101020) & = & 5 \end{array}$$

La distance de Hamming nous permet de définir la notion de boule et de volume sur l'espace $\mathcal{Y}^n$.

Soit y une chaîne de $\mathcal{Y}^n$, où $|\mathcal{Y}| = q$ et soit r un entier non négatif. La boule de rayon r centrée en y notée $S_q(y, r)$ est l'ensemble

$$S_q(y, r) = \{z \in \mathcal{Y}^n | d(y, z) \leq r\}$$

Et le volume de la boule $S_q(y, r)$ noté $V_q(n, r)$ est le nombre d'éléments de $S_q(y, r)$. Le volume est indépendant de y et est donné par

$$V_q(n, r) = \sum_{k=0}^{r} \binom{n}{k} (q-1)^k$$

Dans toute la suite de ce document, on notera $\mathbb{F}$ un corps fini, On supposera $\mathcal{X} = \mathcal{Y} = \mathbb{F}$.

Définition 1.3.3 *Un code linéaire $\mathcal{C}$ de longueur n, défini sur $\mathbb{F}$, est un $\mathbb{F}$-sous-espace vectoriel de $\mathbb{F}^n$. Chacun de ses éléments est appelé mot du code $\mathcal{C}$. On associe à $\mathcal{C}$, outre sa longueur, une dimension k correspondant à la dimension de l'espace vectoriel (le code contient alors $|\mathbb{F}|^k$ mots du code) ainsi qu'une distance minimale d (c'est la plus petite distance séparant deux mots du code).*

$$d = min\{d(x, y) \mid x \neq y, x, y \in \mathcal{C}\} = min_{x \in \mathcal{C} \setminus \{0\}} \{wt(x)\}$$

On dit que $\mathcal{C}$ est un $[n, k, d]$-code linéaire, ou plus simplement on le note $\mathcal{C}[n, k]$. De plus, $\mathcal{C}$ est caractérisé par sa matrice génératrice, $G \in \mathcal{M}_{k \times n}(\mathbb{F})$: c'est une matrice à k lignes (linéairement indépendantes) et n colonnes, à coefficients dans $\mathbb{F}$, et telle que

$$\mathcal{C} = \{mG, m \in \mathbb{F}^k\}$$

La matrice génératrice n'est bien sûr pas unique.

Définition 1.3.4 *Soit $\mathcal{C}$ un $[n, k, d]$-code linéaire. On appelle rayon de recouvrement de $\mathcal{C}$ le plus petit entier ρ tel que l'ensemble de toutes les boules de rayon ρ (pour la distance de Hamming) centrées en les mots de $\mathcal{C}$ recouvre tout l'espace $\mathbb{F}_2^m$.*

Le but du codage est évidemment la détection et surtout la correction d'erreurs. Il nous faut cependant définir ce que l'on entend par détection et correction d'erreurs dans le cadre de la théorie du codage.

Définition 1.3.5 *Soit $t \geq 1$. Nous dirons qu'un code $\mathcal{C}$ détecte t erreurs si pour $x \in \mathcal{C}$, les mots $y \in \mathbb{F}_2^n$ vérifiant $x \neq y$ et $d(x, y) \leq t$ ne sont pas dans $\mathcal{C}$.*

Définition 1.3.6 *Soit $t \geq 1$. Nous dirons qu'un code $\mathcal{C}$ corrige t erreurs ou moins si pour tout $y \in \mathbb{F}_2^n$ il existe au plus un $x \in \mathcal{C}$ tel que $d(x, y) \leq t$.*

Pour ce qui est de la détection, il est clair qu'un $[n, k, d]$-code $\mathcal{C}$ détectera t erreurs si et seulement si $t < d$. Pour ce qui est de la correction, un code avec distance minimale d corrige t erreurs si et seulement si $d \geq 2t + 1$, et si t est le plus grand entier vérifiant $2t + 1 \leq d$, alors des sphères disjointes de rayon t, peuvent être centrées sur chaque mot de code. t est alors appelée **la capacité de correction** du code et vaut $\lfloor (d-1)/2 \rfloor$.

Nous définissons le code le plus simple qui est le code à répétition, que nous utilisons aussi comme exemple pour définir les propriétés des codes et leur encodage.

Définition 1.3.7 *Le code à répétition consiste à transmettre plusieurs copies de chaque bit, c'est le $[n, 1, n]$-code linéaire binaire, a pour matrice génératrice $G \in \mathcal{M}_{1\times n}$:*

$$G = (1 \ \ldots \ 1)$$

Ainsi la dimension du code est $k = 1$, sa longueur n, et sa distance minimale $d = n$. Autrement dit, ce code à répétition, encode la transmission des bits ainsi :

$$\begin{array}{ccc} 1 & \longrightarrow & 11\ldots1 \\ 0 & \longrightarrow & 00\ldots0 \end{array}$$

Ce code détecte $n - 1$ erreurs, et corrige $\lfloor (n-1)/2 \rfloor$.

1.3.1 Code dual

La contrainte de linéarité sur le code donne naturellement naissance à la notion de code dual d'un code linéaire. Puisque $\mathcal{C}$ est un espace vectoriel, on peut considérer l'ensemble des formes linéaires qui s'annulent sur $\mathcal{C}$. Cet ensemble est un espace vectoriel que l'on appelle code dual de $\mathcal{C}$.

Définition 1.3.8 *Soit $\mathcal{C}$ un $[n, k, d]$-code linéaire. Soit $< ., . >$ le produit scalaire euclidien usuel[2] : $< c, v >= \sum_{i=1}^{n} c_i v_i$. Le code dual, noté $\mathcal{C}^\perp$, est donc un code linéaire de la même longueur. Sa dimension est $n - k$.*

$$\mathcal{C}^\perp = \{v \in \mathbb{F}^n \mid \forall\ c \in \mathcal{C} \ : < c, v >= 0\}.$$

Il découle directement des définitions que si H est une matrice génératrice du code $\mathcal{C}^\perp$, elle est communément appelée matrice de parité[3] du code $\mathcal{C}$. De même, une matrice génératrice de $\mathcal{C}$ est une matrice de parité de $\mathcal{C}^\perp$.

Remarque 1 *Il n'y a pas (a priori) de relation simple entre la distance minimale de $\mathcal{C}$ et celle de $\mathcal{C}^\perp$. Le théorème de MacWilliams [34] ne peut pas être utilisé simplement pour*

[2] Le produit scalaire euclidien est noté plus simplement $< c, v >= c.v^t$

[3] H est une $(n, n-k)$-matrice de parité d'un code $\mathcal{C}$ si et seulement si $\forall c \in \mathcal{C} : H.c^t = 0$

déterminer la distance minimale ; pour le faire nous devons connaître la distribution des poids du code. Si $G_{\mathcal{C}}$ la matrice génératrice du code $\mathcal{C}$ est de la forme $G_{\mathcal{C}} = [I_k, M]$ où I_k est la matrice unité de taille $(k \times k)$ et M une matrice de taille $k \times (n-k)$, alors nous avons : $H = [M^t, -I_{n-k}]$ est une matrice de parité du code $\mathcal{C}$ où M^t est la matrice transposée de la matrice M.

1.4 Groupe d'automorphismes

Nous gardons les notations de [42], mais nous choisissons les opérations à gauche des éléments des groupes définis par la suite (voir Annexe-B). Soit $\mathcal{C}[n,k]$ un code linéaire sur $\mathbb{F}$, avec des coordonnées indiqués par les éléments de $\mathcal{I}_n = \{1, 2, \ldots, n\}$. Rappelons que $\mathcal{S}_n$ est le groupe symétrique de toutes les permutations de n-symboles. Soit σ une permutation de $\mathcal{S}_n$, elle est définie, de manière équivalente, par sa matrice P_σ de taille $(n \times n)$, avec $P_\sigma = [P_{i,j}]$ tel que $p_{i,j} = 1$ ssi $i = \sigma(j)$.
L'action d'une permutation $\sigma \in \mathcal{S}_n$ sur un vecteur $x \in \mathbb{F}^n$ est donnée par l'équation suivante[4] :

$$\forall x = (x_1, \ldots, x_n) \in \mathbb{F}^n, \sigma(x) = (x_{\sigma(1)}, \ldots, x_{\sigma(n)})$$

Elle s'accorde avec la multiplication ordinaire d'un vecteur par une matrice $v\ P_\sigma$.

$$\begin{array}{rcl} P_\sigma : \mathbb{F}^n & \rightarrow & \mathbb{F}^n \\ x = (x_1, \ldots, x_n) & \mapsto & x.P_\sigma = (x_{\sigma(1)}, \ldots, x_{\sigma(n)}) \end{array}$$

Si π est une autre permutation, le produit $\pi\sigma$ veut dire qu'il faut d'abord appliquer σ, après π. Alors $(\pi\sigma)x = \pi(\sigma(x))$.

Exemple 1.4.1 *Soit* $\pi = (123)$, $\sigma = (14)$

$$\begin{array}{rcl} \sigma(x) & = & (x_4, x_2, x_3, x_1) \\ (\pi\sigma)(x) & = & (x_2, x_3, x_4, x_1) \end{array}$$

Notons $\mathcal{M}_n(\mathbb{F})$ l'ensemble des matrices de taille $n \times n$ sur $\mathbb{F}$ contenant un seul élément non nul sur chaque ligne et chaque colonne. Ce groupe est isomorphe à $\mathcal{S}_n \times (\mathbb{F}^*)^n$, il est défini de la façon suivante :

$$(\sigma; d)x = \sigma(d.x);\ \text{pour } x \in \mathbb{F}^n, (\sigma; d) \in \mathcal{S}_n \times (\mathbb{F}^*)^n$$

où la multiplication dans $\mathbb{F}^n$ est définie par

$$(x_1, \ldots, x_n).(y_1, \ldots, y_n) = (x_1.y_1, \ldots, x_n.y_n)$$

Nous avons exigé que l'alphabet $\mathbb{F}$ soit un corps. Nous notons $Gal(\mathbb{F})$ Le groupe Galois[5] de $\mathbb{F}$ sur son sous-corps premier (En particulier si $\gamma \in Gal(\mathbb{F})$ nous avons $\gamma(0) = 0$), la

[4] Nous disons que $\mathcal{S}_n$ opère à gauche sur $\mathbb{F}^n$, Annexe-B

[5] $\mathbb{F}$ un corps fini, soit $\mathbb{F}_p$ son sous-corps premier, $Gal(\mathbb{F})$ est l'ensemble des $\mathbb{F}_p$-automorphismes, ce sont les automorphismes de $\mathbb{F}$ fixant les éléments de son sous-corps premier $\mathbb{F}_p$

multiplication dans ce groupe est donné par $(\gamma\tau)x = \gamma(\tau x)$, $x \in \mathbb{F}$, $\gamma, \tau \in Gal(\mathbb{F})$. L'action de $Gal(\mathbb{F})$ sur $\mathcal{M}_n(\mathbb{F})$ produit le groupe de bijections semi-linéaires de $\mathbb{F}^n$. L'effet d'un élément de $W_n(\mathbb{F}) \stackrel{\triangle}{=} Gal(\mathbb{F}) \times \mathcal{M}_n(\mathbb{F})$ (défini dans [42]) sur les mots d'un code $\mathcal{C}$ génère un code avec les mêmes propriétés (la même distance minimale, distribution des poids ,etc.), puisque les éléments de ce groupe préservent le poids de Hamming des mots du code.

Définition 1.4.1 *On appelle groupe d'automorphismes de $\mathcal{C}$ (noté $\mathcal{A}ut(\mathcal{C})$) l'ensemble des éléments de $W_n(\mathbb{F})$ qui laissent $\mathcal{C}$ globalement invariant, c'est-à-dire qui transforment un mot de $\mathcal{C}$ en un autre mot de $\mathcal{C}$.*

$$(\gamma, \sigma; d) \in \mathcal{A}ut(\mathcal{C}) \Leftrightarrow \forall c \in \mathcal{C}, (\gamma, \sigma; d)c = (\gamma(c_{\sigma(1)}.d_{\sigma(1)}), \ldots, \gamma(c_{\sigma(n)}.d_{\sigma(n)})) \in \mathcal{C} \tag{1.1}$$

Dans la littérature nous trouvons aussi deux autres groupes. Le premier est le groupe d'automorphismes monomial $MAut(\mathcal{C})$, ce sont les éléments de $\mathcal{M}_n(\mathbb{F})$ qui laissent le code globalement invariant. Le deuxième est le groupe de permutation $PAut(\mathcal{C})$ c'est les éléments de $\mathcal{S}_n(\mathbb{F})$ qui laissent le code globalement invariant.

Pour un code linéaire $\mathcal{C}$, nous distinguons deux cas particuliers :
- $\mathbb{F} = \mathbb{F}_2$, nous avons $\mathcal{M}_n(\mathbb{F}_2) = \mathcal{S}_n, Gal(\mathbb{F}_2) = \{Id\} \Rightarrow \mathcal{A}ut(\mathcal{C}) \equiv MAut(\mathcal{C}) \equiv PAut(\mathcal{C})$
- $X = \mathbb{F}_p$, p premier, nous avons $Gal(\mathbb{F}_p) = \{Id\} \Rightarrow \mathcal{A}ut(\mathcal{C}) \equiv MAut(\mathcal{C})$

Ainsi pour un code linéaire binaire $\mathcal{C}$, $\mathcal{A}ut(\mathcal{C})$ est un sous-groupe du groupe symétrique $\mathcal{S}_n$.

Exemple 1.4.2 *Le groupe d'automorphismes du code à répétition binaire (definition-1.3.7) est $\mathcal{S}_n$.*
Pour toute permutation $\forall \sigma \in \mathcal{S}_n$, σ n'a pas d'effet sur le code à répétition qui contient deux mots :

$$\forall c \in \{11\ldots1, 00\ldots0\},\ \sigma(c) = c$$

Pour les codes linéaires, nous avons deux résultats importants, qui nous permettent d'identifier leur groupe d'automorphismes. Rappelons que l'ensemble des matrices carrées inversibles d'ordre n à coefficient dans un corps $\mathbb{F}$ est noté $GL_n(\mathbb{F})$:

Lemme 1.4.1 *Soit $\mathcal{C}$ un $[n, k]$-code linéaire défini sur $\mathbb{F}$, avec une matrice génératrice G. Une permutation P de taille $(n \times n)$ sur les coordonnées du code $\mathcal{C}$ appartient au groupe d'automorphismes du code, si et seulement si, il existe une matrice inversible $K \in GL_n(\mathbb{F})$ vérifiant :*

$$K.G = G.P$$

▷ Preuve : Rappelons d'abord que si E est un $\mathbb{F}$-espace vectoriel de dimension finie, que e et $\tilde{e}$ sont deux bases de E, alors la matrice de passage de la base e à la base $\tilde{e}$ est inversible. Soit G une matrice génératrice d'un code linéaire $\mathcal{C}$, qui est un sous espace vectoriel donc un espace vectoriel. Les lignes de cette matrice forment une base de l'espace vectoriel $\mathcal{C}$.

La matrice $G.P$ est une matrice génératrice du code linéaire $\mathcal{C}$, qui est un sous espace vectoriel de $\mathbb{F}^n$, si et seulement si la matrice $G.P$ peut être obtenue à partir de G par une transformation linéaire K. ◁

Proposition 1.4.1 *Soit $\mathcal{C}$ un code linéaire défini sur $\mathbb{F}$, le groupe de permutation de $\mathcal{C}$ est aussi le groupe de permutation de son dual $\mathcal{C}^\perp$:*

$$PAut(\mathcal{C}) = PAut\left(\mathcal{C}^\perp\right)$$

▷ Preuve : Soient G la matrice génératrice et H la matrice de parité d'un code linéaire $\mathcal{C}$, et une permutation $P \in PAut(\mathcal{C})$. Selon le lemme précédent il existe une matrice inversible K tel que : $G.P = K.G$. la permutation P est dans le groupe d'automorphismes du code dual si pour les mots du code dual $c \in \mathcal{C}^\perp$ on a $c.P \in \mathcal{C}^\perp$. Soit $c \in \mathcal{C}^\perp$, G est la matrice de parité du code $\mathcal{C}^\perp$, donc :

$$G.c^t = 0 \Rightarrow G.P.P^t.c^t = 0 \text{ puisque } P.P^t = I_n \text{ (matrice identité)}$$

Selon le lemme précédent, il existe une matrice inversible K tel que $K.G = G.P$, alors

$$K.G.P^t.c^t = 0 \Rightarrow G.P^t.c^t = 0 \Rightarrow G.(cP)^t = 0 \Rightarrow c.P \in \mathcal{C}^\perp$$

D'où $P \in PAut\left(\mathcal{C}^\perp\right)$, nous avons prouvé que

$$PAut(\mathcal{C}) \subset PAut\left(\mathcal{C}^\perp\right)$$

Et puisque

$$(\mathcal{C}^\perp)^\perp = \mathcal{C}$$

Alors $PAut\left(\mathcal{C}^\perp\right) \subset PAut\left((\mathcal{C}^\perp)^\perp\right) = PAut(\mathcal{C})$. ◁

Note 1 *Pour un code linéaire binaire $\mathcal{C}$, où $\mathbb{F} = \mathbb{F}_2 \Rightarrow \mathcal{A}ut\left(\mathcal{C}\right) \equiv PAut(\mathcal{C})$, la proposition-1.4.1 nous permet d'écrire $\mathcal{A}ut\left(\mathcal{C}\right) = \mathcal{A}ut\left(\mathcal{C}^\perp\right)$.*

Nous avons parlé des codes qui ont pour but d'éviter les erreurs dues à la transmission. Nous allons parler maintenant des différents types d'algorithme de décodage, et comment caractériser leur performance.

1.5 Le décodage

Le récepteur utilise une fonction de décodage $D : \mathcal{Y}^n \to \mathcal{X}^k$. Le décodage D doit être rapide, et tel que, avec une grande probabilité il nous retourne le mot transmis. Intuitivement, le code introduit une redondance en augmentant la longueur des messages, et cette redondance sera utilisée pour décoder le message transmis, même s'il est bruité.

Définition 1.5.1 *Soit $\mathcal{C}$ un code sur $\mathcal{Y}$. Soit $c \in \mathcal{C}$ un mot émis à travers le canal, et y le mot reçu.*

1. ***MLD** (Maximum Likelihood Decoding) On appelle décodage à maximum de vraisemblance, l'algorithme qui permet d'obtenir le mot de code qui a la plus grande probabilité d'être transmis ayant reçu le mot erroné.*
2. ***MDD**(Minimum Distance Decoding) On appelle décodage à Distance minimale, l'algorithme qui permet d'obtenir le mot de code le plus proche du mot reçu, au sens de la métrique de Hamming[6] ou de la métrique euclidienne.*
3. ***LD** (List Decoding : décodage en liste)[7] Soit $\delta > 0$, On dit qu'un algorithme de décodage D décode en liste le code $\mathcal{C}$ avec un rayon de décodage δ, si D nous fournis la liste de tous (éventuellement aucun) les mots du code à une distance au plus δ du mot reçu.*
4. ***BDD** (Bounded Distance Decoding : décodage borné)[7] Une borne e est donnée. L'obtention d'un mot parmi les mots de code à distance e du mot reçu (s'il en existe).*
5. ***UD** (Unambiguous Decoding : décodage non ambigu)[7] Ici on se donne $e = (d-1)/2$, où d est la distance minimale du code, et on cherche le mot de code à distance e du mot reçu (s'il existe).*

Selon le canal de transmission considéré nous avons deux sorte de décodages, nous allons définir ces types :

Définition 1.5.2 *Soit $\mathcal{Y}$ l'alphabet de sortie d'un canal de transmission. Dans la Littérature du codage nous rencontrons deux grandes catégories de décodage*

- *décodage dur (hard decoding) : Pour un alphabet de sortie fini, et le plus souvent quand $\mathcal{Y} = \mathcal{X}$, pour ce type de canal une erreur de transmission consiste à remplacer un symbole par un autre, par exemple : décodage pour une transmission sur un canal binaire BSC.*
- *décodage souple (soft decoding) : Pour un alphabet de sortie non fini, le plus souvent $\mathbb{R}$, par exemple : décodage pour une transmission sur un canal gaussien $AWGN$.*

Un algorithme de décodage est conçu pour faire du décodage dur ou souple, et il peut faire les deux, à notre connaissance il n'existe pas un algorithme de décodage souple qui

[6] définition en page 17

[7] Ces définitions viennent de [3].

ne fait pas de décodage dur. Les algorithmes de décodage ont, en général, une meilleure performance en décodage souple qu'en dur.

Note 2 *Un algorithme, qui fait du décodage souple, fait aussi du décodage dur à condition de faire tous les calculs nécessaires dans* $\mathbb{R}$.

1.6 Performances d'un algorithme de décodage

Nous mesurons la capacité de décodage d'un algorithme de décodage D, à l'aide de deux notions principales, le ***BER*** (Bit Error Rate) est la probabilité de mal décoder un bit d'information, par exemple : pour trois bits erronés sur 1000 bits codés et transmis, le BER vaut 3.10^{-3}. et ***WER*** (Word Error Rate) qui est la probabilité de mal décoder un mot de k bits. Pour un canal de transmission sans mémoire, nous avons la relation suivante entre ces deux termes :

$$WER = 1 - (1 - BER)^k$$

Pour comparer les algorithmes de décodage nous nous intéressons à la borne de décodage asymptotique de ces algorithmes. La notion de borne de décodage asymptotique a été définie et utilisée par Dumer dans la plupart de ses articles traitant les codes de Reed-Muller (voir [15],[9],[8],[10],[16]) :

Définition 1.6.1 *Étant donnée une séquence de codes* $\mathcal{C}_i$ *de longueur croissante* $n_i \mapsto \infty$, *soit* τ_i *un terme résiduel* $\tau_i \mapsto 0$, *nous dirons que l'algorithme de décodage* D *a* $\delta_i(1 - \tau_i)$ *comme* ***borne de décodage asymptotique*** *si :*

$D(\mathcal{C}_i)$ retrouve avec une grande probabilité le mot transmis lorsque le poids (Hamming) du vecteur d'erreur est plus petit que $\delta_i(1 - \kappa\tau_i),\ \kappa > 1$
Le terme "grande probabilité" veut dire dans ce contexte que si $P_{err}(t)$ *est la probabilité que l'algorithme* D *n'arrive pas à décoder un mot du code erroné avec une erreur de poids* t, *alors :*

$$P_{err}(t) \stackrel{\triangle}{=} \Pr\left[D(y) \neq c \mid y = c + e,\ wt(e) = t \leq \delta_i(1 - \kappa\ \tau_i),\ \kappa > 1\right] \xrightarrow[n_i \to +\infty]{} 0$$

La borne est dite ***stricte*** *si en plus :*
$D(\mathcal{C}_i)$ a une probabilité non négligeable de ne pas retourner le mot transmis si le poids (Hamming) du vecteur d'erreur est plus grand que $\delta_i(1 - \tau_i)$.

La borne de décodage asymptotique reviendra souvent dans la thèse et elle sera établie pour tous les algorithmes présentés dans ce manuscrit. Soit A un événement, nous utilisons

le terme "avec une grande probabilité" qui signifie que la probabilité, que l'événement A arrive, tend vers 1 ou que son complémentaire tends vers 0.

Nous allons maintenant parler du décodage du code à répétition. (définition-1.3.7).

Exemple 1.6.1 *Décodage à distance minimale MDD du code à répétition :*
Puisque nous n'avons que deux mots dans ce code, nous pouvons décoder à distance minimale en mesurant la distance du mot reçu à tous les mots du code. Le poids de Hamming du mot reçu $y \in \mathbb{F}_2^n$ *n'est que sa distance au mot tout à zéro* $(0 \dots 0)$*, noté plus simplement* 0*. Soit*

$$t = d(y, 0) = wt(y)$$

Nous estimons que 1 *a été transmis si* $t > \lfloor (n-1)/2 \rfloor$*, et* 0 *sinon.*

Notons que le décodage par énumération est possible pour des codes de petite dimension, et il devient rapidement impraticable si la dimension croît.

Chapitre 2

Codes de Reed-Muller

Contenu

Dans ce chapitre, nous allons présenter les codes de Reed-Muller (RM), dont l'étude a fait l'objet de ma thèse. Les principales références utilisées sont dans [34] et dans [41]. Nous avons décrit ces codes d'une manière similaire à [34], mais avec plus de détails, surtout quand nous parlerons de décodage. Nous décrivons les codes en terme de fonctions booléennes et nous avons démontré à l'aide de cette description tous les résultats concernant ces codes, comme les preuves des théorèmes 2.4.2 et 2.4.1, ces deux théorèmes sont démontrés dans [34] en utilisant la description géométrique des codes RM.

2.1 Introduction

Étudiant l'application de l'algèbre de Boole aux circuits de communication, en 1954 D. E. Muller [36] construit une classe de codes connus sous le nom de Reed-Muller car S. I. Reed [43] aura soin de développer un algorithme de décodage en vote majoritaire pour ces codes.

Malgré le fait que leur distance minimale soit relativement petite, ces codes ont un intérêt pratique grâce à la simplicité de leurs algorithmes de codage et de décodage. Cette caractéristique leur a valu d'être utilisés entre autres lors de la transmission des images de la planète Mars envoyées par la sonde Mariner 9 en 1972.

Les codes de Reed-Muller peuvent être définis très simplement en termes de fonctions booléennes. Nous allons définir les fonctions booléennes et leur caractéristique qui nous sont utiles pour la description des codes de RM.

Notations générale : Dans tous les chapitres qui suivent : Nous avons choisi l'ordre lexicographique pour représenter les éléments de $\mathbb{F}_2^m = \{\alpha_1, \ldots, \alpha_n\}, n = 2^m$, avec α_i comme l'écriture en binaire[1] sur m bits de l'entier $i-1$. L'addition dans cet espace vectoriel est le ou-exclusif bit-à-bit. On trouvera un index des notations à la page 133.

2.2 Fonctions booléennes $\mathcal{F}_m$

Dans toute la suite, on choisit un entier $m > 1$. Soit $\mathcal{F}_m$ l'espace des fonctions de $\mathbb{F}_2^m$ dans $\mathbb{F}_2$, c'est-à-dire des fonctions booléennes dont les arguments sont des mots binaires de longueur m. Nous démontrons que la dimension de $\mathcal{F}_m$ est 2^m, qui n'est autre que la cardinalité de l'espace $\mathbb{F}_2^m$.

Soit f une fonction booléenne de m variables, $f : \mathbb{F}_2^m \to \mathbb{F}_2$. Elle est entièrement caractérisée par sa table de vérité, c'est-à-dire la liste de tous les éléments de $\mathbb{F}_2^m$ avec les valeurs qu'elle prend en chacun d'eux. Nous allons maintenant définir les termes qui caractérisent une fonction booléenne.

Définition 2.2.1 *On appelle support de la fonction booléenne f l'ensemble des éléments x tels que $f(x) \neq 0$, et on le note $supp(f)$. On appelle poids[2] de f le cardinal de son support, et on le note $wt(f)$. On définit également pour deux fonctions f et g la distance[2] qui les sépare par $d(f,g) = wt(f+g)$*

$$\begin{aligned} supp(f) &= \{x \in \mathbb{F}_2^m | f(x) \neq 0\} \\ wt(f) &= \#supp(f) \\ d(f,g) &= wt(f+g) \end{aligned}$$

[1] $(x_1, \ldots, x_m)_{LSB}$ l'écriture en binaire de x si $x = \sum_{i=1}^{m} x_i.2^{i-1}$, LSB : (Least significant bit) Bit de poids faible d'abord

[2] Ce sont la distance et le poids de Hamming de la table de vérité de la fonction, voir définition 1.3.1

Exemple 2.2.1 *Considérons le cas $m = 3$. Voici la définition d'une fonction f par sa table de vérité :*

éléments de $\mathbb{F}_2^3$	*valeur de* f
$(0,0,0)$	0
$(1,0,0)$	1
$(0,1,0)$	1
$(1,1,0)$	0
$(0,0,1)$	0
$(1,0,1)$	1
$(0,1,1)$	1
$(1,1,1)$	0

On a ici $supp(f) = \{(1,0,0),(0,1,0),(1,0,1),(0,1,1)\}$ *et* $wt(f) = 4$.

Nous allons maintenant introduire le concept de Forme Algébrique Normale d'une fonction booléenne. Pour une description plus détaillée des fonctions booléennes, on peut par exemple se reporter à la thèse de Caroline Fontaine [21].

2.2.1 Fonctions booléennes $\mathcal{F}_m$ et espace vectoriel $\mathbb{F}_2^{2^m}$

Nous allons montrer dans ce paragraphe la correspondance entre l'espace des fonctions booléennes $\mathcal{F}_m$ et l'espace vectoriel $\mathbb{F}_2^{2^m}$.
Nous avons vu que toute application $f : \mathbb{F}_2^m \to \mathbb{F}_2$ est polynomiale. En effet, si $y = (y_1, \ldots, y_m) \in \mathbb{F}_2^m$, soit le polynôme :

$$\begin{array}{rcl} \delta_y : \mathbb{F}_2^m & \to & \mathbb{F}_2 \\ x & \mapsto & \prod_{i=1}^m (1 + x_i + y_i) \ , \quad \text{qui vérifie} \end{array}$$

$$\delta_y(x) = \begin{cases} 1 & \text{si } x = y \\ 0 & \text{sinon} \end{cases}$$

À partir de là, toute fonction $f : \mathbb{F}_2^m \to \mathbb{F}_2$ s'écrit

$$f(x) = \sum_{\alpha \in \mathbb{F}_2^m} f(\alpha)\delta_\alpha(x) \tag{2.1}$$

(et la famille est donc génératrice),

- C'est manifestement une famille libre. Ainsi nous avons :

$$dim(\mathcal{F}_m) = 2^m, \text{ le nombre de vecteur de cette base}$$

La famille des $(\delta_x)_{x \in (\mathbb{F}_2^m)}$ est une base de $\mathcal{F}_m$. L'équation-2.1 nous indique que toute fonction de $\mathcal{F}_m$ est une fonction polynomiale, nous donnons maintenant la définition de la forme algébrique normale.

Définition 2.2.2 *La Forme Algébrique Normale (FAN) d'une fonction booléenne f à m variables est l'unique polynôme Q_f de $\mathbb{F}_2[x_1,\ldots,x_m]/(x_1^2-x_1,\ldots,x_m^2-x_m)$ tel que*

$$\forall (x_1,\ldots,x_m)\in\mathbb{F}_2^m, f(x_1,\ldots,x_m)=Q_f(x_1,\ldots,x_m).$$

On appelle degré de f, et on le note $\deg(f)$, le degré du polynôme Q_f.

Un monôme est une expression de la forme :

$$p=x_1^{i_1}x_2^{i_2}\ldots x_m^{i_m}\ ,\ (i_j)_{j=1\ldots m}\in\{0,1\}$$

Soit $I_\alpha=\{\alpha_1,\ldots,\alpha_{2^m}\}$ une énumération de tous les éléments de $\mathbb{F}_2^m$. Nous allons montrer la correspondance entre une table de vérité et une fonction.

Définition 2.2.3 *Écriture vectorielle des fonctions booléennes de m variables.*

L'application bijective

$$\begin{array}{rcl} ev_{I_\alpha}:\mathcal{F}_m & \to & \mathbb{F}_2^{2^m} \\ f & \mapsto & (f(\alpha_1),\ldots,f(\alpha_{2^m})) \end{array}$$

Cette seconde écriture de f revient à remplacer la fonction f par la 2^m-liste de ses coordonnées dans la base $(\delta_x)_{x\in(\mathbb{F}_2^m)}$. Nous allons maintenant définir l'inverse de l'évaluation qui est la suivante :

Définition 2.2.4 *Soit $v\in\mathbb{F}_2^{2^m}$, nous allons définir la fonction booléenne $f_v\in\mathcal{F}_m$ qui a le vecteur v comme table de vérité :*

$$\begin{array}{rcl} ev_{I_\alpha}^{-1}:\mathbb{F}_2^{2^m} & \to & \mathcal{F}_m \\ v & \mapsto & f_v(x)=\sum_{i=\alpha_1}^{\alpha_{2^m}} v_{\alpha_i}\delta_{\alpha_i}(x) \end{array}$$

La fonction d'évaluation des fonctions booléennes (définition-2.2.3), notée ev, est une fonction linéaire possédant les propriétés suivantes :

Propriété 2.2.1 *Soient $f,g\in\mathcal{F}_m$, $a\in\mathbb{F}_2$, nous avons :*
- $ev(f+g)=ev(f)+ev(g)$
- $ev(f\cdot g)=ev(f)\cdot ev(g)$
- $ev(a\cdot f)=a\cdot ev(f)$

avec l'addition et la multiplication dans $\mathbb{F}_2^{2^m}$ éléments par éléments : $v,w\in\mathbb{F}_2^{2^m}, v+w=(v_1+w_1,\ldots,v_{2^m}+w_{2^m}), v\cdot w=(v_1\cdot w_1,\ldots,v_{2^m}\cdot w_{2^m})$

Exemple 2.2.2 *Considérons le cas m=2. définissons la fonction f suivante, qui correspond au xor :*

$$\begin{array}{rcl} f : \mathbb{F}_2^m & \rightarrow & \mathbb{F}_2 \\ (0,0) & \mapsto & 0 \\ (1,0) & \mapsto & 1 \\ (0,1) & \mapsto & 1 \\ (1,1) & \mapsto & 0 \end{array}$$

On a :

$$\begin{array}{rcl} f & = & 0.\delta_{(0,0)} + 1.\delta_{(1,0)} + 1.\delta_{(0,1)} + 0.\delta_{(1,1)} \\ & = & \delta_{(1,0)} + \delta_{(0,1)} \end{array}$$

d'où, $ev(f) = (0110)$.

On notera qu'il est très commode, lorsque l'on désigne δ_u, d'identifier le vecteur binaire u et l'entier dont il est la décomposition binaire (par exemple, δ_1 et $\delta_{(1,0)}$). Nous avons

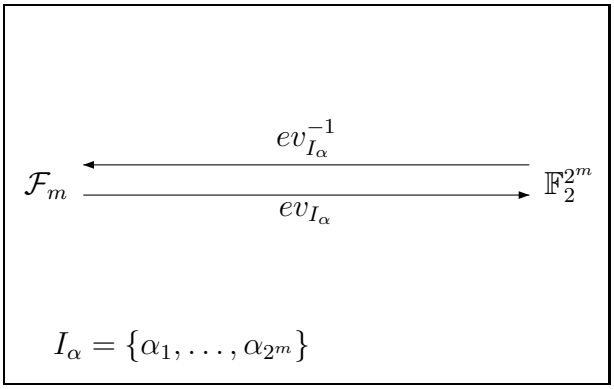


FIG. 2.1 – Correspondance entre l'espace des fonctions et l'espace vectoriel.

démontré qu'il y a une correspondance entre l'espace des fonctions à m variables et l'espace vectoriel $\mathbb{F}_2^{2^m}$ (figure 2.1). Ainsi toute opération que nous allons définir sur l'un des espaces sera définie de manière équivalente sur l'autre.

2.2.2 Base de l'espace des fonctions $\mathcal{F}_m$

Nous allons décrire comment associer un vecteur de $\mathbb{F}_2^{2^m}$ à une fonction booléenne écrite sous forme algébrique normale.

Dans cet espace de fonctions (i.e. $\mathcal{F}_m$), nous allons maintenant définir des fonctions particulières. Pour tout $i \in \{1 \ldots m\}$, on note X_i la $i$ème projection :

$$\begin{array}{rcl} X_i : (\mathbb{F}_2^m) & \rightarrow & \mathbb{F}_2 \\ (x_1, \ldots, x_m) & \mapsto & x_i \end{array}$$

Notons que pour plus de simplicité, dans la suite de cette thèse, nous noterons simplement les fonctions X_i en minuscule, quand cela n'introduit pas d'ambiguïté.

Par ailleurs, on note **1** la fonction unité (c'est-à-dire la fonction dont la valeur est 1 quel que soit son argument).

Comme nous travaillons sur $\mathbb{F}_2$, alors on a $x_i^k = x_i, \forall k > 0$, nous définissons le groupe de fonctions suivant :

$$\mathcal{M} = \{\mathbf{1} \cdot x_{i_1} \cdot x_{i_2} \dots x_{i_k} \,|\, 0 \leq i_1 < i_2 \cdots < i_k \leq m, \text{ pour } 0 \leq k \leq m\} \tag{2.2}$$

C'est l'ensemble de 2^m monômes ($\dim(\mathcal{F}_m)$=2^m), La combinaison linéaire de tous ces monômes génère toutes les fonctions polynomiales $\mathcal{F}_m$, et comme nous pouvons réduire toute fonction polynomiale en x_i, en calculant le modulo $x_i^2 - x_i$, **L'ensemble $\mathcal{M}$ est une autre base de l'espace $\mathcal{F}_m$, qui sera très intéressante pour définir les codes de Reed-Muller**.

2.2.3 Base de l'espace vectoriel $\mathbb{F}_2^{2^m}$

Nous savons que l'image d'une base par une application linéaire bijective est une base, donc l'image de la base $\mathcal{M}$ (2.2) par l'application ev (def-2.2.3) est une base de l'espace $\mathbb{F}_2^{2^m}$.

En utilisant la propriété-2.2.1, nous pouvons calculer l'image de tous les monômes de la base $\mathcal{M}$ si on connaît l'image des monômes de degré zéro et un.

Le monôme de degré zéro est la fonction constante $\mathbf{1} : \mathbb{F}_2^m \to \mathbb{F}_2$, qui donne la valeur 1 pour tous les éléments de $\mathbb{F}_2^m$.

$$ev(\mathbf{1}) = (1, 1, \dots, 1)$$

On notera que pour tout $i \in \{1, \dots, m\}$, on a :

$$ev(X_i) = (X_i(\alpha_1), \dots, X_i(\alpha_{2^m}))$$

Donc la j-ème coordonnée de $ev(X_i)$ est 1 si $(\alpha_j)_i$ est non nul et 0 sinon.

Exemple 2.2.3 *Dans le cas $m = 3$, Nous pouvons écrire l'exemple 2.2.1.*

	α_1	α_2	α_3	α_4	α_5	α_6	α_7	α_8
$ev(X_1)$	0	1	0	1	0	1	0	1
$ev(X_2)$	0	0	1	1	0	0	1	1
$ev(X_3)$	0	0	0	0	1	1	1	1
$ev(f)$	0	1	1	0	0	1	1	0

Nous avons $\mathbb{F}_2^3 = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6, \alpha_7, \alpha_8\}$. et par exemple :

$$\begin{aligned} \alpha_4 &= (1, 1, 0) \\ ev(X_1 \cdot X_2) &= ev(X_1) \cdot ev(X_2) = (0, 0, 0, 1, 0, 0, 0, 1) \end{aligned}$$

2.2.3.1 Ordre lexicographique

En choisissant l'ordre lexicographique pour représenter les éléments de $\mathbb{F}_2^m = \{\alpha_1, \ldots, \alpha_{2^m}\}$, où α_i est l'écriture en binaire sur m bits de l'entier $i-1$. Dans cette écriture un entier $x \in [0, 2^m - 1]$ est écrit sous la forme :

$$x = (x_1, \ldots, x_m), x_i \ \in \ \mathbb{F}_2 \mid x = \sum_{i=1}^{m} x_i.2^{i-1}$$

- Pour $m = 1$ nous avons $\mathbb{F}_2 = \{0, 1\}$
- Notons $O_m = \{\alpha_1, \ldots, \alpha_{2^m}\}$
- Nous avons

$$O_{m+1} = \{(\alpha_1, 0), \ldots, (\alpha_{2^m}, 0), (\alpha_1, 1), \ldots, (\alpha_{2^m}, 1)\}$$

Puisque :
$(\alpha_i, 0)$ c'est l'entier $i-1$, alors $(\alpha_i, 1)$ c'est l'entier $i - 1 + 2^m$. Ainsi :

$$\begin{aligned}\{(\alpha_1, 0), \ldots, (\alpha_{2^m}, 0), (\alpha_1, 1), \ldots, (\alpha_{2^m}, 1)\} &= \{0, 1, \ldots, 2^m - 1, \ 2^m + 0 \ , \ \ldots, 2^m + 2^m - 1\} \\ &= \{0, 1, \ldots, 2^m - 1, \ 2^m \ , \ \ldots, 2^{m+1} - 1\} \\ &= O_{m+1}\end{aligned}$$

En choisissant l'ordre lexicographique pour représenter les éléments de $\mathbb{F}_2^m = \{\alpha_1, \ldots, \alpha_n\}, n = 2^m$, avec α_i comme l'écriture en binaire[3] sur m bits de l'entier $i-1$.

Les monômes de degrés 1 sont $X_1, \ldots, X_m$, et le vecteur associé au monôme X_i est une concaténation de 2^{m-i} blocs, chaque bloc formé de 2^{i-1} zéro suivi de 2^{i-1} un.

$$X_i = (\overbrace{\underbrace{0, \ldots, 0}_{2^{i-1}}, \underbrace{1, \ldots, 1}_{2^{i-1}}, \ldots\ldots, \underbrace{0, \ldots, 0}_{2^{i-1}}, \underbrace{1, \ldots, 1}_{2^{i-1}},}^{2^{m-i}}) \tag{2.3}$$

Pour les vecteurs associés aux monômes de degrés supérieurs, il faut calculer le produit des vecteurs associés à ses termes.

Exemple 2.2.4 *Pour* $m = 3, f = \mathbf{1} + X_2 + X_3 + X_1X_2 + X_1X_2X_3$.

$\mathbf{1}$	=	1	1	1	1	1	1	1	1
X_1	=	0	1	0	1	0	1	0	1
X_2	=	0	0	1	1	0	0	1	1
X_3	=	0	0	0	0	1	1	1	1
X_1X_2	=	0	0	0	1	0	0	0	1
$X_1X_2X_3$	=	0	0	0	0	0	0	0	1

[3] $(x_1, \ldots, x_m)_{LSB}$ l'écriture en binaire de x si $x = \sum_{i=1}^{m} x_i.2^{i-1}$, LSB : (Least significant bit) Bit de poids faible d'abord

$$\begin{aligned} f &= \mathbf{1} + X_2 + X_3 + X_1X_2 + X_1X_2X_3 \\ ev(f) &= (11111111) + (00001111) + (00110011) + (00010001) + (00000001) \\ &= (11010011) \end{aligned}$$

2.3 Les Codes de Reed-Muller

Il y a en fait deux approches pour décrire les codes de Reed-Muller, l'une dite à une variable qui correspond à la définition "cyclique" et l'autre à plusieurs variables qui correspond à la définition par fonctions booléennes. La correspondance entre ces deux approches a été développée par P. Charpin dans [5]. Nous utilisons la deuxième approche pour décrire ces codes.
Le code binaire de Reed-Muller d'ordre r en m variables, que l'on note $RM(r, m)$ est l'ensemble des tables de vérité des fonctions booléennes en m variables dont la forme algébrique normale (ANF) est de degré total au plus r.

La famille des codes RM est identifiée à l'aide de deux paramètres, usuellement notés r et m, appelés respectivement ordre et nombre de variables. Ces paramètres interviennent dans la description utilisant les fonctions booléennes.

On choisit un ordre quelconque sur les éléments de $\mathbb{F}_2^m = \{\alpha_1, \ldots, \alpha_{2^m}\}$. Une fonction f en m variables est alors identifiée au mot binaire[4] défini par :

$$c_f = ev(f) = (f(\alpha_1), \ldots, f(\alpha_{2^m}))$$

En d'autres termes, le mot du code est la liste des valeurs prises par la fonction dans un ordre choisi. Le code est défini de la façon suivante :

$$RM(r, m) = \{ev(f) : \deg(f) \leq r\}$$

Nous parlerons aussi du poids d'une fonction booléenne ; cela signifie bien évidemment le poids[5] du mot du code associé : $wt(f) = wt(ev(f))$. Dans tous ce qui suit : Pour simplifier les notations nous parlerons d'un mot f du code $RM(r, m)$, au lieu de dire $ev(f) \in RM(r, m)$, et nous utilisons $f(x)$ pour indiquer que nous traitons une fonction.

Définition 2.3.1 *Le code de Reed-Muller d'ordre r noté $RM(r, m)$, de longueur $n = 2^m$, pour $0 \leq r \leq m$ est l'ensemble des images des fonctions booléennes de degré maximal r en m variables.*

Historiquement, le code $RM(1, 5)$ a été utilisé par les sondes *Mariner* lancées par la NASA entre 1969 et 1973 pour assurer une transmission correcte des photos numérisées de Mars.

[4] Regarder définition-2.2.3 pour plus de détails sur cette écriture vectorielle (ev)
[5] Le poids de Hamming : définition-1.3.2

Théorème 2.3.1 *: Pour tout m, r, $0 \leq r \leq m$, le code de Reed-Muller d'ordre r, noté $RM(r,m)$, a pour longueur $n = 2^m$ pour dimension $k = 1+\binom{m}{1}+\cdots+\binom{m}{r}$ et pour distance minimale $d = 2^{m-r}$. C'est un code linéaire binaire.*

▷ Preuve : Le code $RM(r,m)$ est un sous-espace vectoriel de l'espace $\mathcal{F}_m$, l'ensemble des monômes de $\mathcal{M}$ (défini en 2.2) de degré inférieur ou égal à r, est une base de ce sous espace. Le cardinal de cet ensemble est la dimension du code, et il vaut $1+\binom{m}{1}+\cdots+\binom{m}{r}$. Soit $f_r^m \in RM(r,m),\ \deg(f_r^m) \leq r$, nous pouvons l'écrire sous la forme suivante :

$$f_r^m(x_1,\ldots,x_m) = f_r^{m-1}(x_1,\ldots,x_{m-1}) + x_m.f_{r-1}^{m-1}(x_1,\ldots,x_{m-1}) \tag{2.4}$$

Notez que le produit de deux fonctions $f,g \in \mathcal{F}_m$ est le produit dans $\mathbb{F}_2$ de leur évaluation, $(f.g)(x) = f(x).g(x)$. Ce formalisme est connu sous le nom de la ***construction Plotkin***, qui est à la base de plusieurs algorithmes de décodage récursifs. Nous avons trois propriétés concernant les fonctions de l'équation précédente :

1. $wt(f_r^m) = wt(f_r^{m-1}) + wt(f_r^{m-1} + f_{r-1}^{m-1})$
2. $wt(f_r^{m-1} + f_{r-1}^{m-1}) = wt(f_r^{m-1}) + wt(f_{r-1}^{m-1}) - 2.wt(f_r^{m-1}.f_{r-1}^{m-1})$
3. $wt(f_r^{m-1}.f_{r-1}^{m-1}) \leq min\{wt(f_r^{m-1}), wt(f_{r-1}^{m-1})\}$

En utilisant (2) et (3) dans (1) nous aurons :

$$\begin{array}{rcl} wt(f_r^m) & = & wt(f_r^{m-1}) + wt(f_r^{m-1} + f_{r-1}^{m-1}) \\ & = & 2.wt(f_r^{m-1}) + wt(f_{r-1}^{m-1}) - 2.wt(f_r^{m-1}.f_{r-1}^{m-1}) \\ & \geq & 2.wt(f_r^{m-1}) + wt(f_{r-1}^{m-1}) - 2.min\{wt(f_r^{m-1}, wt(f_{r-1}^{m-1})\} \\ & \geq & wt(f_{r-1}^{m-1}). \end{array} \tag{2.5}$$

Notons $d_{r,m}$ la distance minimale du code $RM(r,m)$, c'est un code linéaire, donc sa distance minimale est d'une façon équivalente définie comme le plus petit poids des mots non nuls de code, d'où nous réécrivons l'équation (2.5) à l'aide de la distance minimale de ces codes.

$$d_{r,m} \geq d_{r-1,m-1}.$$

Alors

$$d_{r,m} \geq d_{r-1,m-1} \geq d_{r-2,m-2} \cdots \geq d_{0,m-r}\ .$$

Le code $RM(0, m-r)$ est le code à répétition à deux éléments de longueur 2^{m-r}, donc sa distance minimale est exactement égale à sa longueur. Nous en déduisons que :

$$d_{r,m} \geq 2^{m-r} \tag{2.6}$$

Le mot de code correspondant au monôme $x_1x_2\ldots x_r$ est de poids 2^{m-r},

$$\begin{array}{rcl} wt(x_1x_2\ldots x_r) & = & \#supp(x_1x_2\ldots x_r) \\ & = & \#\{x \in \mathbb{F}_2^m \mid x_i = 1,\ 1 \leq i \leq r\} \\ & = & 2^{m-r} \end{array}$$

alors l'inégalité dans 2.6 est une égalité, ce qui prouve que la distance minimale de code $RM(r,m)$ est 2^{m-r} ◁

A partir de l'équation 2.4 nous pouvons définir récursivement les codes $RM(r,m)$ de la façon suivante :

$$\begin{array}{lcl} RM(0,m) & = & \{0\ldots0,1\ldots1\} \\ RM(m,m) & = & \mathbb{F}_2^n \\ RM(r,m) & = & \{(u,u+v)|u\in RM(r,m-1),v\in RM(r-1,m-1)\},0<r\leq m \end{array}$$

Une relation similaire permet de construire la matrice génératrice de $RM(r,m)$. Nous noterons cette matrice $G(r,m)$). Pour $0<r<m$, nous définissons :

$$G(r,m)=\begin{bmatrix} G(r,m-1) & G(r,m-1) \\ 0 & G(r-1,m-1) \end{bmatrix} \tag{2.7}$$

Pour $r=0$, $G(0,m)$ est la matrice $(1,2^m)$ suivante

$$G(0,m)=[11\ldots1]$$

et pour $r=m$ $G(m,m)$ est la matrice $(2^m,2^m)$ suivante

$$G(m,m)=\begin{bmatrix} G(m-1,m) \\ 00\ldots1 \end{bmatrix}$$

Propriété 2.3.1 *Les mots de code de Reed-Muller $RM(r,m)$, $\forall\ 0\leq r\leq m-1$ ont un poids pair.*

▷ Preuve :
Nous allons montrer que toute fonction booléenne a un support de cardinal pair. Prenons un monôme $x_{i_1}x_{i_2}\ldots x_{i_l}$, $l\leq m-1$ de la base [6] $\mathcal{M}$ de l'espace des fonctions $\mathcal{F}_m$, ce monôme vaut 1 pour tout

$$x=(x_1,\ldots,x_{i_1-1},1,x_{i_1+1},\ldots,x_{i_2-1},1,\ldots,x_{i_l-1},1,_{i_l+1},\ldots,x_m)\in\mathbb{F}_2^m$$

Donc le poids du mot associé à tout monôme est pair. Il suffit alors de prouver que la somme de deux mots de longueur n de poids pair est lui aussi un mot de poids pair. Soient $x,y\in\mathbb{F}_2^n$, deux mots de poids pair, supposons que l'intersection des supports de ces deux mots est de poids i, alors le poids de leurs somme $wt(x+y)=wt(x)-i+wt(y)-i=wt(x)+wt(y)-2.i$ qui est pair aussi. ◁

Théorème 2.3.2 *Le dual d'un code de Reed-Muller est un code de Reed-Muller :*

$$\begin{array}{lcl} RM(r,m)^\perp & = & RM(m-r-1,m)\ \ 0\leq r\leq m-1 \\ RM(m,m)^\perp & = & \{0\} \end{array}$$

[6] équation 2.2, page 32

▷ PREUVE : Les codes de Reed-Muller sont des sous espace vectoriels de $\mathbb{F}_2^n$, $n = 2^m$. Soient $f \in RM(r,m)$, $g \in RM(m-r-1,m)$. Alors $\deg(f) \leq r$, $\deg(g) \leq m-r-1$, et leur produit est de degré $m-1$ au plus, donc $fg \in RM(m-1,m)$, et d'après 2.3.1 fg a un poids pair, en conséquence :

$$< ev(f), ev(g) >= wt(fg) \pmod 2 \equiv 0$$

Donc $RM(m-r-1,m) \subset RM(r,m)^{\perp}$. Mais

$$\begin{array}{rcl} dim\ RM(r,m) \ + \ dim\ RM(m-r-1,m) & = & \sum_{i=0}^{r} \binom{m}{i} + \sum_{i=0}^{m-r-1} \binom{m}{i} \\ & = & \sum_{i=0}^{r} \binom{m}{i} + \sum_{i=0}^{m-r-1} \binom{m}{m-i} \\ & = & \sum_{i=0}^{r} \binom{m}{i} + \sum_{i=r+1}^{m} \binom{m}{m-i} \\ & = & 2^m \\ & = & n \end{array}$$

Ce qui implique que $RM(r,m)^{\perp} = RM(m-r-1,m)$
et comme $RM(m,m) = \mathbb{F}_2^n$, donc $RM(m,m)^{\perp} = \{0\}$ ◁

Quelques codes particuliers :

$RM(m,m)$	:	l'espace binaire $\mathbb{F}_2^n$ de dimension n sur $\mathbb{F}_2$
$RM(m-1,m)$	:	le code avec un seul bit de parité.
$RM(m-2,m)$	:	le code de Hamming étendu.
$RM(1,m)$	:	le code de Reed-Muller d'ordre 1.
$RM(0,m)$	:	le code à répétition.

2.3.1 Le code de Reed-Muller d'ordre 1

Le code de $RM(1,m)$ représente les éléments affines de l'espace ambiant : c'est l'ensemble des polynômes linéaires en $(X_1, \ldots, X_m)$ de $\mathcal{F}_m$, c'est-à-dire des fonctions de la forme :

$$a_0.\mathbf{1} + a_1.X_1 + \cdots + a_m.X_m, a_i \in \mathbb{F}_2$$

Une fonction d'encodage pour ce code peut-être :

$$\begin{array}{rcl} E_{RM_1} : (\mathbb{F}_2)^{m+1} & \rightarrow & \mathcal{F}_m \\ (a_0, \ldots, a_m) & \mapsto & a_0.\mathbf{1} + a_1.X_1 + \cdots + a_m.X_m \end{array}$$

Cette fonction est clairement injective, et comme $(\mathbb{F}_2)^{m+1}$ est de dimension $m+1$, ce code est donc un code de dimension $m+1$ et de longueur 2^m.

Exemple 2.3.1 *Prenons $m = 3$. En choisissant l'ordre lexicographique, une base de $\mathcal{F}_3$ est :*

$$\begin{array}{rcll} \delta_0 & = & \delta_{(0,0,0)} & = (1,0,0,0,0,0,0,0) \\ \delta_1 & = & \delta_{(1,0,0)} & = (0,1,0,0,0,0,0,0) \\ \delta_2 & = & \delta_{(0,1,0)} & = (0,0,1,0,0,0,0,0) \\ \delta_3 & = & \delta_{(1,1,0)} & = (0,0,0,1,0,0,0,0) \\ \delta_4 & = & \delta_{(0,0,1)} & = (0,0,0,0,1,0,0,0) \\ \delta_5 & = & \delta_{(1,0,1)} & = (0,0,0,0,0,1,0,0) \\ \delta_6 & = & \delta_{(0,1,1)} & = (0,0,0,0,0,0,1,0) \\ \delta_7 & = & \delta_{(1,1,1)} & = (0,0,0,0,0,0,0,1) \end{array}$$

qui est la base canonique de $\mathbb{F}_2^8 \equiv \mathcal{F}_3$

Nous allons maintenant coder le message :

$$x = (0, 1, 1, 0) \in \mathbb{F}_2^4.$$

La fonction affine correspondant à ce message est donc $f(x) = 0.\mathbf{1} + 1.X_1 + 1.X_2 + 0.X_3 = X_1 + X_2$. Pour calculer le mot du code correspondant à f nous allons calculer ses coordonnées dans la base $(\delta_i)_{i\in\{0...7\}}$. On a :

$$X_1 = \delta_1 + \delta_3 + \delta_5 + \delta_7$$

et

$$X_2 = \delta_2 + \delta_3 + \delta_6 + \delta_7$$

Par conséquent :

$$X_1 + X_2 = \delta_1 + \delta_2 + \delta_5 + \delta_6$$

et les coordonnées de $f(x)$ dans la base $(\delta_i)_{i\in\{0...7\}}$ sont donc :

$$(0, 1, 1, 0, 0, 1, 1, 0)$$

Nous pouvons obtenir ce résultat directement en calculant les valeurs que la fonction f prend pour tous les éléments de l'espace $\mathbb{F}_2^3$, de la façon suivante :

$$(f(0,0,0), f(1,0,0), f(0,1,0), f(1,1,0), f(0,0,1), f(1,0,1), f(0,1,1), f(1,1,1))$$

avec $f(x) = f(X_1, X_2, X_3) = X_1 + X_2$.

De même, une matrice génératrice de $\mathcal{C}$ est donnée par :

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

(la première ligne correspond aux coordonnées de $\mathbf{1}$ dans la base $(\delta_i)_{i\in\{0...7\}}$, la seconde aux coordonnées de X_1, la troisième aux coordonnées de X_2 et la dernière à celles de X_3). On vérifie que l'on a bien :

$$(0, 1, 1, 0) \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} = (0, 1, 1, 0, 0, 1, 1, 0)$$

Nous pouvons enfin écrire une matrice génératrice G_m d'un code de Reed-Muller d'ordre 1, $RM(1, m)$:

$$G_m = \begin{bmatrix} \mathbf{1} \\ X_1 \\ X_2 \\ \vdots \\ X_m \end{bmatrix}$$

Où les X_i sont les vecteurs associés aux monômes de degré 1, (voir 2.2.2, et l'équation 2.3)

2.3.2 Codes de Reed-Muller d'ordre r

Toute fonction $f \in \mathcal{F}_m$ peut s'écrire comme une fonction de m variables, en effet si $x \in \mathbb{F}_2^m$, on peut écrire :

$$x = (x_1, x_2, \ldots, x_m) = \sum_{i=1}^{m} x_i e_i.$$

avec $(e_i)_{i=1..m}$ la base canonique de $\mathbb{F}_2^m$.

$$e_i = (\underbrace{0, 0, \ldots, 1}_{i}, 0, \ldots, 0)$$

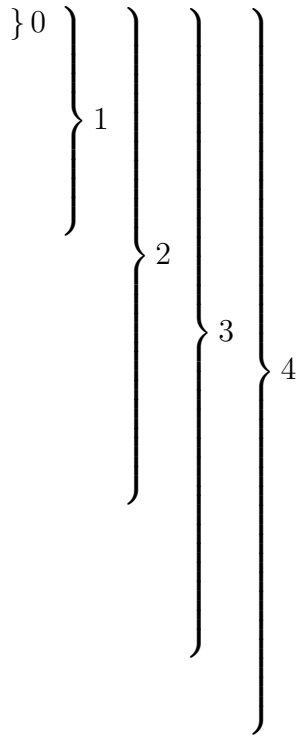
alors $f = f(x_1, x_2, \ldots, x_m)$. La fonction de projection X_i donne la valeur x_i pour $(x = \sum_{j=1}^{m} x_j e_j)$.

Nous donnons une autre définition des codes de Reed-Muller.

Définition 2.3.2 *Le code de Reed-Muller binaire d'ordre $r, 0 \leq r \leq m$, et de longueur $n = 2^m$, noté $RM(r, m)$, est l'ensemble des mots résultant de toutes les combinaisons linéaires des monômes $1, x_1, \ldots, x_m, x_1x_2, x_1x_3, \ldots$ (jusqu'au degré r). Le code $RM(r, m)$ est linéaire binaire, de dimension $\sum_{i=0}^{r} \binom{m}{i}$ et distance minimale 2^{m-r}.*

Exemple 2.3.2 *Pour $m = 4$, voici les matrices génératrices des codes de Reed-Muller d'ordre 1,2,3,4. Le nombre situé à droite de chaque accolade désigne l'ordre du code engendré par la matrice génératrice formée des vecteurs à gauche de cette dernière :*

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
x_1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
x_2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
x_3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
x_4	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
x_1x_2	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
x_1x_3	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	1
x_1x_4	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1
x_2x_3	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1
x_2x_4	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1
x_3x_4	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
$x_1x_2x_3$	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
$x_1x_2x_4$	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
$x_1x_3x_4$	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
$x_2x_3x_4$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
$x_1x_2x_3x_4$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1



Proposition 2.3.1 *Les codes de Reed-Muller réalisent la filtration suivante de l'espace des fonctions booléennes (donc de $\mathbb{F}_2^n$ également).*

$$0, 1 = RM(0, m) \subset RM(1, m) \subset \cdots \subset RM(m-1, m) \subset RM(m, m) = \mathbb{F}_2^n$$

▷ Preuve : Nous avons vu dans 2.2.2 que l'ensemble des monômes $\mathcal{M}$ (2.2) est une base de l'espace des fonctions, les vecteurs de cette base sont linéairement indépendants et les codes de Reed-Muller sont définis comme des sous espaces vectoriels engendrés par les vecteurs de la base de la façon suivante :

$$RM(r, m) = Vect(\{x_{i_1} \dots x_{i_l}\}_{1 \leq i_1 < \cdots < i_l \leq m,\ 0 \leq\ l\ \leq r})$$

D'où le résultat. ◁

Le groupe d'automorphismes des codes de Reed-Muller n'est pas trivial et il est utilisé dans l'algorithme de décodage étudié dans cette thèse.

2.4 Groupe d'automorphismes de $RM(r, m)$

Nous allons définir le groupe des transformations affines sur $\mathbb{F}_2^m$. Nous verrons ensuite qu'il constitue le groupe d'automorphismes des codes RM que nous notons $\mathcal{A}ut\,(RM(r, m))$ ([34] pp.398-405).

Définition 2.4.1 ·

- *On appelle $GL(m)$ le Groupe Linéaire Général d'ordre m, i,e le groupe de transformations linéaires inversibles de $\mathbb{F}_2^m$. Une fonction τ est dans $GL(m)$ si elle est de la forme $\tau(x) = Ax$, où A est une matrice de taille $m \times m$ inversible à coefficients dans $\mathbb{F}_2$.*
- *On appelle $GA(m)$ le Groupe Affine Général d'ordre m, c'est le groupe de transformations affines inversibles de $\mathbb{F}_2^m$. Une fonction τ est dans $GL(m)$ si elle est de la forme $\tau(x) = Ax + b$, où A est une matrice de taille $m \times m$ inversible à coefficients dans $\mathbb{F}_2$, et b est un élément de $\mathbb{F}_2^m$.*

Soient $A = (a_{ij})$ une (m, m)-matrice inversible, et b un vecteur binaire de $\mathbb{F}_2^m$. Définissons la transformation suivante sur l'ensemble des mots binaires de longueur m qui envoie 0 à b.

$$\begin{array}{rcl} \sigma : \mathbb{F}_2^m & \rightarrow & \mathbb{F}_2^m \\ x^t = (x_1, \dots, x_m)^t & \mapsto & A.x^t + b \end{array}$$

L'ensemble de toutes les transformations σ forme un groupe, non commutatif pour la composition, noté $GA(m)$[7], avec la composition comme opération du groupe. L'ordre de

[7] GA : Groupe général affine (General affine group)

ce groupe vaut :

$$|GA(m)| = 2^m(2^m-1)(2^m-2)(2^m-2^2)\dots(2^m-2^{m-1}).$$

C'est le nombre de possibilités dont on peut choisir les colonnes de la matrice A et le vecteur b. A est une matrice inversible donc elle est de rang m, et il y a (2^m-1) possibilités de choisir la première colonne, (2^m-2) possibilité pour la deuxième, et ainsi de suite. Pour le vecteur b nous avons 2^m possibilités. Une approximation de l'ordre est la suivante :

$$|GA(m)| \approx 0.29\ 2^{m^2+m}, \text{pour une grande valeur de } m.$$

Pour chaque transformation $\sigma \in GA(m)$

$$\begin{array}{rcl} \sigma : \mathbb{F}_2^m & \rightarrow & \mathbb{F}_2^m \\ \alpha & \mapsto & \sigma(\alpha), \end{array}$$

nous définissons une permutation associée π_σ sur l'espace $\mathbb{F}_2^n$

$$\begin{array}{rcl} \pi_\sigma : \mathbb{F}_2^n & \rightarrow & \mathbb{F}_2^n \\ x = (x_{\alpha_1}, \dots, x_{\alpha_n}) & \mapsto & (x_{\sigma(\alpha_1)}, \dots, x_{\sigma(\alpha_n)}) \end{array}$$

Nous allons démontrer que les transformations affines forment le groupe d'automorphismes des codes de Reed-Muller.

Nous avons le résultat suivant sur les groupes d'automorphismes de ces codes.

Théorème 2.4.1 *Le groupe d'automorphismes des codes RM vérifie :*

$$\mathcal{A}ut\left(RM(r,m)\right) \subseteq \mathcal{A}ut\left(RM(r+1,m)\right), 1 \leq r \leq m-2$$

▷ Preuve : Nous allons démontrer le théorème par récurrence sur r :

– il est vrai pour $r = 1$:
Soit P une permutation du groupe d'automorphismes de $RM(1,m)$, donc pour tous les monômes de degré un, $\{x_i\}_{i=1\dots m}$ de la matrice génératrice du code, nous avons :

$$x_i \in RM(1,m) \Rightarrow x_iP \in RM(1,m) = Vect(1, x_1, \dots, x_m).$$

Pour les éléments $\{x_{i_1}x_{i_2}\}$, nous avons :

$$(x_{i_1}x_{i_2})P = (x_{i_1}P)(x_{i_2}P) \in RM(2,m).$$

Alors :

$$\forall c \in RM(2,m), cP \in RM(2,m) \Rightarrow P \in \mathcal{A}ut\left(RM(2,m)\right)$$

D'où :

$$\mathcal{A}ut\left(RM(1,m)\right) \subseteq \mathcal{A}ut\left(RM(2,m)\right).$$

– Nous le supposons vrai pour r.

– Nous allons démontrer qu'il est vrai pour $r+1$:
Soit P une permutation du groupe d'automorphismes de $RM(r,m)$, donc pour tous les monômes de degré un, $\{x_i\}_{i=1\ldots m}$ de la matrice génératrice du code, nous avons :

$$x_i \in RM(r,m) \Rightarrow x_i P = f_i \in RM(r,m).$$

Pour les éléments $\{x_{j_1} \ldots x_{j_l}\}, 1 < l \leq r$, nous avons :

$$(x_{j_1} \ldots x_{j_r})P = (x_{j_1}P) \ldots (x_{j_r}P) = f_{j_1} \ldots f_{j_r} \in RM(r,m).$$

Supposons que P n'est pas dans le groupe d'automorphismes du code $RM(r+1,m)$, donc il existe un monôme v d'ordre-$(r+1)$ de la matrice génératrice de $RM(r+1,m)$, tel que : $vP \notin RM(r+1,m)$.

$$\exists\{i_1,\ldots,i_{r+1}\} \mid (x_{i_1} \ldots x_{i_{r+1}})P, d'ordre \geq r+2, \text{pour } r+2 \leq m$$

Mais $P \in \mathcal{A}ut\,(RM(r,m))$, alors pour tout $\{j_1,\ldots,j_r\} \subset \{i_1,\ldots,i_{r+1}\}$:

$$(x_{j_1} \ldots x_{j_r})P = f_{j_1} \ldots f_{j_r} \in RM(r,m)$$

Ainsi $\forall i \in \{i_1,\ldots,i_{r+1}\}$:

$$f_{i_1} \ldots f_{i_{r+1}} = f_i.(\tilde{f_i}),\ \deg(\tilde{f_i}) = r,\ \deg(f_i.\tilde{f_i}) = r+2.$$

Donc $\forall i \in \{i_1,\ldots,i_{r+1}\}$, f_i a un monôme de degré au moins deux contenant deux variables différentes des variables du monôme de grand degré r de $\tilde{f_i}$. Ceci n'est pas possible puisque dans ce cas $f_{i_1} \ldots f_{i_r}$ contiendra un monôme de degré 2 r. Ceci prouve que :

$$P \in \mathcal{A}ut\,(RM(r+1,m))\,.$$

◁

Théorème 2.4.2 *Pour $1 \leq r \leq m-2$ le groupe d'automorphismes $\mathcal{A}ut\,(RM(r,m))$ est le groupe de transformation affine $GA(m)$.*

$$\mathcal{A}ut\,(RM(r,m)) = GA(m)$$

▷ Preuve :
Soient $f \in RM(r,m)$, et $\sigma \in GA(m)$. Calculons $f \circ \sigma$

$$f \circ \sigma = f(\sum a_{1j}x_j + b_1, \ldots, \sum a_{mj}x_j + b_m).$$

donc $f \circ \sigma$ aussi est une fonction de degré maximal r, nous avons donc l'inclusion suivante

$$GA(m) \subset \mathcal{A}ut\,(RM(r,m)) \tag{2.8}$$

Notons G la matrice génératrice du code $RM(1,m)$ et prenons une permutation P du Groupe d'automorphismes de ce code. D'après le lemme 1.4.1 il existe une matrice inversible H, vérifiant $HG = GP$. Nous allons écrire cette matrice de la façon suivante :

$$H = \begin{pmatrix} h_{00} & V_2 \\ V_1^T & B \end{pmatrix}, h_{00} \in \mathbb{F}_2, V_1, V_2 \in \mathbb{F}_2^m, B \in M(\mathbb{F}_2)_{m,m}$$

Prenons le message $(a_0 a)$, avec $a_0 \in \mathbb{F}_2, a \in (\mathbb{F}_2)^m$; le mot du code correspondant c'est $f_{(a_0 a)}(x) = a_0 + a.x^T$; le mot permuté par P est $f_{(\tilde{a}_0 \tilde{a})}(x) = \tilde{a}_0 + \tilde{a}.x^T$. Nous avons :

$$\begin{array}{rcl} ev(\ f_{(a_0 a)}\ ) & = & (a_0 a)G \\ ev(\ f_{(\tilde{a}_0 \tilde{a})}(x)\ ) & = & (a_0 a)GP \\ & = & (a_0 a)HG \\ & = & (\tilde{a}_0 \tilde{a})G \end{array}$$

Nous avons les deux égalités suivantes :

$$\begin{array}{c} \tilde{a}_0 = a_0 h_{00} + a.V_1^T \\ \tilde{a} = a_0 V_2 + a.B. \end{array}$$

Mais comme la permutation du mot tout à un **1** est lui même, cela induit que :

$$h_{00} = 1, V_2 = 0$$

et la fonction permutée s'écrit :

$$f_{(\tilde{a}_0 \tilde{a})}(x) = a_0 + a.V_1^T + a.B.x^T = f_{(a_0 a)}(x.B^T + V_1)$$

donc la permutation P est une permutation du groupe $GA(m)$. Nous avons

$$\forall P \in \mathcal{A}ut\,(RM(1,m)) \Rightarrow P \in GA(m).$$

D'ou :

$$\mathcal{A}ut\,(RM(1,m)) \subset GA(m) \tag{2.9}$$

En utilisant 2.8 pour $r = 1$, et 2.9, nous avons :

$$\mathcal{A}ut\,(RM(1,m)) \equiv GA(m)$$

Ce résultat et le théorème 2.4.1 nous donne que :

$GA(m) \equiv \mathcal{A}ut\,(RM(1,m)) \subset \mathcal{A}ut\,(RM(2,m)) \subset \cdots \subset \mathcal{A}ut\,(RM(m-2,m)) \equiv GA(m)$

car $\mathcal{A}ut\,(RM(m-2,m)) = \mathcal{A}ut\left(RM(m-2,m)^{\perp}\right) = \mathcal{A}ut\,(RM(1,m))$ ◁

Note 3 *Le groupe d'automorphismes du code $RM(m-1,m)$ (le code avec un seul bit de parité) est le groupe symétrique $\mathcal{S}_n$ de toutes les permutations de n-symboles.*

$$\mathcal{S}_n = \mathcal{A}ut\,(RM(0,m)) = \mathcal{A}ut\left(RM(0,m)^{\perp}\right) = \mathcal{A}ut\,(RM(m-1,m))$$

et comme le code d'ordre m est tout l'espace vectoriel $\mathbb{F}_2^n$, nous avons aussi :

$$\mathcal{A}ut\,(RM(m,m)) = \mathcal{S}_n$$

Chapitre 3

Décodage des Codes de Reed-Muller

Contenu

Décoder un mot erroné d'un code quelconque consiste à trouver un des mots les plus proches pour une certaine métrique. C'est ce que nous appelons le principe du ***décodage à distance minimale***. C'est le principe qui nous guidera dans nos procédures de décodage. Le décodage à distance minimale par énumération de tous les mots du code, reste possible pour les codes de petite dimension. A partir d'une certaine borne, qui dépend bien évidemment de la puissance de calcul des ordinateurs, un tel décodage à distance minimale devient impossible. Nous aimerions concevoir un algorithme

de décodage rapide qui approche, le plus, le résultat de décodage à distance minimale. Nous désignons un tel algorithme par le terme ***optimal***.

Nous décrivons le problème de décodage à maximum de vraisemblance MLD et l'algorithme de décodage FHT qui le fait pour les codes RM de premier ordre. Nous présenterons l'algorithme de décodage "Vote Majoritaire", qui a été développé par I.S. Reed [43], et a été suivi de plusieurs améliorations. La complexité d'un tel algorithme est de l'ordre de nk, (n=longueur de code, k = la dimension). Nous exposerons l'algorithme de décodage à maximum de vraisemblance "maximum-likelihood (ML)" conçu à l'aide de la structure des treillis à plusieurs niveaux [22], il a une complexité exponentielle en la longueur du code ce qui limite l'utilisation de l'algorithme. Les techniques de décodage récursif (voir [32],[11],[7],[9],[8],[10],[14]) sont basées sur la construction de Plotkin $(u, u+v)$, qui nous permet de décomposer $RM(r,m)$ en des codes plus courts [34]. Ces algorithmes permettent l'encodage récursif avec une complexité de l'ordre de $n\min(r, m-r)$. Rappelons que l'encodage en utilisant la matrice génératrice coûte nk pour $k = 1 + \binom{m}{1} + \cdots + \binom{m}{r}$.

3.1 La dérivée discrète

Nous allons définir l'opérateur de la dérivée discrète pour une fonction booléenne. Cette opérateur va être utilisé pour décrire toutes les techniques de décodage des codes RM.

Définition 3.1.1 *Soient $f \in \mathcal{F}_m, \alpha \in \mathbb{F}_2^m$. On appelle $D_\alpha f$ la dérivée discrète de la fonction f dans la direction α. La dérivée est définie par la formule suivante :*

$$D_\alpha f(x) \stackrel{\triangle}{=} f(x) + f(x+\alpha).$$

Rappelons que ev (l'écriture vectorielle des fonctions) est un isomorphisme d'espaces vectoriels de $\mathcal{F}_m$ dans $\mathbb{F}_2^n$:

$$\begin{array}{lcl} ev(f) & \stackrel{\triangle}{=} & (f(\alpha_1), \ldots, f(\alpha_n)) \\ ev(D_\alpha f) & = & (f(\alpha_1) + f(\alpha_1+\alpha), \ldots, f(\alpha_n) + f(\alpha_n+\alpha)), \end{array}$$

pour $\{\alpha_1, \ldots, \alpha_n\} = \mathbb{F}_2^m$.

Exemple 3.1.1 *Pour $m = 3$, nous avons $f \in \mathcal{F}_m \equiv \mathbb{F}_2^8$, et pour $f : \mathbb{F}_2^3 \mapsto \mathbb{F}_2$*

	α_1	α_2	α_3	α_4	α_5	α_6	α_7	α_8
$x = (x_1, x_2, x_3)$	$(0,0,0)$	$(1,0,0)$	$(0,1,0)$	$(1,1,0)$	$(0,0,1)$	$(1,0,1)$	$(0,1,1)$	$(1,1,1)$
$f(x) = x1.x2$	0	0	0	1	0	0	0	1
$D_{\alpha_2} f(x) = x_2$	0	0	1	1	0	0	1	1

$ev(f) = (f(\alpha_1), \ldots, f(\alpha_n)) = (0,0,0,1,0,0,0,1)$
$ev(D_{\alpha_2} f) = (0,0,1,1,0,0,1,1)$

Remarquons que dans l'exemple précédent la fonction f est d'ordre 2 et que sa dérivée est d'ordre 1. Cette propriété de diminution d'ordre associée à l'opérateur de la dérivée sur les fonctions polynomiales réelles reste vraie pour la dérivée discrète.

Propriété 3.1.1 *Soit $f : \mathbb{F}_2^m \mapsto \mathbb{F}_2$, une fonction binaire d'ordre r, la dérivée discrète $D_\alpha f$ est une fonction binaire d'ordre $r-1$.*

▷ Preuve : La propriété est vraie pour tous les éléments de la base $\mathcal{M}$ de l'espace $\mathcal{F}_m$, rappelons que

$$\mathcal{M} = \{\mathbf{1} \cdot x_{i_1} \cdot x_{i_2} \dots x_{i_k} \,|\, 0 \leq i_1 < i_2 \cdots < i_k \leq m, \text{ pour } 0 \leq k \leq m\},$$

La dérivée discrète d'un monôme de degré r est une somme de monômes de degré au plus $r-1$.

$$\forall \alpha \in \mathbb{F}_2^m,\ D_\alpha(x_{i_1}x_{i_2}\dots x_{i_k}) = x_{i_1}x_{i_2}\dots x_{i_k} + (x_{i_1}+(\alpha)_{i_1})\cdot(x_{i_2}+(\alpha)_{i_2})\dots(x_{i_k}+(\alpha)_{i_k})$$

Donc la propriété est vérifiée pour toute fonction binaire de l'espace des fonctions. ◁

3.2 Décodage à maximum de vraisemblance

Le moyen le plus évident pour décoder un mot de façon optimale est de comparer le mot reçu avec tous les mots de code, c'est ce qu'on appelle algorithme énumératif. Pour les codes de grande dimension appliquer un tel algorithme n'est pas pratique et même infaisable. Pour un code binaire $\mathcal{C}[n,k,d]$, le coût de décodage est $n2^k$ opérations sans compter le coût d'encodage des mots de code, en supposant qu'il est possible de stocker tous les mots de code.

Exemple 3.2.1 *Le code $RM(1,m) = \mathcal{C}[2^m, m+1, 2^{m-1}]$: Pour $m = 16$, l'algorithme énumératif nécessite $2^{33} \approx 8.10^9$ opérations qu'il faut multiplier par $m2^m$, qui est le coût d'encodage des mots du code, car on ne peut pas les garder en mémoire.*
Le code $RM(2,m) = \mathcal{C}[2^m, 1+m+\frac{m(m-1)}{2}, 2^{m-1}]$. Pour $m = 10$ l'algorithme énumératif nécessite $10.2^{76} \approx 7.10^{23}$, ce n'est même pas à la porté des machines dans un proche futur.

Soit $\mathcal{C}$ un code de longueur n sur un alphabet Σ, et soit $c = (c_1, \dots, c_n)$ un mot du code $\mathcal{C}$. Ayant transmis c, nous recevons le vecteur $y = (y_1, \dots, y_n)$. L'entrée du canal de transmission bruité est un mot de code de longueur n. Nous considérons l'entrée comme un vecteur aléatoire X, avec une distribution de probabilité définie pour les mots de code par $\Pr[X = c] = \Pr[c]$. Chaque vecteur d'entré X induit un vecteur de sortie Y avec une distribution de probabilité définie par la formule :

$$\Pr[Y = y] = \sum_{c \in \mathcal{C}} \Pr[Y = y | X = c] \Pr[X = c]$$

Décoder de façon optimale c'est trouver le mot de code c qui maximise la probabilité que c soit transmis sachant que y est reçu ($\Pr[c|y] \triangleq \Pr[X = c|Y = y]$). Un décodeur qui trouve le mot de code le plus probable, ou un des mots les plus probables s'il y en a plusieurs, est appelé décodeur à maximum de vraisemblance (voir 1.5 définition 1.5.1) .

$$MLD(y) \triangleq \{c \in \mathcal{C} \mid \Pr[c|y] \geq \Pr[\tilde{c}|y] \ \forall \ \tilde{c} \in \mathcal{C}\} \tag{3.1}$$

Rappelons la loi de Bayes concernant la probabilité conditionnelle :

$$\Pr[y|c] = \frac{\Pr[y]}{\Pr[c]} \Pr[c|y] \,. \tag{3.2}$$

Nous supposons que les mots du code $\mathcal{C}$ ont la même probabilité qu'ils soient transmis :

$$\Pr[X = c] = \frac{1}{|\mathcal{C}|} \ .$$

Donc, selon (3.2) pour un vecteur reçu y, le mot de code c qui maximise la probabilité $\Pr[y|c]$ est celui recherché par le décodeur à maximum de vraisemblance pour maximiser la probabilité $\Pr[c|y]$. Notez bien que nous n'avons pas besoin de calculer la valeur $\Pr[y]$ qui d'ailleurs n'est pas facile à trouver. Ainsi nous réécrivons l'équation (3.1)

$$MLD(y) \equiv \{c \in \mathcal{C} \mid \Pr[y|c] \geq \Pr[y|\tilde{c}] \ \forall \ \tilde{c} \in \mathcal{C}\}. \tag{3.3}$$

Étudions maintenant cette probabilité $\Pr[y|c]$, pour un canal de transmission sans mémoire, donc $\forall c \in \mathcal{C}$ nous pouvons écrire :

$$\Pr[y|c] = \prod_{i=1}^{n} \Pr[y_i|c_i] \,. \tag{3.4}$$

Remarque 2 *Écrire la probabilité sous une forme de produit est possible puisque le canal est sans mémoire, mais nous ne pouvons pas écrire* $\Pr[c|y]$ *comme un produit* $\prod_{i=1}^{n} \Pr[X_i = c_i|Y_i = y_i]$ *puisque les symboles des mots de code sont liés par les équations de parité, donc ne sont pas indépendants.*

Une explication de la dernière remarque : nous ne pouvons pas écrire la probabilité $\Pr[X = c]$, qui est supposée uniforme et vaut $\frac{1}{|\mathcal{C}|}$ comme produit de probabilités $\prod_{i=1}^{n} \Pr[X_i = c_i]$, qui n'est pas uniforme.

Le bruit est une réalisation d'une variable aléatoire indépendante ; cette variable suit une loi de probabilité connue, donc nous avons la probabilité conditionnelle $\Pr[y_i|\sigma]$ de recevoir le symbole y_i pour tous les symboles d'entrée $\sigma \in \Sigma$.

En utilisant 3.4 et en prenant le log de l'équation 3.4, nous écrivons l'algorithme suivant :

Algorithme 1 . **MLD**
Entrée : $y = (y_1, \ldots, y_n) \in \Sigma^n$
Sortie : $\tilde{c} \in \mathcal{C}$.

1. Pour tout $c \in \mathcal{C}$ calculer :

$$d_{ML}(y, c) \stackrel{\triangle}{=} -\log(\Pr[y|c]) = -\sum_{i=1}^{n} \log(\Pr[y_i|c_i])$$

2. Choisir $\tilde{c}$ qui minimise la distance $d_{ML}(y, c)$.

Note 4 *Dans un canal binaire symétrique* ***BSC***[1] *le décodage à maximum de vraisemblance est un décodage à distance minimale* ***MDD***[2]*, puisque dans ce cas* $\Pr[0|1] = \Pr[1|0] = p$ *et pour tous les mots de code* c *si on note* d_H *la distance de Hamming, le terme qu'on cherche à minimiser dans l'algorithme1* ***MLD*** *s'écrit :*

$$d_{ML}(y, c) = -n.\log(1-p) + \log\left(\frac{1-p}{p}\right).d_H(y, c)$$

Note 5 *Dans un canal Gaussien* ***AWGN***[3] *le décodage à maximum de vraisemblance est un décodage à distance minimale* ***MDD*** *voir [51] et [4].*

L'algorithme-1 est un algorithme énumératif. Nous allons voir son utilité quand nous traiterons les treillis 3.2.3, nous verrons que l'algorithme de Viterbi n'est d'autre que l'application de MLD à un graphe représentant tous les mots du code RM.

3.2.1 Borne de décodage théorique à maximum de vraisemblance

Avec une grande probabilité, un algorithme de décodage quelconque ne pourra pas corriger, un mot de code erroné par une erreur aléatoire au delà de la capacité de correction d'un algorithme de décodage à maximum de vraisemblance.

Pour les codes RM, cette borne a été établie par Sidel'nikov et Pershakov dans leur article ([50]), nous allons la présenter. Nous allons d'abord établir cette borne pour un

[1] Regarder 1.2.1
[2] Définition 1.5.1
[3] Regarder 1.2.2

code quelconque $\mathcal{C}$ en fonction de sa distribution de poids et ensuite la déterminer pour le code $RM(r,m)$.

Définition 3.2.1 *Soit $\mathcal{C}$ un code linéaire binaire. Nous disons qu'un vecteur d'erreur $e \in \mathbb{F}_2^n$ est trompeur pour le code $\mathcal{C}$, si il existe un mot de code $x \in \mathcal{C}$ tel que $d(e,x) \leq wt(e)$*

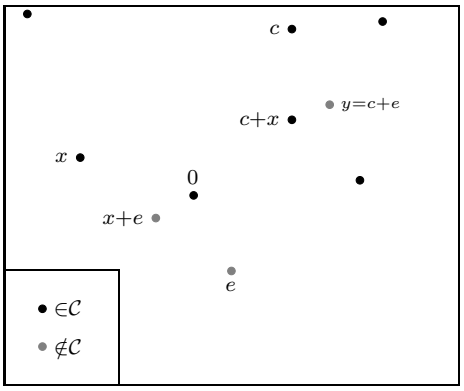


FIG. 3.1 – Vecteur trompeur pour un code linéaire $\mathcal{C}$

Cela veut dire qu'un algorithme MLD n'arrivera pas à décoder un mot du code erroné avec un vecteur trompeur. Puisque, si on transmet $c \in \mathcal{C}$, et on reçoit le vecteur $y = c+e$, Nous avons

$$d(c+x,y) = d(x,e) \leq wt(e) = d(c,y)$$

Ainsi l'algorithme MLD ne corrigera certainement pas y (voir figure-3.1).

Notons $r_t(\mathcal{C})$ Le nombre de vecteurs trompeurs de poids t pour le code $\mathcal{C}$. Le vecteur d'erreur e est une réalisation d'une variable aléatoire uniforme dans l'ensemble des vecteurs de poids t. Notons $P_{er}(\mathcal{C},t)$ la probabilité qu'un algorithme MLD se trompe en décodant un mot erroné par une erreur de poids t, nous avons intuitivement la borne suivante :

$$P_{er}(\mathcal{C},t) \leq \frac{r_t(\mathcal{C})}{\binom{n}{t}}.$$

Dans le but de trouver une borne asymptotique, nous ferons l'utilisation fréquente des inégalités suivantes (voir A.3.1, [41] page-655) :

$$\binom{n}{k} < \frac{1}{\sqrt{2\pi k(1-k/n)}} 2^{nH(\frac{k}{n})} \tag{3.5}$$

$$H\left(\frac{1-x}{2}\right) < 1 - \frac{x^2}{2ln(2)}, \tag{3.6}$$

où log est la fonction de logarithme à base 2, ln la fonction de logarithme népérien, et H est la fonction d'entropie $H(x) = -x\log(x) - (1-x)\log(1-x)$. Nous avons aussi besoin de borner une somme de coefficients binomiaux ; nous définissons l'inégalité suivante :

Proposition 3.2.1 *Soient $n, s \in \mathbb{R}$ avec $0 < s \leq n$, $\exists\, C \in \mathbb{R}$, tel que $\forall t \leq s$:*

$$\sum_{j=s/2}^{t} \binom{s}{j} \leq C\sqrt{n}\binom{s}{s/2}$$

▷ Preuve : La démonstration utilise le théorème central limite et l'approximation gaussienne Annexe-A.3.1 : Soit X une variable aléatoire de loi binomiale $B(s, \frac{1}{2})$. Le théorème central limite spécifie que lorsque s tend vers l'infini, la loi de la variable X tend vers la loi normale $\mathcal{N}(\mu = \frac{s}{2}, \sigma^2 = \frac{s}{4})$. En utilisant l'inégalité de Chebyshev A.3.2, nous déduisons qu'il existe C tel que :

$$\sum_{j=s/2}^{t} \Pr\left[X = j\right] \leq 2\ C\ \sigma \Pr\left[X = \frac{s}{2}\right]$$

ayant $\sigma = \frac{\sqrt{s}}{2} \leq \frac{\sqrt{n}}{2}$ et $\Pr\left[X = j\right] = \binom{s}{j}\frac{1}{2^s}$, d'où :

$$\sum_{j=s/2}^{t} \binom{s}{j} \leq C\sqrt{n}\binom{s}{\frac{s}{2}}.$$

◁

Nous cherchons une borne supérieure de la quantité $r_t(\mathcal{C})$. Nous pouvons ensuite trouver une borne pour la probabilité d'erreur de décodage bornée d'un algorithme MLD. Soit $c \in \mathbb{F}_2^n,\ wt(c) = s$; le nombre de vecteurs e de poids t vérifiant $d(e, c) \leq t$ est donné par la formule :

$$H(t, s) = \sum_{\frac{s}{2} \leq i \leq t} \binom{s}{i}\binom{n-s}{t-i}$$

L'explication de la dernière formule est illustrée graphiquement dans la figure 3.2.

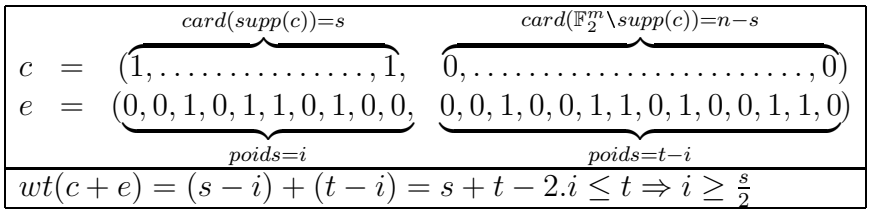


Fig. 3.2 – Condition sur le vecteur trompeur (e) de poids t, pour un mot de code c de poids s, (s est pair voir la propriété 2.3.1)

Lemme 3.2.1 *Soit A_s la distribution du poids du code $\mathcal{C}$; le nombre de vecteurs trompeurs vérifie :*

$$r_t(\mathcal{C}) \leq \sum_{c\in\mathcal{C},\ c\neq 0} H(t, wt(c)) = \sum_{2t\geq s>0} A_s H(t,s).$$

▷ Preuve : Soit $c \in \mathcal{C},\ wt(c) = s$, le nombre de vecteurs trompeurs de poids t pour le mot c est $H(t,s)$. A partir de la figure-3.2, nous avons $H(t,s) = 0\ \forall s > 2t$. Sachant que le nombre des mots de code de poids s est A_s :

$$A_s = \#\{c \in \mathcal{C} \mid wt(c) = s\},$$

nous écrivons :

$$\begin{array}{rcl} \sum_{c\in\mathcal{C},\ c\neq 0} H(t, wt(c)) & = & \sum_{c\in\mathcal{C}, 2t\geq wt(c)>0} H(t, wt(c)) \\ & = & \sum_{2t\geq s>0} A_s H(t,s). \end{array}$$

◁

Nous allons maintenant utiliser ces résultats pour le code de Reed-Muller afin d'établir une borne de décodage à maximum de vraisemblance.

Théorème 3.2.1 *([50]) La probabilité d'erreur de décodage à maximum de vraisemblance pour les codes de Reed-Muller $RM(r,n)$, avec $r = const$ et $\lambda > 0, t = \frac{n}{2}(1 - \frac{\lambda}{\sqrt{n}})$ erreurs, vérifie :*

$$P_{er}(RM_r, t) \leq C\ 2^{-\frac{\lambda^2}{2ln2(2^r-1)} + \sum_{j=0}^{r}\binom{n}{j}}.$$

Pour $\lambda = 2((2^r-1)ln2\sum_{j=0}^{r}\binom{m}{j})^{\frac{1}{2}}$, nous avons

$$P_{er}(RM_r, t) \leq C\ 2^{-\sum_{j=0}^{r}\binom{m}{j}} = C|RM_r|^{-1}.$$

▷ Preuve : La distribution des poids des codes de Reed-Muller $RM(r,m)$ n'est pas connue en général, mais la distance minimale de ce code est de $2^{m-r} = n2^{-r}$, donc

$$A_s = 0, \text{ si } s \in]0, n2^{-r}[$$

. Mais c'est un code linéaire, et le mot $\mathbf{1}$ est un mot de code, alors :

$$A_s = 0, \text{ si } s \in]n(1-2^{-r}), n[$$

Ces résultats et le lemme 3.2.1, nous permet d'écrire :

$$r_t(RM_r) \leq (2^{\sum_{j=0}^{r}\binom{m}{j}}) \max_{s\in[n2^{-r}, n(1-2^{-r})]} H(t,s)$$

Le premier facteur est le nombre des mots de code.
Pour $t = \frac{n}{2}(1 - \frac{\lambda}{\sqrt{(n)}})$, Nous écrivons $H(t,s)$ de la façon suivante :

$$\begin{array}{rcl} H(t,s) & = & \sum_{\frac{s}{2} \leq i \leq t} \binom{s}{i}\binom{n-s}{t-i} \\ & \leq & \binom{n-s}{t-s/2} \sum_{j=s/2}^{t} \binom{s}{j} \\ & \leq & C_1\sqrt{n}\binom{s}{s/2}\binom{n-s}{t-s/2} \end{array}$$

Ici nous avons utiliser l'inégalité 3.2.1 :
Notez que le terme $H(t,s)$ est maximum pour $s = n\ 2^{-r}$.
A l'aide des équations 3.5 et 3.6 nous déduisons :

$$\begin{array}{rcl} \binom{s}{s/2} & < & \sqrt{\frac{2}{\pi s}}\ 2^s < \sqrt{\frac{2^{r+1}}{\pi n}} 2^s, \text{ pour } 2^{-r}n \leq s \leq n(1-2^{-r}) \\ \binom{n-s}{t-s/2} & < & \frac{1}{\sqrt{\pi n}}\ 2^{n-s-\frac{\lambda^2 n}{(n-s)^2}}, \text{ avec } t = \frac{n}{2}(1 - \frac{\lambda}{\sqrt{(n)}}) \ . \end{array}$$

Pour un taux d'erreurs proche de la moitié de la longueur du code nous pouvons écrire l'égalité suivante :

$$\binom{n}{t} \approx \frac{C_2}{\sqrt{n}}\ 2^{n - \frac{\lambda^2}{2\ ln(2)}} \ .$$

Alors la probabilité d'erreur est majorée par le terme suivant :

$$P_{er}(RM_2, t) \leq \frac{r_t(RM_2)}{\binom{n}{t}} \leq C\ 2^{\frac{\lambda^2}{2}(1 - \frac{n}{n-s})} . 2^{\sum_{j=0}^{r}\binom{m}{j}} \ .$$

◁

Concernant les codes de Reed-Muller d'ordre deux, nous déduisons qu'un algorithme de décodage à distance minimale peut décoder avec une grande probabilité une erreur de poids inférieur de $t = \frac{n}{2}(1 - \sqrt{\frac{6\ ln(2)\ m^2}{n}})$. Il a été prouvé dans [53] que la séquence des codes de Reed-Muller d'ordre deux est asymptotiquement optimal comme code linéaire binaire.

Définition 3.2.2 *Pour une transmission sur un canal sans mémoire BSC, en appliquant le théorème précédent, nous définissons les bornes de décodage asymptotique à maximum de vraisemblance (Minimum distance aussi pour BSC) pour les codes RM d'ordre* 1 *et* 2 *respectivement :*

$$\begin{array}{rcl} ML_1 & = & \frac{n}{2}\left(1 - \sqrt{4\ln(2)\frac{m}{n}}\ \right) \\ ML_2 & = & \frac{n}{2}\left(1 - \sqrt{12\ \ln(2)\frac{m^2}{n}}\ \right) \end{array}$$

En 2003 Helleseth, Kløve et Levenshtein [53] ont prouvé que pour les codes $RM(2,m)$ la borne de décodage asymptotique ML_2 est stricte. Donc un algorithme de décodage à maximum de vraisemblance a une probabilité non négligeable de ne pas retourner le mot transmis si le poids du vecteur d'erreur est plus grand que ML_2.

3.2.2 Algorithme de décodage à maximum de vraisemblance Pour $RM(1, m)$

Nous nous proposons maintenant d'étudier l'algorithme de décodage à maximum de vraisemblance présenté dans [2]. Il présente une accélération pour le décodage à distance minimale pour les codes de Reed-Muller d'ordre-1, ce qui veut dire que c'est un algorithme énumératif optimisé.

Nous nous limitons à étudier deux algorithmes de l'article [2], l'algorithme d'encodage et le premier algorithme de décodage (Simple FHT). Vous pouvez trouvez dans ce même article, plusieurs algorithmes de décodage à maximum de vraisemblance optimisés, pour des erreurs de poids bornés.

3.2.2.1 Algorithme d'encodage de $RM(1, m)$

Nous allons étudier dans ce paragraphe l'algorithme d'encodage de Reed-Muller code d'ordre 1. Le but, de ce paragraphe est de bien détailler l'algorithme d'encodage présenté dans l'article [2].

Considérons le vecteur d'information $u = (u_0, \ldots, u_m)$, l'algorithme engendre le vecteur $v = (v_1, \ldots, v_n)$, dans l'alphabet $\{-1, 1\}$.

Algorithme 2 Encodage de RM d'ordre 1.
Entrée : $u = (u_0, u_1, \ldots, u_m)$.
Sortie : $v = (v_1, \ldots, v_n), v_i \in \{-1, 1\}$.

1. posons $v^{(1)} = \Big(\ 1, (-1)^{u_1} \Big)$.
2. pour $i = 2$ à m faire $v^{(i)} = \Big(\ v^{(i-1)}, (-1)^{u_i} v^{(i-1)} \Big)$.
3. $v = (-1)^{u_0} v^{(m)}$.

Nous allons maintenant démontrer que le résultat d'encodage v, vérifie :

$$v = \Big(\ (-1)^{f(\alpha_1)}, \ldots, (-1)^{f(\alpha_n)} \Big)$$

où $\Big(\ f(\alpha_1), \ldots, f(\alpha_n) \Big) \ \in RM(1, m)$, et $(\alpha_1, \ldots, \alpha_n) = \mathbb{F}_2^m$. Ensuite nous allons donner une écriture équivalente de cet algorithme à l'aide de la matrice génératrice de $RM(1, m)$.

Rappelons nous que la fonction d'encodage du code Reed-Muller d'ordre 1, est la suivante :

$$\begin{array}{rcl} C : \mathbb{F}_2^{m+1} & \rightarrow & \mathbb{F}_2^n \\ u = (u_0, u_1, \ldots, u_m) & \mapsto & \Big(f(\alpha_1), \ldots, f(\alpha_n) \Big) \end{array}$$

telle que :

$$\begin{array}{rcl} f : \mathbb{F}_2^m & \rightarrow & \mathbb{F}_2 \\ (x_1, \ldots, x_m) & \mapsto & u_0 + \sum_{i=1}^{m} u_i x_i \end{array}$$

Nous définissons les fonctions $f_i, i = 1, \ldots, m$ suivantes :

$$\begin{array}{rcl} f_i : \mathbb{F}_2^i & \to & \mathbb{F}_2 \\ x = (x_1, \ldots, x_i) & \mapsto & u_1 x_1 + \cdots + u_i x_i \end{array}$$

Il est immédiat de vérifier que :

$$f_i(x_1, \ldots, x_i) = f_{i-1}(x_1, \ldots, x_{i-1}) + u_i.x_i \tag{3.7}$$

soit

$$U_{f_i} = \Big(\ (-1)^{f_i(\alpha_1)}, \ldots, (-1)^{f_i(\alpha_{2^i})} \Big)$$

Nous avons pour $i = 1$,$\mathbb{F}_2 = \{0, 1\}$ et $f_1 = u_1 x_1$, donc :

$$\begin{array}{rcl} U_{f_1} & = & \Big(\ (-1)^{f_1(0)}, (-1)^{f_1(1)} \Big) \ , \\ & = & \Big(\ 1, (-1)^{u_1} \Big) \ , \end{array}$$

Proposition 3.2.2 *Pour $i = 1 \cdots m$, Nous avons :*

1. $U_{f_i} = (U_{f_{i-1}}, (-1)^{u_i} U_{f_{i-1}})$.
2. $U_f = (-1)^{u_0} U_{f_m}$.

Preuve

En effet, si $\mathbb{F}_2^i = (\alpha_1, \ldots, \alpha_{2^i})$ alors

$$\mathbb{F}_2^{i+1} = \Big(\ (\alpha_1, 0), \ldots, (\alpha_{2^i}, 0), (\alpha_1, 1), \ldots, (\alpha_{2^i}, 1) \Big)$$

Par définition

$$\begin{array}{rcl} U_{f_{i+1}} & = & \Big(\ (-1)^{f_{i+1}(\alpha_1,0)}, \ldots, (-1)^{f_{i+1}(\alpha_{2^i},0)}, (-1)^{f_{i+1}(\alpha_1,1)}, \ldots, (-1)^{f_{i+1}(\alpha_{2^i},1)} \Big) \\ & = & \Big(\ (-1)^{f_i(\alpha_1)}, \ldots, (-1)^{f_i(\alpha_{2^i})}, \\ & & \quad (-1)^{f_i(\alpha_1)+u_i}, \ldots, (-1)^{f_i(\alpha_{2^i})+u_i} \Big) \quad \text{(d'après l'équation 3.7)} \\ & = & (U_{f_{i-1}}, (-1)^{u_i} U_{f_{i-1}}) \end{array}$$

De même, comme $f = f_m + u_0$, alors $U_f = (-1)^{u_0} U_{f_m}$. Nous avons $U_{f_1} = v^{(1)}$ et la même relation de récurrence pour les U_{f_i} et $v^{(i)}$.

Alors :

$$\begin{array}{rcl} v & = & (-1)^{u_0} v^{(m)} \\ & = & (-1)^{u_0} U_{f_m} \\ & = & U_f \\ & = & \Big(\ (-1)^{f(\alpha_1)}, \ldots, (-1)^{f(\alpha_n)} \Big) \end{array}$$

L'encodage naturel d'un code linéaire se fait à l'aide de sa matrice génératrice.

Algorithme 3 .
Entrée : $u = (u_0, u_1, \ldots, u_m)$.
Sortie : $v = (v_1, \ldots, v_n), v_i \in \{-1, 1\}$.

1. poser $w = u.G_m$. avec G_m la matrice d'encodage de $RM(1, m)$ (voir 2.3.1).
2. $v = 1 - 2w = \left((-1)^{w_1}, \ldots, (-1)^{w_{2^m}} \right)$

3.2.2.2 Algorithme de décodage de $RM(1, m)$

L'outil de base pour le décodage à maximum de vraisemblance des codes de Reed-Muller d'ordre 1 est la transformée d'Hadamard.

Nous allons introduire cet outil, ensuite démontrer qu'avec cet outil nous arrivons à décoder en maximum de vraisemblance, et l'utiliser ensuite pour décoder les codes de Reed-Muller d'ordre 1.

Soit $Y = (y_1, \ldots, y_n)$ un vecteur reçu. Nous définissons la fonction f :

$$\begin{array}{rcl} f_Y : \mathbb{F}_2^m & \rightarrow & \mathbb{F}_2 \\ \alpha & \mapsto & y_\alpha \end{array}$$

et la fonction $F_Y : \mathbb{F}_2^m \rightarrow \{-1, 1\}$,tel que $F_Y(\alpha) = (-1)^{f_Y(\alpha)}$

La transformée d'Hadamard de F_Y, notée $\widehat{F_Y}$:

$$\begin{array}{rcl} \widehat{F_Y}(\alpha) & = & \sum_{\beta \in \mathbb{F}_2^m} (-1)^{<\alpha,\beta>} (-1)^{f_Y(\beta)}, \text{ où } \ <\alpha, \beta> = \alpha_1\beta_1 + \cdots + \alpha_m v_m \\ & = & \sum_{\beta \in \mathbb{F}_2^m} (-1)^{<\alpha,\beta> + f_Y(\beta)} \\ & = & \sum_{\beta \in \mathbb{F}_2^m} (-1)^{(f_Y + \sum_{i=1}^m \alpha_i X_i)(\beta)} \end{array}$$

Par conséquent, $\widehat{F_Y}(\alpha)$ est le nombre de 0 moins le nombre de 1 du vecteur de $\mathbb{F}_2^n$ associé à la fonction suivante (l'association entre $\mathcal{F}_m$ et $\mathbb{F}_2^n$ est décrite dans 2.2.2) :

$$f_Y + \sum_{i=1}^{m} \alpha_i X_i$$

Or :

* le nombre de 1 est le poids [4] de ce vecteur :

$$wt(f_Y + \sum_{i=1}^{m} \alpha_i X_i)$$

[4] le poids et la distance de Hamming sont définis dans 2.2.1

* le nombre de 0 est le nombre de bits restants, soit :

$$2^m - wt(f_Y + \sum_{i=1}^{m} \alpha_i X_i)$$

D'où :

$$\begin{aligned} \widehat{F_Y}(\alpha) &= 2^m - 2.wt(f_Y + \textstyle\sum_{i=1}^{m} \alpha_i X_i) \\ &= 2^m - 2.d(f_Y, \textstyle\sum_{i=1}^{m} \alpha_i X_i) \end{aligned}$$

où $d(.,.)$ désigne la distance définie dans 2.2.1.

Alors :

$$d\left(f_Y, \sum_{i=1}^{m} \alpha_i X_i\right) = \frac{1}{2}\Big(\ 2^m - \widehat{F_Y}(\alpha)\Big) \tag{3.8}$$

Par ailleurs, nous avons :

$$\begin{aligned} -\widehat{F_Y}(\alpha) &= \textstyle\sum_{\beta \in \mathbb{F}_2^m} - \Big((-1)^{<\alpha,\beta>}(-1)^{f_Y(\beta)}\Big) \\ &= \textstyle\sum_{\beta \in \mathbb{F}_2^m} (-1)^{1+<\alpha,\beta>+f_Y(\beta)} \\ &= \textstyle\sum_{\beta \in \mathbb{F}_2^m} (-1)^{\left(f_Y+\mathbf{1}+\sum_{i=1}^{m} \alpha_i X_i\right)(\beta)} \end{aligned}$$

Donc $-\widehat{F_Y}(\alpha)$ est le nombre de 0 moins le nombre de 1 du vecteur associé à la fonction suivante :

$$f_Y + \mathbf{1} + \sum_{i=1}^{m} \alpha_i X_i$$

Comme précédemment, nous en déduisons que :

$$d\left(f_Y, \mathbf{1} + \sum_{i=1}^{m} \alpha_i X_i\right) = \frac{1}{2}\left(2^m + \widehat{F_Y}(\alpha)\right) \tag{3.9}$$

Les deux équations (3.8) et (3.9), nous permettent d'écrire :

$$d\left(f_Y, \alpha_0\mathbf{1} + \sum_{i=1}^{m} \alpha_i X_i\right) \geq \frac{1}{2}\Big(\ 2^m - |\widehat{F_Y}(\alpha)|\Big)\ , \forall \alpha_0 \in \mathbb{F}_2 \tag{3.10}$$

Nous allons maintenant utiliser les résultats précédents afin de décrire la méthode de décodage à maximum de vraisemblance en accord avec le principe de la distance minimale.

Soit Y le mot reçu après transmission sur un canal bruité :

- Calculer $\widehat{F_Y} = \Big(\widehat{F_Y}(\alpha_1), \ldots, \widehat{F_Y}(\alpha_n)\Big)$.
- Choisir β, tel que $|\widehat{F_Y}(\beta)| = max\left\{|\widehat{F_Y}(\alpha_1)|, \ldots, |\widehat{F_Y}(\alpha_n)|\right\}$.

– Si $\widehat{F_Y}(\beta) \geq 0$, alors d'après l'équation (3.10), nous avons :

$$\begin{array}{rcl} d\left(f_Y, \sum_{i=1}^{m} \beta_i X_i\right) & = & \frac{1}{2}\left(2^m - \widehat{F_Y}(\beta)\right) \\ & \leq & \frac{1}{2}\left(2^m - |\widehat{F_Y}(\alpha)|\right) \ ,\forall \alpha \in \mathbb{F}_2^m \\ & \leq & d\left(f_Y, \alpha_0 \mathbf{1} + \sum_{i=1}^{m} \alpha_i X_i\right) \ ,\forall \alpha \in \mathbb{F}_2^m, \forall \alpha_0 \in \mathbb{F}_2 \\ & \leq & d(f_Y, g) \ ,\forall \ g \ \in \ \mathcal{F}_m, \ deg(g) \leq 1 \end{array}$$

– Si $\widehat{F_Y}(\beta) < 0$, alors d'après l'équation [3.9], nous avons :

$$\begin{array}{rcl} d(f_Y, \mathbf{1} + \sum_{i=1}^{m} \beta_i X_i) & = & \frac{1}{2}\Big(\ 2^m + \widehat{F_Y}(\beta)\Big) \\ & \leq & \frac{1}{2}\Big(\ 2^m - |\widehat{F_Y}(\alpha)|\Big) \quad ,\forall \alpha \in \mathbb{F}_2^m \\ & \leq & d(f_Y, \alpha_0 \mathbf{1} + \sum_{i=1}^{m} \alpha_i X_i) \ ,\forall \alpha \in \mathbb{F}_2^m, \forall \alpha_0 \in \mathbb{F}_2 \\ & \leq & d(f_Y, g) \ ,\forall g \in \mathcal{F}_m, deg(g) \leq 1 \end{array}$$

– En résultat, le calcul de β détermine la fonction de degré 1 le plus proche à f_Y, et cela répond au principe de décodage à distance minimale.

3.2.2.3 La matrice de transformée d'Hadamard

Maintenant que nous avons vu l'importance de la transformée d'Hadamard, nous allons calculer la matrice qui va nous permettre de calculer la transformée d'Hadamard d'une fonction représentée par le vecteur de ses valeurs.

Les matrices de transformées d'Hadamard sont définies par :

$$H_{2^m} = \Big(\ (-1)^{<i,j>}\Big)_{\ i,j<2^m}$$

L'utilité principale des matrices H_{2^m} est que si f est une fonction $f : \mathbb{F}_2^m \to \mathbb{C}$, ou $\mathbb{C}$ désigne le corps des complexes, Alors :

$$\widehat{f} = H_{2^m}.f$$

et nous avons

$$H_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, H_{2^m} = \underbrace{H_2 \otimes H_2 \otimes \cdots \otimes H_2}_{m \text{ fois}}$$

Nous rappelons que le produit de Kronecker de A par B est la matrice :

$$A \otimes B = \Big((a_{ij}B)\Big) = \begin{pmatrix} a_{11}B & a_{12}B & \dots & a_{1k}B \\ a_{21}B & a_{22}B & \dots & a_{2k}B \\ \vdots & \vdots & & \vdots \\ a_{n1}B & a_{n2}B & \dots & a_{nk}B \end{pmatrix}$$

Plusieurs études sont faites pour optimiser le calcul de la transformée d'Hadamard, nous allons utiliser la décomposition de la matrice d'Hadamard suivante :

Proposition 3.2.3 *La transformée d'Hadamard rapide (Walsh-Hadamard, Walsh [2]), Nous avons :*

$$H_{2^m} = \underbrace{M_m \dots M_m}_{m \text{ fois}}$$

où

$$M_m = \begin{bmatrix} I_{2^{m-1}} & \otimes & (1 & 1) \\ I_{2^{m-1}} & \otimes & (1 & -1) \end{bmatrix}$$

Nous renvoyons le lecteur à [2], pour la démonstration de cette proposition.

Et l'algorithme de décodage à distance minimale FHT :

Algorithme 4 FHT.
- Entrée : $Y = (Y_1, \dots, Y_n), Y_i \in \{-1, 1\}$.
- Sortie : $\beta = (\beta_0, \dots, \beta_m) \in \mathbb{F}_2^{m+1}$

1. Calculer $\widehat{F}$
   pour $i = 0$ à $m-1$ faire $Y = M_m Y^t$.

$$M_m = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 1 & 1 & 0 & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & 1 & 1 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & 0 & 0 & 1 & 1 \\ 1 & -1 & 0 & 0 & 0 & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 1 & -1 & 0 & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & 1 & -1 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & 0 & 0 & 1 & -1 \end{bmatrix}$$

2. Choisir β tel que $|Y_\beta|$ soit maximum, et soit $(\beta_0, \dots, \beta_{m-1})$ l'écriture binaire de β :
   - si $Y_\beta \geq 0$, le mot le plus proche est $(0, \beta_0, \dots, \beta_{m-1})$
   - si $Y_\beta < 0$, le mot le plus proche est $(1, \beta_0, \dots, \beta_{m-1})$

Notons que la première étape de l'algorithme revient à transformer le vecteur $(Y_1, Y_2, \dots, Y_{2^m})$ en $(Y_1 + Y_2, Y_3 + Y_4, \dots, Y_{2^m-1} + Y_{2^m}, Y_1 - Y_2, Y_3 - Y_4, \dots, Y_{2^m-1} - Y_{2^m})$

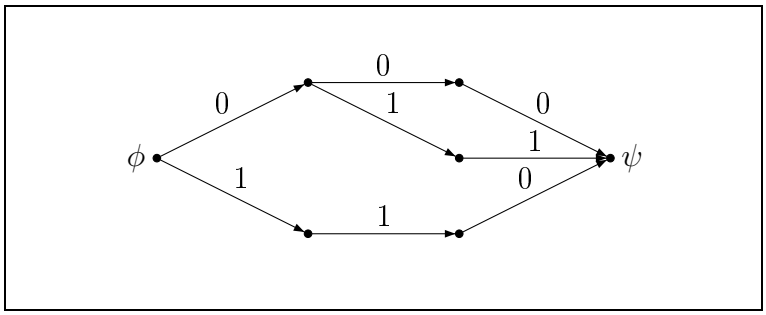


FIG. 3.3 – Treillis

et qui coûte n addition,

La complexité de cet algorithme est donc ramenée à celle du calcul de la transformée d'Hadamard de F, c'est-à-dire $O(m \times n)$, ce qui est largement mieux qu'avec une méthode brutale, dont la complexité serait en $O(n \times n)$.

Cet algorithme va être de grande utilité pour nous, et nous allons l'utiliser pour accélérer l'algorithme de *"V.M.Sidel'nikov et A.S.Pershakov"*.

3.2.2.4 Capacité de correction

L'algorithme de décodage 4 à distance minimale du code de Reed-Muller de premier ordre nous retourne un des mots les plus proches du mot reçu (voir 3.2.2.2), donc la borne de décodage asymptotique de l'algorithme-4 est $ML_1 = \frac{n}{2}\left(1 - \sqrt{4\ln(2)\frac{m}{n}}\right)$ (voir 3.2.2).

Nous nous intéressons au calcul de la borne de décodage pour laquelle, avec une grande probabilité, le résultat du décodage est le mot transmis. Pour chaque code $\mathcal{C}$, il est important de connaître cette borne qui est par définition celle de l'algorithme de décodage à maximum de vraisemblance. Nous notons cette borne comme $MLD(\mathcal{C})$. Arriver à prouver qu'un algorithme réussit à décoder jusqu'à cette borne signifie que c'est un algorithme de décodage à maximum de vraisemblance. Il est possible de retrouver $MLD(\mathcal{C})$ en étudiant la répartition des poids des mots de ce code.

3.2.3 Algorithme de décodage utilisant les Treillis

L'introduction des treillis (lattice : en anglais), par Forney [22] a été motivée suite à la mise en œuvre de l'algorithme de Viterbi par Andrew J. Viterbi [57] en 1967. L'application essentielle des treillis est le décodage à maximum de vraisemblance des codes en blocs et les turbo-codes, en utilisant l'algorithme de Viterbi. Le lecteur intéressé trouvera une description complète dans [42], chapitre 24 page 1989.

Nous allons tout d'abord définir le treillis et donner un exemple d'un treillis représentant un code, ensuite nous décrivons l'algorithme de Viterbi pour un treillis, et nous décrivons le décodage à l'aide de cet algorithme. Nous donnons ensuite la complexité d'un tel processus de décodage. Nous verrons que la complexité est liée à la taille du treillis,

et que pour les codes $RM(r,m)$ la taille est exponentielle en la longueur du code.

Définition 3.2.3 *Un Treillis est un graphe orienté valué défini par le triplet* $T = (V, E, A)$*, où* V *est un ensemble de sommets (V pour vertex),* A *est un alphabet et* E *un ensemble de triplets* $(v, \acute{v}, a), a \in A$ *(qui constitue un ensemble d'arêtes, E pour edge). L'ensemble des sommets* V *se divise en* $V = V_0 \cup V_1 \cup \cdots \cup V_n$ *toutes les arêtes de* E *partant des sommets de* V_i*,* $i = 0, \ldots, n-1$ *se terminent en* V_{i+1}*.*

Le flots d'un treillis est l'ensemble de tous les chemins reliant les sommets de V_0 aux sommets de V_n, nous présentons un code de longueur n, par un treillis $T = (V, E, A)$ contenant une seule racine ($V_0 = \phi$) et une seule feuille ($V_n = \psi$), tels que le flots du treillis T est l'ensemble de tous les mots du code $\mathcal{C}$.

Exemple 3.2.2 *Le code* $\mathcal{C} = \{000, 011, 110\}$ *est représenté par le treillis de la figure 3.3*

Nous allons décrire l'algorithme de décodage de Viterbi pour un code représenté par un treillis, c'est un algorithme de décodage à maximum de vraisemblance dont la complexité dépend seulement de la taille du treillis (nombre de sommets et d'arêtes). La question de la taille minimale de treillis représentant un code se pose naturellement, nous verrons que pour les codes de Reed-Muller un treillis minimal a toujours une taille exponentielle en la longueur de code.

3.2.3.1 Algorithme de Viterbi

L'application de la programmation dynamique sur un treillis permet de calculer un chemin spécifique. Étant donnée la bijection entre mots de code et chemin dans le treillis, le décodage à maximum de vraisemblance devient un cas spécial et simple à effectuer.

Soit $T = (V, E, A)$ un treillis, la partition de l'ensemble des sommets $V = V_0 \cup V_1 \cup \cdots \cup V_n$ est l'élément essentiel qui permet l'utilisation de l'algorithme de Viterbi sur les treillis. L'approche classique du calcul en programmation dynamique du bas-en-haut (bottom-to-top) des flots sur les $V_i, i = 0 \ldots n$ nous donne le chemin qui a le coût le plus bas (min-sum). Rappelez vous qu'un algorithme MLD[5], pour un mot reçu y, cherche le mot du code c minimisant le terme $d_{ML}(y, c) = -\sum_{i=1}^{n} \log(\Pr[y_i|c_i])$. Donc si nous modifions les étiquettes des arêtes de façon convenable et nous appliquons l'algorithme min-sum sur le treillis nous aurons le mot du code le plus probable. Nous allons décrire maintenant plus en détail ces étapes.

Nous avons besoin de fixer quelques notations : si $e = (v, \acute{v}, a)$ est une arête, nous appelons v son origine que nous notons aussi $*e = v$, et nous appelons destination de l'arête $\acute{v} = e*$. Nous définissons la fonction de coût $\alpha(e) = a$ qui renvoie l'étiquette d'une arête.

[5] Algorithme-1 en page 49

Algorithme 5 . Viterbi
Entrée : $y = (y_1, \ldots, y_n) \in \Sigma^n$
Sortie : $\tilde{c} = (c_1, \ldots, c_n) \in \mathcal{C}$.

1. $\mu(\phi) = 0$ //Initialisation
2. Pour $i = 1$ à n faire
   Pour $v \in V_i$ faire
   $\mu_y(v) = min_{e:e*=v}(\mu_y(*e) - \log(\Pr[y_i|\alpha(e)]))$
   $flux(v) = arg\ min_{e:e*=v}(\mu_y(*e) + \alpha(e))$
3. $v = \psi$
4. Pour $i = n$ à 1 faire
   $c_i = \alpha(flux(v))$
   $v = *flux(v)$
5. $\tilde{c} = (c_1, \ldots, c_n)$

Chaque mot du code $c = (c_1, \ldots, c_n)$ est représenté comme un parcourt $P_c = e_1, e_2, \ldots, e_n$ du treillis à partir de la racine ϕ jusqu'au nœud ψ, avec $\alpha(e_i) = c_i, i = 1 \ldots n$. L'algorithme Viterbi calcul, pour chaque nœud dans la graphe, le chemin à partir de la racine qui a le coût le plus bas.
En utilisant la fonction du coût d'un chemin $e_1, e_2, \ldots, e_l$ est $\sum_{i=1}^{l} - \log(\Pr[y_i|\alpha(e_i)])$
Nous définissons la fonction de coût minimal :

$$\begin{aligned} \mu_y(v) &= \min_{\{(e_1,e_2,\ldots,e_l) \mid *e_0=\phi, e_l*=v, *e_j=e_{j-1}* \ \forall\ 0<j<l\}}(\sum_{i=l}^{j} - \log(\Pr[y_i|\alpha(e_i)]) \\ &= \min_{e:e*=v}(\mu_y(*e) - \log(\Pr[y_i|\alpha(e)])) \end{aligned}$$

L'algorithme MLD trouve le mot du code qui minimise le terme $\sum_{i=1}^{n} - \log(\Pr[y_i|c_i]) = \mu_y(P_c)$, qui n'est autre que $\mu_y(\phi)$.

$$\begin{aligned} d_{ML}(y, c) &= -\log \Pr[y \mid c] \\ &= -\sum_{i=1}^{n} \log(\Pr[y_i|c_i]) \\ &= -\sum_{i=1}^{n} \log(\Pr[y_i|\alpha(e_i)]) \ , \quad P_c = (e_1, \ldots, e_n) \end{aligned}$$

Nous écrivons l'algorithme-5 qui effectue alors le décodage à maximum de vraisemblance pour une transmission sur un canal sans mémoire.

La complexité de l'algorithme précédent sur un treillis $T = (V, E, A)$ est de $|E|$ additions et $|E| - |V| + 1$ opérations comparaisons (min). Donc la complexité totale est de $2\,|E| - |V| + 1$ opérations.

La construction récursive Plotkin [39] des codes $RM(r, m)$ est à l'origine des algorithmes de décodage récursif. La construction géométrique présenté par Forney [23], lui a permis de construire des treillis pour ces codes. Il a prouvé dans cet article que le nombre des arêtes $|E|$ du treillis représentant $RM(r, m)$ est exponentiel. La taille

d'un treillis minimal représentant le code $RM(r,m)$ n'est pas connu encore. Plusieurs résultats intéressants concernant l'optimalité de La construction géométrique de Forney pour $RM(r,m)$ se trouve dans [42] chapitre 24.

3.3 Vote majoritaire MVD

Nous allons définir le **MVD** (en anglais : Majority vote decoding) pour le code à répétitions. Nous présenterons l'algorithme pour un code RM d'ordre 1. Ensuite nous le décrivons pour un code $RM(r,m)$ à l'aide de l'opérateur de la dérivée discrète.

Le code à répétitions consiste à envoyer plusieurs copies de chaque bit transmis. La première chaîne de caractères est appelée le 0 logique et la deuxième le 1 logique puisqu'elles jouent le rôle de 0 et 1 respectivement. Le décodage se fait par vote majoritaire, et c'est décider suivant le nombre de 0 où le nombre de 1
Ainsi, un code à répétitions sur cinq bits

$$0 \longrightarrow 00000$$
$$1 \longrightarrow 11111$$

Si le message reçu n'est ni le 0 logique ni le 1 logique, mais la chaîne de caractères 01010, alors il est très probable que ce soit un 0 logique qui ait été transmis à la source.

3.3.1 Exemple de MVD

Prenons le code $RM(1,4) = [n = 16, k = 5, d = 8]$, c'est un code qui détecte 7 erreurs et qui corrige 3 erreur. Soit $a = a_0a_1a_2a_3$ l'information à coder. Considérons la matrice génératrice G donnée par les 5 premières lignes de l'exemple 2.3.2. Le mot du code à transmettre vaut :

$$c = a.G = a_0 + a_1.x_1 + a_2.x_2 + a_3.x_3 + a_4.x_4 = (c_0c_1c_2c_3c_4c_5c_6c_7c_8c_9c_{10}c_{11}c_{12}c_{13}c_{14}c_{15})$$

Où on a :

$$\begin{aligned}
c_0 &= a_0 \\
c_1 &= a_0 + a_1 \\
c_2 &= a_0 + a_2 \\
c_3 &= a_0 + a_1 + a_2 \\
c_4 &= a_0 + a_3 \\
c_5 &= a_0 + a_1 + a_3 \\
c_6 &= a_0 + a_2 + a_3 \\
c_7 &= a_0 + a_1 + a_2 + a_3 \\
c_8 &= a_0 + a_4 \\
c_9 &= a_0 + a_1 + a_4 \\
c_{10} &= a_0 + a_2 + a_4 \\
c_{11} &= a_0 + a_1 + a_2 + a_4 \\
c_{12} &= a_0 + a_3 + a_4 \\
c_{13} &= a_0 + a_1 + a_3 + a_4 \\
c_{14} &= a_0 + a_2 + a_3 + a_4 \\
c_{15} &= a_0 + a_1 + a_2 + a_3 + a_4
\end{aligned}$$

A partir de ces équations, nous pouvons déduire :

$$\begin{array}{llll}
a_1 = c_0 + c_1 & a_2 = c_0 + c_2 & a_3 = c_0 + c_4 & a_4 = c_0 + c_8 \\
= c_2 + c_3 & = c_1 + c_3 & = c_1 + c_5 & = c_1 + c_9 \\
= c_4 + c_5 & = c_4 + c_6 & = c_2 + c_6 & = c_2 + c_{10} \\
= c_6 + c_7 & = c_5 + c_7 & = c_3 + c_7 & = c_3 + c_{11} \\
= c_8 + c_9 & = c_8 + c_{10} & = c_8 + c_{12} & = c_4 + c_{12} \\
= c_{10} + c_{11} & = c_9 + c_{11} & = c_9 + c_{13} & = c_5 + c_{13} \\
= c_{12} + c_{13} & = c_{12} + c_{14} & = c_{10} + c_{14} & = c_6 + c_{14} \\
= c_{14} + c_{15} & = c_{13} + c_{15} & = c_{11} + c_{15} & = c_7 + c_{15}
\end{array}$$

Le but est de reconstruire l'information a du mot reçu $y = c + e$, tel que e est le vecteur d'erreur. Soit $y = (y_0y_1y_2y_3y_4y_5y_6y_7y_8y_9y_{10}y_{11}y_{12}y_{13}y_{14}y_{15})$, nous utilisons les y_i à la place des c_i dans les équations précédentes pour trouver les bits d'informations a_i. Nous donnons la valeur 0 à a_i s'il y au moins cinq équations qui valent 0, et vice versa. Nous calculons maintenant le vecteur

$$\begin{aligned}
\acute{y} &= y - (a_4.x_4 + a_3.x_3 + a_2.x_2 + a_1.x_1) \\
&= a_0.\mathbf{1} + \text{ error}
\end{aligned}$$

Nous donnons la valeur 0 à a_0 si $\acute{y}$ a plus de zéros que de 1, et la valeur 1 sinon.

3.3.2 MVD de $RM(r, m)$

Un mot de code $RM(r, m)$ est une fonction booléenne de degré au plus r, et nous avons vu que l'opérateur de la dérivée discrète[6] a la propriété de diminuer l'ordre, et que

[6] définition 3.1.1 en page 46

$D_\alpha f(x)$ est d'ordre $r-1$. Donc, si on dérive r fois nous obtiendrons une fonction d'ordre 0 (un mot du code à répétitions).

Proposition 3.3.1 *Soit $f_r \in RM(r,m)$ écrite sous sa forme algébrique normale, notons $f_r(x) = f_{r-1}(x) + \sum_{1\leq i_1<i_2<\cdots<i_r\leq m} a_{i_1\ldots i_r}.x_{i_1}.\ldots.x_{i_r}$ avec $deg(f_{r-1}) \leq r-1$. Donc :*

$$D_{e_{i_1}} D_{e_{i_2}} \ldots D_{e_{i_r}} f_r = a_{i_1\ldots i_r}.\mathbf{1}$$

▷ PREUVE :
- Soit f une fonction constante, donc $\forall e_i, D_{e_i} f = 0$.
- Pour la fonction f_{r-1} de degré $r-1$, sa $(r-1)$ dérivée discrète est de degré 0, donc :

$$D_{e_{i_1}} D_{e_{i_2}} \ldots D_{e_{i_r}} f_{r-1} = 0$$

- Pour les monômes de degrés r, nous avons l'égalité suivante :

$$D_{e_{i_1}} \ldots D_{e_{i_r}} (x_{j_1}.\ldots.x_{j_r}) = \begin{cases} 1 & \text{si } \{i_1,\ldots,i_r\} = \{j_1,\ldots,j_r\} \\ 0 & \text{sinon} \end{cases}$$

Supposons qu'il existe $i_l \notin \{j_1,\ldots,j_r\}$, alors :

$$D_{e_l} f\ (x) = f(x) + f(x+e_l)$$
$$D_{e_l}(x_{j_1}.\ldots.x_{j_r}) = (x_{j_1}.\ldots.x_{j_r}) + (x_{j_1}.\ldots.x_{j_r}) = 0$$

C.Q.F.D ◁

Soit Y le vecteur reçu suite à la transmission du mot f_r de $RM(r,m)$, le décodage par vote majoritaire consiste à chercher d'abord les coefficients $a = a_{i_1\ldots i_r}$ d'ordre r. Le vecteur $D_{e_{i_1}} D_{e_{i_2}} \ldots D_{e_{i_r}} Y \in \mathbb{F}_2^{2^{m-r}}$ est considéré comme le vecteur erroné par le canal bruité suite à la transmission du mot $(a,\ldots,a)$. Décoder par vote majoritaire c'est décoder le code à répétitions de longueur 2^{m-r}. Après avoir trouvé les $\binom{m}{r}$ coefficients d'ordre r, Nous procédons avec les coefficients d'ordre $r-1$

Algorithme 6 . **MVD**
Entrée : $y = (y_1,\ldots,y_n) \in \mathbb{F}_2^n$
Sortie : $a \in \mathbb{F}_2^k$.

1. $a = []$, initialiser le résultat.
2. Pour $h = r$ jusqu'a 0 faire :
   Pour $1 \leq i_1 < i_2 < \cdots < i_h \leq m$ faire
   $z = D_{e_{i_1}} D_{e_{i_2}} \ldots D_{e_{i_h}} Y$
   $a_{i_1\ldots i_h} = 0$ si z a plus de zéros que de un, et 1 sinon.
   $a = [a_{i_1\ldots i_h}\ a]$
   $y = y + \sum_{1\leq i_1<i_2<\cdots<i_h\leq m} a_{i_1\ldots i_h}.x_{i_1}.\ldots.x_{i_h}$
3. $a = [a_0 a]$, avec $a_0 = 0$ si y a plus de zéros que de 1, et $a_0 = 1$ sinon.

3.3.3 capacité de correction

Le décodage par vote majoritaire du code $RM(r,m)$, consiste à retrouver les bits d'informations de plus grand ordre r. En dérivant r fois nous nous retrouvons dans le code à répétition (propriété-3.3.1). Nous allons calculer, tout d'abord, la capacité de correction du décodage MVD pour le code à répétition. Ensuite, nous détaillons le calcul pour le code $RM(r,m)$.

Nous considérons ici une transmission sur un canal binaire symétrique (BSC), supposons que la probabilité de mal transmettre un bit d'information est $p = \frac{1}{2}(1-\epsilon)$. Nous notons w le poids du vecteur d'erreur pour une transmission de n bits d'informations. L'espérance et la variance de w s'écrivent[7] :

$$\begin{aligned} E(w) &= \tfrac{n}{2}(1-\epsilon) \\ Var(w) &= \tfrac{n}{4}(1-\epsilon^2) \end{aligned}$$

La probabilité de mal décoder les bits d'informations noté BER[8] mesure le nombre d'erreurs par données reçues. Elle ne dépend pas du mot transmis. Nous supposons donc que le mot $f \equiv 0$ ait été transmis, et que à la réception le vecteur e est reçu.

Pour ce canal, l'erreur sur un bit d'information est une réalisation d'une variable aléatoire suivant la loi de Bernoulli (voir Annexe-A pour plus d'explication), et le poids de l'erreur sur un mot composé de n bits, suit la loi binomiale. Pour une grande valeur de n nous pouvons appliquer l'approximation de la loi binomiale par une loi gaussienne.

3.3.3.1 code à répétitions de longueur n

Nous transmettons n fois un bit d'information sur un canal bruité, pour mesurer le BER nous avons besoin de calculer la probabilité que l'erreur du canal ne se produit pas plus de $\frac{n}{2}$ fois, de telle sorte que le bit d'information sera correctement décodé.

$$\begin{aligned} \Pr\left[w \geq \tfrac{n}{2}\right] &= \Pr\left[w - E(w) \geq \tfrac{n}{2} - E(w)\right] \\ &= \Pr\left[w - E(w) \geq \tfrac{n}{2} - \tfrac{n}{2}(1-\epsilon)\right] \\ &= \Pr\left[w - E(w) \geq \tfrac{n}{2}\epsilon\right] \end{aligned}$$

En utilisant l'inégalité de Chebyshev

$$\begin{aligned} &\leq \frac{Var(w)}{(\frac{n}{2}\epsilon)^2} \\ &\leq \frac{(1-\epsilon^2)}{(n\epsilon^2)} \end{aligned}$$

Nous déduisons que si $\epsilon > \frac{1}{\sqrt{n}}$, Nous retrouvons tous les bits d'information avec une probabilité d'échec au plus $\frac{1}{n\epsilon^2}$.

[7] voir Note :12 Annexe-A

[8] Abréviation de l'expression anglaise "Bit Error Rate", voir 1.6

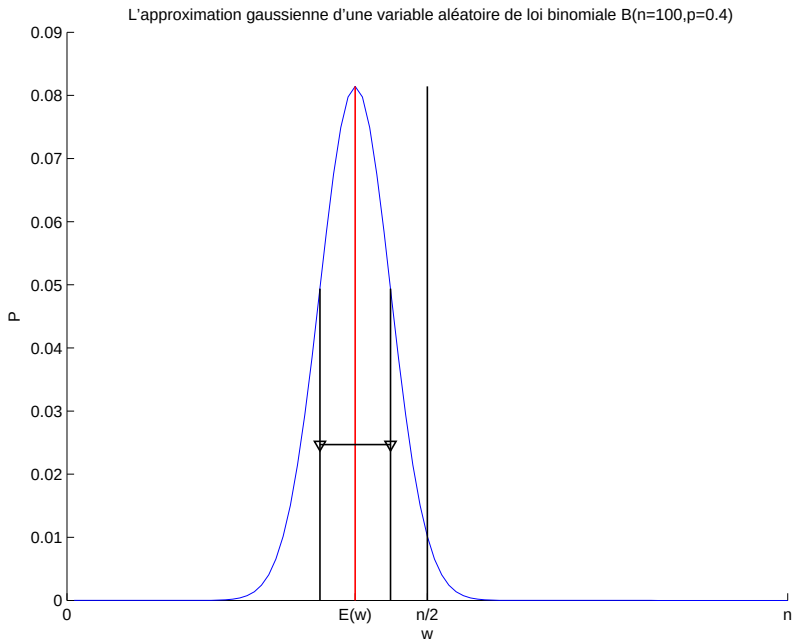


FIG. 3.4 – Erreur

3.3.3.2 code $RM(r,m)$

Nous pouvons établir la borne de de décodage asymptotique de l'algorithme-6 MVD en utilisant la proposition 3.3.1 qui ramène le décodage du $RM(r,m)$ à un code à répétition.

Théorème 3.3.1 *Pour le code de Reed-Muller $RM(r,m)$, de grande longueur $n=2^m$, l'algorithme MVD a La borne de décodage asymptotique suivante :*

$$\frac{n}{2}\left(1-\sqrt[2^{r+1}]{2^r\,\frac{m}{n}}\right)$$

▷ PREUVE : Soit c un mot du code $RM(r,m)$, soit $y=c+e$, le vecteur reçu, pour une transmission sur un canal binaire symétrique le vecteur d'erreur e est la réalisation d'une variable aléatoire avec probabilité $p=\frac{1}{2}(1-\epsilon)$. Nous avons vu que pour décoder ce mot avec un MVD algorithme (algorithme 6) nous commençons par retrouver les coefficients d'ordre r, ensuite ceux d'ordre $r-1$ et ainsi de suite.
Notons le vecteur $z=D_{e_{i_1}}D_{e_{i_2}}\dots D_{e_{i_\ell}}y\in\mathbb{F}_2^\ell$, $n_\ell=2^{m-\ell}$ la dérivée d'ordre ℓ de longueur n_ℓ (le nombre de coordonnées distinctes de z). L'erreur dans le vecteur z est vu comme la réalisation d'une variable de Bernoulli avec une probabilité $\widetilde{p}=\frac{1}{2}(1-\epsilon^{2^\ell})$, soit w le poids du vecteur d'erreur dans z. d'après Annexe-A Note-12 :

$$\begin{aligned}\mu_w &= n_\ell\,\widetilde{p}\\ \sigma_w^2 &= n_\ell\,\widetilde{p}(1-\widetilde{p})\end{aligned}$$

Nous avons :

$$\begin{aligned}\Pr\left[w \geq \frac{n_\ell}{2}\right] &= \Pr\left[w - E(w) \geq \frac{n_\ell}{2} - E(w)\right] \\ &= \Pr\left[w - E(w) \geq \frac{n_\ell}{2}\epsilon^{2^\ell}\right]\end{aligned}$$

En utilisant l'inégalité de Chernoff A.3.3

$$\begin{aligned}&\leq e^{-2n_\ell \frac{\epsilon^{2^\ell}{}^2}{2}} \\ &\leq e^{-\frac{n_\ell}{2}\epsilon^{2^{\ell+1}}}\end{aligned}$$

Notons $P_{err}^{\ell}(\epsilon)$ la probabilité de se tromper en décodant un bit d'information d'ordre ℓ , nous avons :

$$P_{err}^{\ell}(\epsilon) \leq e^{-\frac{2^{m-\ell}}{2}\epsilon^{2^{\ell+1}}} \tag{3.11}$$

Soit $\epsilon = \sqrt[2^{r+1}]{2^r \frac{m}{n}}$, d'après 3.11, $\forall \ell \leq r$:

$$P_{err}^{\ell}(\epsilon) \leq e^{-\frac{m}{2}}$$

Ainsi nous retrouvons un bit d'information avec une probabilité d'échec (BER) inférieur à $e^{-\frac{m}{2}}$ qui tends vers 0 quand la longueur du code tends vers ∞.
D'après 1.6, la probabilité WER de mal décoder un mot de k bits vaut $1 - \left(1 - P_{err}^{\ell}(\epsilon)\right)^k \approx kP_{err}^{\ell}(\epsilon)$ qui tend vers zéros pour $\epsilon = \sqrt[2^{r+1}]{2^r \frac{m}{n}}$, puisque la dimension du code $k = \sum_0^r \binom{m}{i}$ est polynomiale en m.

◁

Pour un code d'ordre deux $RM(2, m)$, cela veut dire qu'un algorithme de décodage par vote majoritaire décode avec une grande probabilité toute erreur survenue pour une transmission sur un canal $BSC(p), p \leq \frac{n}{2}(1 - \sqrt[8]{4\frac{m}{n}})$.

3.4 Décodage récursif

Le décodage récursif des codes de Reed-Muller a été introduit pour la première fois par G. Kabatianskiy [29] en 1990. Nous allons résumer les travaux de Dumer sur le décodage récursif en commençant par décrire la construction récursive des codes RM, qui permet de donner un algorithme d'encodage très rapide, et nous parlerons ensuite des deux algorithmes récursifs qui ont fait l'objet de plusieurs publications ([11],[7],[9],[8],[10],[14],[12]). Nous n'allons pas rentrer dans les détails de l'étude des bornes de décodages pour ces algorithmes, mais nous tenons à signaler l'originalité de cette étude et nous conseillons au lecteur intéressé par le décodage récursif de regarder dans les papiers de Dumer.

3.4.1 Construction récursive

Prenons une mot d'un code $f_r^m \in RM(r, m)$:

$$f_r^m(x_1, \ldots, x_m) = f_r^{m-1}(x_1, \ldots, x_{m-1}) + x_m . f_{r-1}^{m-1}(x_1, \ldots, x_{m-1})$$

Comme nous avons choisi l'ordre lexicographique pour représenter les éléments de $\mathbb{F}_2^m$, nous pouvons écrire :

$$\begin{array}{lll} ev(f_r^m) & = (\quad ev(f_r^{m-1}) \quad , & ev(f_r^{m-1}) + ev(f_{r-1}^{m-1}) \quad) \\ c & = (\qquad u \qquad , & u+v \qquad\qquad) \end{array} \tag{3.12}$$

Notons $a_r^m = \{a_j \mid j = 1, \ldots, k\}$ les k-bits d'informations pour une fonction $f_r^m \in RM(r,m) = [n = 2^m, k = \sum_0^r \binom{n}{i}, d = 2^{m-r}]$. Ces bits d'informations encodent le mot $c = (u, u+v)$, et nous divisons a_r^m en deux sous blocs a_r^{m-1} et a_{r-1}^{m-1} qui encodent les vecteurs u et v respectivement. Ensuite ces ensembles sont divisés de la même manière jusqu'à arriver à un code $RM(0,g)$ ou $RM(h,h)$, où $g, h \leq m$. Nous illustrons la récursivité de la construction par la figure-3.5, ceci est connu sous le nom de la construction **Plotkin**.

Note 6 *Notez que le nœud tout à droite a_0^g contient un seul bit d'information, et que le nœud tout à gauche a_h^h contient 2^h bits d'informations. Il est possible de coder ces 2^h bits d'informations avec la matrice d'identité, ce qui rend l'encodage et le décodage plus rapides.*

Nous venons de voir qu'il est possible d'encoder tous les mots du code à partir des bits d'informations associés aux nœuds a_0^g et a_h^h, et d'appliquer la construction $(u, u+v)$ ensuite. D'où l'algorithme d'encodage suivant :

Algorithme 7 $\psi(a, r, m)$
Entrée : $a \in \mathbb{F}_2^k$, $k = \sum_{i=0}^r \binom{m}{i}$.
Sortie : $c \in RM(r,m)$.
La longueur du code $n = 2^m$.

1. si $r = 0$, Code à répétition $c = a_0^m(1, \ldots, 1) \in \mathbb{F}_2^n$
2. si $r = m$, tout l'espace $\mathbb{F}_2^n$, utiliser la matrice identité pour coder : $c = a$.
3. si $0 < r < m$.
   diviser les bits d'informations a en $(a^u | a^v)$, et encoder récursivement :
   Calculer le vecteur $u = \psi(a^u, r, m-1)$
   Calculer le vecteur $v = \psi(a^v, r-1, m-1)$
4. Le résultat d'encodage $c = (u, u+v)$

Note 7 *Nous allons expliquer la deuxième étape de l'algorithme-7 : Soit $r > 0$, et le code $RM(r,r), n = 2^r$, Nous voulons coder un vecteur d'information $a = (a_0, \ldots, a_{n-1}) \in \mathbb{F}_2^n$*

avec la matrice d'encodage de ce code est $G = G(r,r)$ *(équation-2.7)*

$$c = a.G = ev(f_a) \mid f_a = \sum_{i=0}^{n-1} a_i \; x_1^{i_1} \dots x_r^{i_r}, \; i = (i_1, \dots, i_r)_{LSB} = i_1 + i_2 \times 2 + \dots + i_r \times 2^{r-1}$$

Pour simplifier les calculs, Dumer a choisi de coder l'information $a.G^{-1}$ *à la place de* a*, ainsi nous avons :*

$$\tilde{c} = (a.G^{-1}).G = a = ev(f_{a.G^{-1}})$$

Comme l'encodage pour le code $RM(r,r)$ *est une bijection (dimension du code égale à la dimension de l'espace), la matrice d'identité est aussi une matrice génératrice du code, pour laquelle le décodage ne nécessite pas de calcul.*

Lemme 3.4.1 *(lemme 4 [9])L'encodage du code* $RM(r,m), n = 2^m$ *peut être fait avec une complexité :*

$$|\psi(.,r,m)| \leq n \min(r, m-r) \tag{3.13}$$

▷ Preuve : Les deux premières étapes de l'algorithmes ne nécessitent pas d'encodage et leur coût vérifie l'inégalité (3.13). Nous allons démontrer que $|\psi(.,r,m)|$ vérifie (3.13) si les deux mots u et v ont une complexité d'encodage $|\psi(.,r,m-1)|$ et $|\psi(.,r-1,m-1)|$ qui vérifie (3.13) aussi. Dans cette proposition la combinaison $(u, u+v)$ nécessite la complexité :

$$|\psi(.,r,m)| \leq |\psi(.,r-1,m-1)| + |\psi(.,r,m-1)| + \frac{n}{2}$$

tel que le terme $\frac{n}{2}$ est nécessaire pour faire l'addition modulo deux pour trouver le terme de droite $u+v$. Maintenant nous utilisons (3.13) pour les termes $|\psi(.,r-1,m-1)|$ et $|\psi(.,r,m-1)|$, et nous avons les cas suivants :

- $r < m - r$
$$|\psi(.,r,m)| \leq \frac{n(r-1)}{2} + \frac{nr}{2} + \frac{n}{2} = nr$$
- $r > m - r$
$$|\psi(.,r,m)| \leq \frac{n(m-r)}{2} + \frac{n(m-r-1)}{2} + \frac{n}{2} = n(m-r)$$
- $r = m - r$
$$|\psi(.,r,m)| \leq \frac{n(r-1)}{2} + \frac{n(m-r-1)}{2} + \frac{n}{2} = nr - \frac{n}{2} \leq nr$$

◁

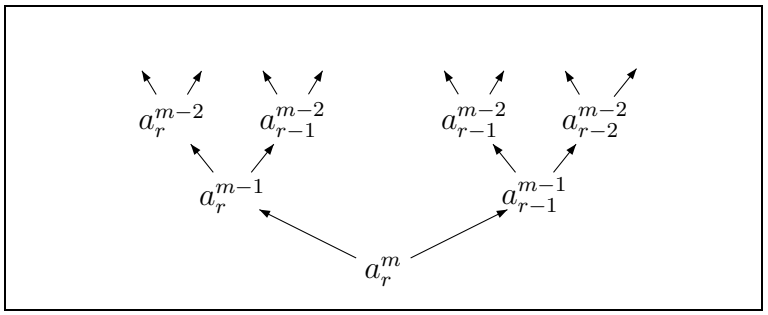


FIG. 3.5 – décodage récursif

3.4.2 Idée de l'algorithme de décodage

Soit $f_r^m \in RM(r,m)$, nous pouvons écrire cette fonction de la façon suivante :

$$f_r^m(x_1,\ldots,x_m) = f_r^{m-1}(x_1,\ldots,x_{m-1}) + x_m.f_{r-1}^{m-1}(x_1,\ldots,x_{m-1})$$

Calculons la dérivée de cette fonction selon la direction e_m :

$$\begin{array}{rcl} D_{e_m}(f_r^m(x)) & = & f(x) + f(x+e_m) \\ & = & f_{r-1}^{m-1}(x_1,\ldots,x_{m-1}) \end{array}$$

Ainsi la dérivée d'une fonction d'ordre r en m variables est une fonction d'ordre $r-1$ en $m-1$ variables. En choisissant l'ordre lexicographique pour représenter les éléments de $\mathbb{F}_2^m$, nous notons en écriture binaire sur m bits : $I_m = \{0,1,2,\ldots,2^m-1\}$, et nous avons :

$$\begin{array}{lcccl} ev_{I_m}(f_r^m) & = (& ev_{I_{m-1}}(f_r^{m-1}) & , & ev_{I_{m-1}}(f_r^{m-1}) + ev_{I_{m-1}}(f_{r-1}^{m-1}) \quad) \\ c & = (& u & , & u+v \quad) \end{array}$$

D'où :

$$D_{e_m}(c) = (v,v)$$

Note 8 *Pour un vecteur $y \in \mathbb{F}_2^n$, la dérivée selon la direction e_m est un vecteur de la forme $D_{e_m}y = (z,z)$, $z \in \mathbb{F}^{\frac{n}{2}}$, ainsi pour décrire le décodage récursif, nous allons considérer que la motié du vecteur dérivé selon cette direction et notez $D_{e_m}y = z$.*

Soit $c = (u, u+v)$ un mot du code $RM(r,m)$ transmis sur un canal de transmission. Le vecteur reçu $y \in \mathbb{F}_2^n$ se compose de deux parties de même taille $y = (y_g, y_d)$.

$$y = (\underbrace{y_1, y_2, \ldots, y_{\frac{n}{2}}}_{y_g}, \underbrace{y_{\frac{n}{2}+1}, \ldots, y_n}_{y_d}) = c + e$$

Où $e = (e_g, e_d)$ est le vecteur d'erreur dû à la transmission sur le canal bruité. Calculons la dérivée du vecteur reçu selon la direction e_m :

$$\begin{aligned} D_{e_m}(y) &= D_{e_m}(c) + D_{e_m}(e) \\ &= v + e_g + e_d \end{aligned}$$

De cette manière, en dérivant un mot erroné d'un code $RM(r, m)$ nous obtenons un mot erroné d'un code $RM(r-1, m-1)$ que nous décodons de manière récursive. Si on retrouve bien le mot $v \in RM(r-1, m-1)$ nous procédons au décodage de la deuxième composante du mot reçu :

$$\begin{aligned} y + (0, v) &= (y_g, y_d) + (0, v) \\ &= (u + e_g, u + v + e_d) + (0, v) \\ &= (u + e_g, u + e_d) \end{aligned}$$

Nous avons deux versions corrompues du mot $u \in RM(r, m-1)$. Dumer dans ses articles les plus récents a étudié la performance de cette méthode en choisissant de décoder la moyenne de ces deux instances qui est : $\frac{y_g + (y_d + v)}{2}$ et de décoder ce vecteur dans $RM(r, m-1)$. L'algorithme continue de décoder de façon récursive jusqu'à arriver sur un code simple, un code qu'on sait décoder de façon optimale (code à répétition, code d'ordre-1 $RM(1, m)$, l'espace binaire $\mathbb{F}_2^n$ $RM(m, m)$).

L'algorithme de Dumer fait du décodage souple et du décodage dur. Ainsi nous considérons le changement d'un symbole binaire $a \Leftrightarrow (-1)^a,\ a \in \mathbb{F}_2$. De cette manière, nous supposons qu'un mot du code $RM(r, m)$ appartient à l'espace $\{1, -1\}^n$ ce qui signifie que la somme dans le corps à deux éléments $\mathbb{F}_2$ sera transformée en le produit de leurs images :

$$a + b \Leftrightarrow (-1)^a (-1)^b = (-1)^{a+b} \mid a, b \in \mathbb{F}_2$$

Dès lors, un mot du code aura la forme $c = (u, uv)$, de manière plus générale nous supposons que le vecteur reçu $y \in \mathbb{R}^n$.

Les algorithmes récursifs procèdent comme suit. Pour un vecteur $y = (y_g, y_d)$:

1. Retrouver le mot du code $v \in RM(r-1, m-1)$.
   En absence de bruit, nous avons l'égalité $v = y_g y_d$ (dans les notations d'origine cela s'écrivait avec une somme de composants binaires). Dans un canal bruité, nous calculons d'abord l'estimation noté y^v :

   $$y^v = y_g y_d \tag{3.14}$$

   et nous appliquons récursivement notre algorithme de décodage. La sortie de l'algorithme est le mot $\hat{v} \in RM(r-1, m-1)$ et son vecteur d'information $\hat{a}^v$.
2. Ayant le vecteur $\hat{v}$ de l'étape 1 de l'algorithme, nous cherchons le bloc $u \in RM(r, m-1)$. Ainsi nous avons deux versions erronées du vecteur u, la partie gauche du mot vecteur reçu y_g et le vecteur $y_d \hat{v}$ de la partie droite. Ces deux estimations (vecteurs réels) sont combinés par leur moyenne

   $$y^u = \frac{y_g + y_d \hat{v}}{2} \tag{3.15}$$

   et nous appliquons récursivement notre algorithme de décodage. La sortie de l'algorithme est le mot $\hat{u} \in RM(r, m-1)$ et son vecteur d'information $\hat{a}^u$.

Notons que dans ces deux étapes nous ne faisons que calculer les vecteurs intermédiaires y^v et y^u en utilisant (3.14) quand le décodeur avance à droite (regarder la figure-3.5), et utilise (3.15) quand le décodeur avance à gauche.
Nous avons décrit précédemment le principe de l'algorithme récursif basé sur la construction Plotkin ([39], équation-3.12). Maintenant nous allons discuter des conditions d'arrêt de cet algorithme. Nous avons dit précédemment que nous arrêtions la récursion quand nous obtenons un mot du code de Reed-Muller que nous savons décoder de façon optimale (**MDD** définition-1.5.1).

Jusqu'à présent nous connaissons des algorithmes de décodage optimale pour les cas suivants :

1. $RM(0,m)$ qui est le code à répétition contient deux mots. Dans ce cas nous prenons l'algorithme de décodage suivant :

$$\begin{array}{l} MDD_{RM(0,g)}(y) = (\hat{c},\hat{a}) \\ \left| \begin{array}{lll} \hat{c} = (1,\ldots,1), \hat{a} = 0 & \text{si} & \sum_{i=1}^{2^g} y_i > 0 \\ \hat{c} = (-1,\ldots,-1), \hat{a} = 1 & \text{si} & \sum_{i=1}^{2^g} y_i < 0 \end{array} \right. \end{array}$$

2. $RM(m,m)$ c'est tout l'espace $\{1,-1\}^n$ (ou l'espace $\mathbb{F}_2^n$ dans les notations d'origine). Décoder à minimum distance c'est trouver un des mot les plus proches dans la métrique euclidienne, et c'est à dire :

$$\begin{array}{l} MDD_{RM(h,h)}(y) = (\hat{c},\hat{a}) \\ \left| \begin{array}{l} a = (a_1,\ldots,a_{2^h}) \mid a_i = \left\{ \begin{array}{lll} 1 & \text{si} & y_i < 0 \\ 0 & \text{si} & y_i > 0 \\ 0 \text{ où } 1 & \text{si} & y_i = 0 \end{array} \right. \\ \hat{c} = (1-2\ a_1,\ldots,1-2\ a_{2^h}) \in \{1,-1\}^{2^h} \\ \hat{a} = a.G(h,h)^{-1} \text{ (Note : } \hat{a} = a \text{ si l'algorithme-7 est utilisé pour l'encodage)} \end{array} \right. \end{array}$$

3. $RM(1,m)$ c'est le code d'ordre un. Nous le décodons à maximum de vraisemblance en utilisant l'algorithme-4 FHT.

Nous avons principalement deux algorithmes récursifs, le premier c'est $\Psi(y,r,m)$ (algo-8) qui s'arrête sur les codes $RM(0,g)$ et $RM(h,h)$.

Algorithme 8 $\Psi(y,r,m)$
Entrée : $y = (y_1,y_2,\ldots,y_n) = (y_g,y_d) \in \mathbb{R}^n$.
Sortie : $\hat{a},\hat{c}$.

1. si $r = 0$, $(\hat{c},\hat{a}) = MDD_{RM(0,m)}(y)$.
2. si $r = m$, alors $(\hat{c},\hat{a}) = MDD_{RM(r,r)}(y)$.
3. si $0 < r < m$.
   Calculer le vecteur $y^v = y_g y_d$, et poser $(\hat{a}^v,\hat{v}) = \Psi(y^v, r-1, m-1)$
   Calculer le vecteur $y^u = \frac{(y_g + y_d\hat{v})}{2}$, et poser $(\hat{a}^u,\hat{u}) = \Psi(y^u, r, m-1)$
4. Le résultat du décodage $\hat{a} = (\hat{a}^u|\hat{a}^v)$; $\hat{c} = (\hat{u}, \hat{v}\hat{u})$

le deuxième c'est $\Phi(y, r, m)$ (algo-9) qui s'arrête sur les codes $RM(1, g)$ et $RM(h, h)$.

Algorithme 9 $\Phi(y, r, m)$
Entrée : $y = (y_1, y_2, \ldots, y_n) = (y_g, y_d) \in \mathbb{R}^n$.
Sortie : $\hat{a}, \hat{c}$.

1. si $r = 1$, décoder y avec l'algorithme-4 FHT.
2. si $r = m$, alors $(\hat{c}, \hat{a}) = MDD_{RM(r,r)}(y)$.
3. si $1 < r < m$.
   Calculer le vecteur $y^v = y_g y_d$, et poser $(\hat{a}^v, \hat{v}) = \Phi(y^v, r-1, m-1)$
   Calculer le vecteur $y^u = \frac{(y_g + y_d \hat{v})}{2}$, et poser $(\hat{a}^u, \hat{u}) = \Phi(y^u, r, m-1)$
4. Le résultat du décodage $\hat{a} = (\hat{a}^u | \hat{a}^v)$; $\hat{c} = (\hat{u} | \hat{v}\hat{u})$

En décodage appelé dur (hard-decision decoding) les symboles transmis sur un canal binaire symétrique (BSC) sont inversés avec une probabilité p. Par contraste, en décodage souple (soft-decoding) le signal reçu n'est pas arrondi à ± 1, en conséquence l'algorithme a de meilleures performances en décodage souple qu'en dur.

Le premier algorithme de décodage récursif a été proposé par G. Kabatianskiy [29] en 1990, dans cet article l'auteur a proposé un algorithme de décodage basé sur la construction récursive des codes de Reed-Muller, l'algorithme génère de façon récursive le mot de code qui est le résultat du décodage, et le décodage se fait au niveau des codes $RM(0, m)$ et $RM(m-1, m)$. La complexité de l'algorithme est de $O(n \min(r, m-r))$ (équation-8 [29])

I. Dumer et R. Krichevskiy ont étudié la capacité de correction de l'algorithme de décodage par vote Majoritaire en décodage souple [13]. Dans cette étude, ils ont analysé la probabilité d'erreur dans les équations de votes (appelé aussi équations de parités). Cette étude a permis d'étudier la capacité de correction de l'algorithme récursif proposé par G. Kabatianskiy [29]. Inspiré de l'algorithme de [29], I. Dumer a proposé dans un congrès à Monticello en 1999 son premier algorithme de décodage récursif pour les codes $RM(r, m)$ basé sur la structure de Plotkin de ces codes. La version longue de cet article est sortie en 2000 [7]. L'algorithme publié dans [7] est celui à l'origine de Φ (algorithme-8) sauf que l'étape de la normalisation (division par 2) de la troisième étape, n'y était pas.

Lemme 3.4.2 *(Lemme 5,[16]) Pour le code $RM(r,m)$, $r < m$, $n = 2^m$, les algorithmes de décodages, $\Phi(.,r,m)$ et $\Psi(.,r,m)$, ont les complexités suivantes :*

$$|\Psi(.,r,m)| \leq 3n\min(r,m-r)+n \tag{3.16}$$

$$|\Phi(.,r,m)| \leq 2n\min(r,m-r)+n(m-r)+n \tag{3.17}$$

Les deux algorithmes ont une complexité majorée par $\frac{3}{2}n\log_2(n)$.

3.4.3 Capacité de correction

Nous allons ici citer les principaux résultats sur l'étude des performances des algorithmes de décodage récursif, nous conseillons au lecteur intéressé de lire l'étude puisque nous la trouvons très intéressante. L'étude des performances a évolué depuis l'apparition du premier algorithme récursif en 1999 [11]. La performance sur un canal binaire symétrique (BSC) est étudiée dans [9] [16] , et pour le canal $AWGN$ dans [13] [10].

3.4.3.1 Canal BSC

Théorème 3.4.1 *(théorème 14, [16]), Pour les codes de Reed-Muller $RM(r,m)$, de grande longueur $n = 2^m$, l'algorithme $\Phi(.,r,m)$ a une borne de décodage asymptotique (voir Définition-1.6.1) $\frac{n}{2}$ avec le terme résiduel $(\frac{m2^{r+1}}{n})^{1/2^{r+1}}$.*

- *Il corrige avec une grande probabilité toute erreur de poids inférieur à*

$$\frac{n}{2}\left(1-\sqrt[2^{r+1}]{\frac{m2^{r+1}}{n}}\right)$$

- *Avec une probabilité non négligeable il ne corrige pas les erreurs de poids supérieur ou égale à :*

$$\frac{n}{2}\left(1-\sqrt[2^{r+1}]{\frac{2^r}{n}}\right)$$

Théorème 3.4.2 *(théorème 16, [16]), Pour les codes de Reed-Muller $RM(r,m)$, de grande longueur $n = 2^m$, l'algorithme $\Psi(.,r,m)$ a une borne de décodage asymptotique (voir Définition-1.6.1) $\frac{n}{2}$ avec le terme résiduel $(\frac{m2^{r+1}}{n})^{1/2^r}$.*

- *Il corrige avec une grande probabilité toute erreur de poids inférieur à*

$$\frac{n}{2}\left(1 - \sqrt[2^r]{\frac{m2^{r+2}}{n}}\right)$$

- *Avec une probabilité non négligeable il ne corrige pas les erreurs de poids supérieur ou égale à :*

$$\frac{n}{2}\left(1 - \sqrt[2^r]{\frac{2^r}{n}}\right)$$

Soit c un mot de code $RM(r,m)$, soit $y = c+e$, le vecteur reçu, pour une transmission sur un canal binaire symétrique le vecteur d'erreur e est la réalisation d'une variable aléatoire avec probabilité $p = \frac{1}{2}(1-\epsilon)$. Nous avons vu que la première étape de ces deux algorithmes consiste à calculer la dérivée du mot reçu, La dérivée fait augmenter le bruit d'un facteur carré ($p = \frac{1}{2}(1-\epsilon^2)$). Regardez la branche tout à droite de la récursion (figure-3.5), qui correspond à la dérivée, elle est visitée r-fois dans l'algorithme $\Psi(.,r,m)$ et $(r-1)$-fois pour $\Phi(.,r,m)$, pour arriver à un nœud que nous savons décoder à distance minimale. C'est pour cette raison qu'on gagne un facteur carré dans le bruit. En ce qui concerne le canal gaussien $AWGN$, le lecteur intéressé trouvera dans [10] l'étude de l'algorithme Φ.

3.5 Décodage en liste

Le décodage en liste a été introduit par P. Elias dans son article [18]. Soit $\mathcal{C}$ un code et δ un rayon de décodage. Pour un mot reçu y, un décodeur en liste retourne la liste :

$$L_{T,\mathcal{C}}(y) = \{c \in \mathcal{C} : d(y,c) \leq T\}$$

de tous les mots du code $\mathcal{C}$ dans une boule de rayon T autour du mot y. Une question importante consiste à définir un rayon de décodage maximal pour lequel il existe des algorithmes de décodage en liste de faible complexité. Il existe une borne qui garantit que la taille de la liste retournée par un décodeur en liste $L_{T,\mathcal{C}}(y)$ reste bornée. C'est ***la borne de Johnson***. Soit $\mathcal{C}[n,d]$ un code de longueur n et de distance minimale $d = \delta n$. Soit $B_{n,T}(y) = \{x : d(x,y) \leq T\}$ la boule de Hamming de rayon $T = \omega n$ centrée sur un point y. La borne de Johnson limite le nombre de mots de code à

$$|L_{T,\mathcal{C}}(y)| = |\mathcal{C} \cap B_{n,T}(y)| \leq \frac{\delta}{\delta - 2\omega(1-\omega)} \tag{3.18}$$

à condition que le dénominateur de l'équation (3.18) soit positif.

La borne de Johnson a été définie dans [34] pour un code binaire (non nécessairement linéaire). Elle a été généralisée dans [25](théorème 4.2 page 554). Pour un code $RM(r,m) = [n = 2^m, k, d = 2^{m-r}]$ la borne de Johnson (J_r) est la fraction d'erreur limite au delà de laquelle l'équation 3.18 n'est plus utilisable, donc nous avons :

$$\begin{array}{lll} d = 2^{m-r} & \Rightarrow & \delta = 2^{-r} \\ \delta - 2J_r(1-J_r) = 0 & \Rightarrow & J_r = \frac{1}{2}(1-\sqrt{1-2^{-r+1}}) \end{array}$$

Pour un rayon de décodage $T_\epsilon(r) = n(J_r - \epsilon)$ l'équation 3.18 borne la taille de la liste des mots du code $RM(r,m)$ dans un rayon de décodage $T_\epsilon(r)$ par la constante suivante :

$$|L_{T,\mathcal{C}}(y)| = |\mathcal{C} \cap B_{n,T}(y)| \leq \frac{1}{2^{r+1}\epsilon(1-2J_r+\epsilon)} \tag{3.19}$$

Ceci est vrai quel que soit le mot reçu y. Notons que cette borne est largement supérieure à la capacité de correction (décodage non ambigu), ceci est vérifié par l'inégalité suivante. $\forall r > 0$, nous avons :

$$\frac{d}{2} + \frac{d}{2^{r+2}} \leq nJ_r \leq \frac{d}{2} + \frac{d}{2^{r+1}}$$

pour le code du premier ordre ($r = 1$), bien que la transformée de Hadamard (algorithme-4 FHT) soit très rapide, elle est impraticable pour les paramètres qui nous intéressent par exemple $m = 64$, comme dans la cryptanalyse linéaire du DES. La borne de Johnson $J_0 = \frac{1}{2}$, ce qui signifie qu'un algorithme de décodage en liste réussit à décoder aussi bien qu'un algorithme à maximum de vraisemblance[9]. Ceci n'est plus vrai pour les ordres supérieurs, donc un algorithme de décodage en liste jusqu'à la borne de Johnson (erreur de poids nJ_r) réussit à décoder bien au-delà de la capacité de correction des codes de Reed-Muller, mais beaucoup moins qu'un algorithme de décodage à maximum de vraisemblance(voir La figure 3.6).

Pour les codes d'ordre 1, Goldreich et Levin [24] ont publié en 1989 le premier algorithme de décodage en liste en caractéristique 2, qui a été étendu par Sudan, Rubinfeld et Goldreich [25] aux autres caractéristiques. Ces algorithmes fonctionnent en m étapes. A chaque étape, on teste tous les candidats, et on garde ceux qui ont une chance de provenir de véritables solutions du problème de décodage. Cédric Tavernier a étudié dans sa thèse [54] l'algorithme de Goldreich et Levin. Il a remarqué que la liste des candidats, qui ne sont pas nécessairement des bons candidats et qui vont être éliminés dans les étapes suivantes, au cours de l'algorithme peut dépasser la borne de Johnson, ceci augmente la complexité de l'algorithme et la mémoire nécessaire à son exécution. Cédric Tavernier a proposé un nouvel algorithme, en modifiant le critère d'acceptation des candidats à la solution et en utilisant l'algorithme de décodage à maximum de vraisemblance (FHT) comme première étape de l'algorithme (trouver rapidement les candidats qui sont des fonctions de quelques premières variables).

[9] **MLD** (Maximum Likelihood Decoding) : On appelle décodage à maximum de vraisemblance, l'obtention du mot de code qui a la plus grande probabilité d'être transmis ayant reçu le mot y.

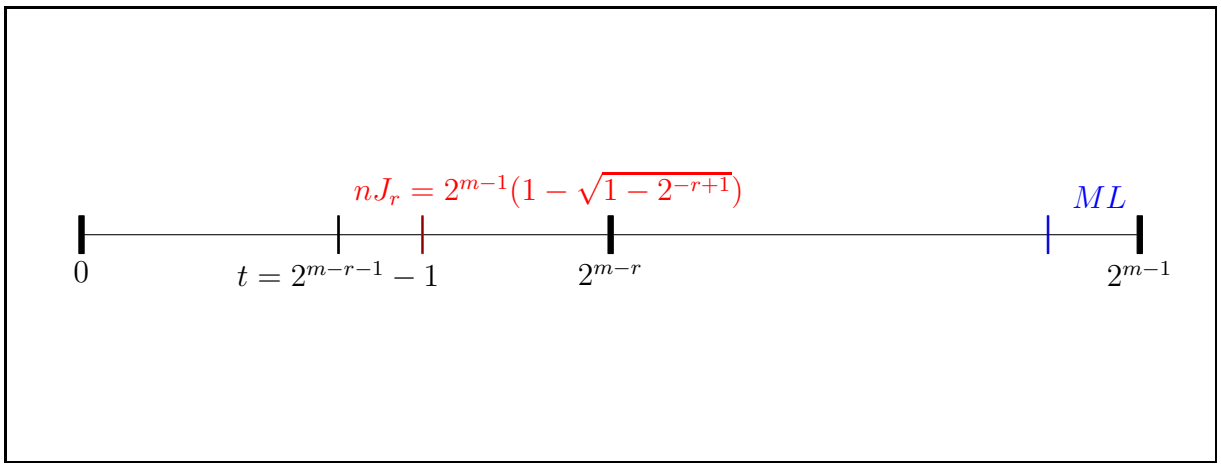


FIG. 3.6 – Les bornes de décodage des codes de Reed-Muller

Chapitre 4

Algorithme de Sidel'nikov et Pershakov

Contenu

Dans ce chapitre nous allons décrire l'algorithme de décodage des codes de Reed-Muller d'ordre deux, proposé par Sidel'nikov et Pershakov [50]. C'est un algorithme de décodage souple. Nous allons analyser sa capacité de correction pour une transmission sur un canal BSC. Les algorithmes de décodage récursif qui sont plus rapides, ont la même borne de décodage asymptotique, mais pour les codes de longueurs moyennes l'algorithme SP décode plus loin que les algorithmes récursifs. Nous allons présenter cet algorithme et faire l'analyse de sa capacité de correction asymptotique. Nous proposons aussi une simplification de l'algorithme qui a une complexité m^2n et qui a la même capacité de correction asymptotiquement. Nous expliquerons ensuite la version paramétrée de l'algorithme SP que les auteurs ont proposé dans leur article.

4.1 Idée de l'algorithme

Le groupe d'automorphismes de $RM(r,m), r>1$ (voir 2.4) est l'espace affine $GA(m)$ formé de transformations de la forme suivante :

$$\begin{array}{rcl} \sigma : \mathbb{F}_2^m & \rightarrow & \mathbb{F}_2^m \\ x & \mapsto & A.x^t + \alpha \end{array}$$

avec $A \in GL_m(\mathbb{F}_2)$ une matrice inversible $m \times m$ d'éléments de $\mathbb{F}_2$, et $\alpha \in \mathbb{F}_2^m$. Nous notons $g(x) \stackrel{\triangle}{=} f(A.x^t + \alpha)$. Alors le vecteur $ev(g)$ est un mot du code $RM(r,m)$ et ce vecteur est obtenu à partir de $ev(f)$ par une permutation σ de ses coordonnées. Le vecteur $ev(f+g) = ev(f) + ev(g)$ est aussi un mot du code $RM(r,m)$.

Pour certaines valeurs de A et α le vecteur $ev(f+g)$ est dans un sous code de $RM(r,m)$. L'article de *V.M.Sidel'nikov et A.S.Pershakov* traite le cas $A = I_m$ (la matrice identité). Dans ce cas particulier, nous avons[1] :

$$f(x) + g(x) = f(x) + f(x+\alpha) = D_\alpha(f)(x)$$

et l'ordre de la fonction $D_\alpha(f)$ est au plus $r-1$ (voir propriété-3.1.1). Dans ce cas :

$$D_\alpha(f) \in RM(r-1,m)$$

De plus, la fonction $D_\alpha(f)$ prend les mêmes valeurs pour les combinaisons x et $x+\alpha$, alors par un changement de variable adéquat elle dépend de $m-1$ variables.

Note 9 *Nous rappelons que l'algorithme de décodage récursif est basé sur la dérivée dans une direction particulière qui est e_m, pour une fonction*

$$f_r^m(x_1,\ldots,x_m) = f_r^{m-1}(x_1,\ldots,x_{m-1}) + x_m.f_{r-1}^{m-1}(x_1,\ldots,x_{m-1})$$

La dérivée s'écrit :

$$D_{e_m}(f_r^m(x)) = f_{r-1}^{m-1}(x_1,\ldots,x_{m-1}) \in RM(r-1,m-1)$$

Nous considérons l'erreur comme un vecteur qui va être rajouté au mot du code $f \in RM(r,m)$, et notons le vecteur reçu $y = f + e$, ainsi :

$$D_\alpha(y) \quad = \quad D_\alpha(f) + D_\alpha(e)$$

[1] La fonction dérivée voir définition-3.1.1, page-46

Notons bien que le nombre d'erreurs dans $D_\alpha(y)$ est plus grand que dans y, mais comme

$D_\alpha(y) \in RM(r-1,m)$ alors nous pouvons décoder $D_\alpha(y)$ plus simplement que y. Dans ce qui suit nous allons exploiter plus en détail cette idée pour le code $RM(2,m)$.

4.2 Reed-Muller d'ordre 2

Nous allons décrire l'algorithme de décodage pour $RM(2,m)$, C'est un code de longueur $n = 2^m$, de dimension $k = 1 + \binom{m}{1} + \binom{m}{2}$, et de distance minimale $d = 2^{m-2}$.

Un mot de $RM(2,m)$ correspond à une fonction $f(x)$ de degré deux de la forme :

$$\begin{aligned} f_a(x) &= f_a(x_1, \cdots, x_m) \\ &= \textstyle\sum_{i<j} a_{ij} x_i x_j + \sum_i a_i.x_i + a_0 \ , \ i,j = 1 \cdots m, a_{ij}, a_i \in \mathbb{F}_2. \end{aligned}$$

la fonction d'encodage est la suivante :

$$\begin{aligned} C_m : \mathbb{F}_2^k &\rightarrow \mathbb{F}_2^n \\ a &\mapsto ev(f_a) = \Big(\ f_a(\alpha_1), \cdots, f_a(\alpha_n) \Big) \end{aligned}$$

tels que

$$\{\alpha_1, \cdots, \alpha_n\} = \mathbb{F}_2^m$$

et

$$a = (a_0, a_1, \cdots, a_m, a_{12}, a_{13}, \cdots, a_{1m}, a_{23}, \cdots, a_{2m}, \cdots, a_{m-1.m})$$

Après transmission du vecteur de code $ev(f_a) \in RM(2,m)$ nous recevons le vecteur $y = (y_{\alpha_1}, \cdots, y_{\alpha_n})$,

Cherchons l'effet de la dérivée sur les mots de code $RM(2,m)$. Nous avons :

$$\begin{aligned} f_a(x) &= f_a(x_1, \cdots, x_m) \\ &= \textstyle\sum_{i<j} a_{ij} x_i x_j + \sum_i a_i x_i + a_0 \ , \ i,j = 1 \cdots m, a_{ij}, a_i \in \mathbb{F}_2. \\ &= x.L.x^t + l.x^t + a_0 \end{aligned}$$

Où :

$$\begin{aligned} l &= (a_1, \ldots, a_m) \\ L &= \begin{pmatrix} 0 & a_{12} & \cdots & a_{1m} \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & a_{m-1,m} \\ 0 & \cdots & 0 & 0 \end{pmatrix} \end{aligned}$$

$$\begin{aligned} D_\alpha(f)(x) &= f(x) + f(x+\alpha) \\ &= x.L.x^t + l.x^t + a_0 + (x+\alpha).L.(x+\alpha)^t + l.(x+\alpha)^t + a_0 \\ &= a_0 + f(\alpha) + \alpha.(L + L^t).x^t \\ &= t_\alpha + \alpha.B.x^t \end{aligned}$$

avec

$$B = L + L^t = \begin{pmatrix} \overbrace{0}^{B_1^t} & \overbrace{a_{12}}^{B_2^t} & \cdots & \overbrace{a_{1m}}^{B_m^t} \\ a_{12} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & a_{m-1,m} \\ a_{1m} & \cdots & a_{m-1,m} & 0 \end{pmatrix}$$

$$\alpha = (\alpha_1, \cdots, \alpha_m) \quad x = (x_1, \cdots, x_m)$$

où $t_\alpha = a_0 + f(\alpha) \in \mathbb{F}_2$, et $\alpha.B = (\alpha.B_1^t, \ldots, \alpha.B_m^t) \in \mathbb{F}_2^m$.

Le vecteur $D_\alpha(f) = t_\alpha + \alpha.B.x^t$, est un mot de code $RM(1,m)$. Nous nous intéressons à la partie linéaire des dérivées de la fonction $(\alpha.B)$.
Soit B_i la ligne i de la matrice B, la matrice $\left((\alpha_1.B^t)^t, (\alpha_2.B^t)^t, \ldots, (\alpha_n.B^t)^t \right) \in \mathcal{M}_{m\times n}(\mathbb{F}_2)$ a la forme suivante :

$$\begin{pmatrix} \overbrace{\alpha_1.B_1^t}^{(\alpha_1.B)^t} & \overbrace{\alpha_2.B_1^t}^{(\alpha_2.B)^t} & \cdots & \overbrace{\alpha_n.B_1^t}^{(\alpha_n.B)^t} \\ \alpha_1.B_2^t & \alpha_2.B_2^t & \cdots & \alpha_n.B_2^t \\ \vdots & \vdots & \cdots & \vdots \\ \alpha_1.B_m^t & \alpha_2.B_m^t & \cdots & \alpha_n.B_m^t \end{pmatrix}$$

Les lignes de cette matrice sont des mots du code Reed-Muller d'ordre-1.

Regardons maintenant comment il est possible de décoder un mot erroné $y = c+e,\ c \in RM(2,m)$ où e est le vecteur d'erreur. Le décodage c'est retrouver les bits d'informations $a \in \mathbb{F}_2^k$, tels qu'on estime que $ev(f_a) = c$. Nous illustrons le décodage par le schéma 4.1. et nous décrivons la procédure de décodage comme suit :

Mot du code	Vecteur erroné
$f_a(x) = x.L.x^t + l.x^t + a_0 \in RM(2,m)$	$y = c + e \ \in \mathbb{F}_2^n$, $c = ev(f_a)$
$\forall \alpha \in \{\alpha_1, \ldots, \alpha_n\}$ $D_\alpha(f)(x) = t_\alpha + \alpha.B.x^t \in RM(1,m)$, $\alpha.B^t \in \mathbb{F}_2^m$	$\forall \alpha \in \{\alpha_1, \ldots, \alpha_n\}$ $D_\alpha(y) = D_\alpha(c) + D_\alpha(e) \Rightarrow$ $b_\alpha = MDD(D_\alpha(y)) = \alpha.B + \tilde{e}_\alpha$
$(\alpha_1.B_1^t, \ldots, \alpha_n.B_1^t) \in RM(1,m)$ $\vdots$ $(\alpha_1.B_m^t, \ldots, \alpha_n.B_m^t) \in RM(1,m)$	$y_{B_1} = \Big(\ (b_{\alpha_1})_1, \ldots, (b_{\alpha_n})_1 \Big) \ \overset{MDD}{\rightarrow} \tilde{B}_1$ $\vdots$ $y_{B_m} = \Big(\ (b_{\alpha_1})_m, \ldots, (b_{\alpha_n})_m \Big) \ \overset{MDD}{\rightarrow} \tilde{B}_m$
$a_{ij} = (B_i)_j = (B_j)_i$	$d_l(f,g) \overset{\triangle}{=} \min\{d_H(f,g), d_H(f,g+1)\}, \ f,g \ \in \ \mathbb{F}_2^n$ $\tilde{a}_{ij} = \begin{cases} (\tilde{B}_i)_j & \text{si} \quad d_l(y_{B_i}, x.\tilde{B_i}^t) < d_l(y_{B_j}, x.\tilde{B_j}^t) \\ (\tilde{B}_j)_i & \text{sinon} \end{cases}$
$f_a(x) + x.L.x^t = l.x^t + a_0 \in RM(1,m)$	$y + ev(\sum_{i<j} \tilde{a}_{ij}.x_i.x_j) \overset{MDD}{\rightarrow} \tilde{a}_0, \tilde{a}_1, \ldots, \tilde{a}_m$

Soit y le mot reçu, nous commençons par calculer sa dérivée $D_\alpha(y)$ dans toutes les directions possibles $\alpha \in \mathbb{F}_2^m$, nous décodons ces dérivées par un algorithme MDD (pour les codes d'ordre-1 on sait décoder de façon optimale FHT), nous obtenons une estimation $b_\alpha \in \mathbb{F}_2^m$ de la combinaison linéaire $(\alpha.B)$ des bits d'informations d'ordre deux a_{ij} et de la direction de la dérivation α. Les vecteurs y_{B_i} composés des coordonnées i des estimations b_α sont des versions corrompues des mots d'ordre 1 $(ev(x.B_i^t) \in RM(1,m))$. En les décodant avec un algorithme optimal nous obtenons les estimations $\tilde{B}_i$ dont la coordonnée j est une estimation du bit d'information a_{ij}.

Ayant deux estimations pour chaque bit d'information, nous faisons le choix selon la distance du résultat du décodage $\tilde{B}_i$ et du vecteur corrompu y_{B_i}.
Dans les vecteurs y_{B_i} nous nous intéressons à la composante linéaire, c'est pourquoi nous choisissons la distance d_l invariante par la translation $+1$, c'est-à-dire que si f et g sont deux fonctions booléennes alors :

$$d_l(f,g) \overset{\triangle}{=} \min\{d_H(f,g), d_H(f,g+1) = \min\{d_H(f,g), n - d_H(f,g)\} \ , \ f,g \in \mathbb{F}_2^n \ .$$

Nous définissons la sûreté notée R d'un résultat de décodage par la distance d_l entre le mot du code retourné par l'algorithme et l'entré y (le mot du code erroné). La sûreté a sa valeur maximum si les deux vecteurs sont identiques ou complémentaires $R(f,f) = R(f,f+1) = n$, et dans le cas spécial des mots d'ordre-1, cela veut dire que les deux mots ont la même composante linéaire.

Nous avons utilisé la sûreté pour choisir une des deux valeurs estimées pour chaque bit d'information d'ordre deux a_{ij}. Regardons le schéma de l'algorithme dans la figure-4.1. Nous pouvons associer aussi une valeur de sûreté pour chaque résultat de décodage,

en particulier ici pour les b_α, et utiliser cette valeur dans l'algorithme dans la suite du processus de décodage.
Nous définissons la sûreté dans une direction α par la formule :

$$R_\alpha = R\Big(\ D_\alpha(y)\ ,\ ev(b_\alpha.x)\Big)\quad ,\ b_\alpha = MDD_{RM_1}(D_\alpha(y))\quad .$$

Ainsi, nous avons une information en plus à exploiter ; nous avons vu le décodage souple dans le premier chapitre et dans le deuxième nous avons décrit des algorithmes de décodage souple, notamment l'algorithme FHT et les algorithmes récursifs. Nous allons maintenant décrire l'algorithme de décodage souple des codes de Reed-Muller d'ordre deux.

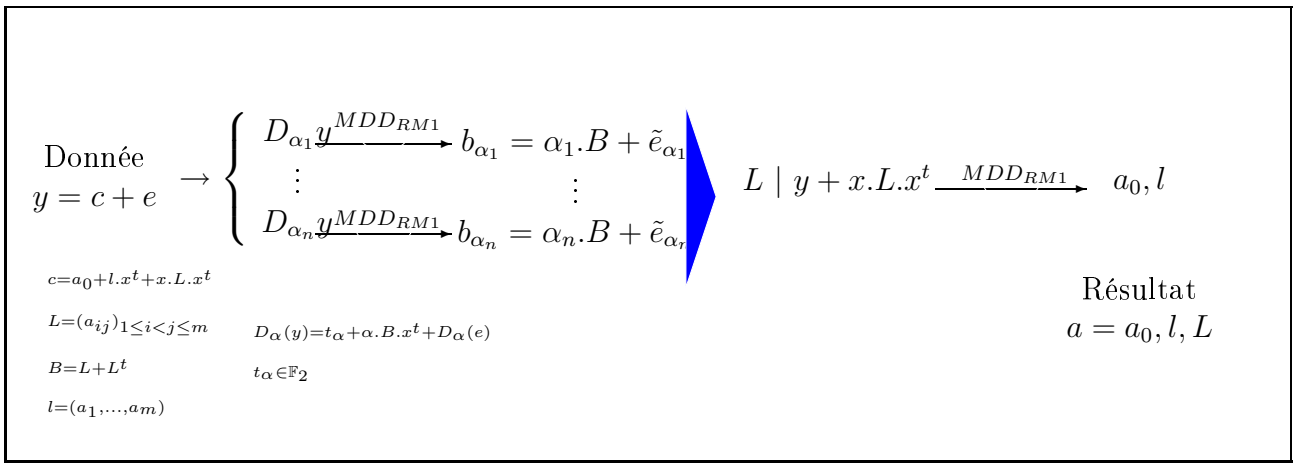


FIG. 4.1 – Schéma de l'algorithme de décodage des codes de Reed-Muller d'ordre deux

4.3 Algorithme de décodage

L'algorithme fait du décodage souple et dur, ainsi nous considérons le changement d'un symbole binaire $a \Leftrightarrow (-1)^a,\ a \in \mathbb{F}_2$. De cette manière, nous supposons qu'un mot du code $RM(r,m)$ appartient à l'espace $\{1,-1\}^n$ ce qui signifie que la somme dans le corps à deux éléments $\mathbb{F}_2$ sera transformée en le produit de leurs images :

$$a + b \Leftrightarrow (-1)^a(-1)^b = (-1)^{a+b} \mid a, b \in \mathbb{F}_2$$

L'algorithme de décodage souple prend en entrée le vecteur réel reçu $Y = (Y_{\alpha_1}, \cdots, Y_{\alpha_n}) \in \mathbb{R}^n$. Dans ce cas la dérivée vaut :

$$D_\alpha(Y) = (Y_{\alpha_1+\alpha}Y_{\alpha_1}, \cdots\cdots, Y_{\alpha_n+\alpha}Y_{\alpha_n})$$

et se déroule de la manière suivante :

Algorithme 10 $SP(y,m)$.
Donnée : $Y=(Y_{\alpha_1},\ldots,Y_{\alpha_n})\in\{-1,1\}^n$.
Résultat :$a=(a_0,a_1,\ldots,a_m,a_{12},a_{13},\ldots,a_{1m},a_{23},\ldots,a_{2m},\ldots,a_{m-1.m})$

1. Pour tout $\alpha\in\mathbb{F}_2^m,\alpha\neq 0$, calculer

$$Z^\alpha=(Y_{\alpha_1+\alpha}Y_{\alpha_1},\cdots\cdots,Y_{\alpha_n+\alpha}Y_{\alpha_n}),\ Z_i^\alpha=Y_{\alpha_i+\alpha}Y_{\alpha_i}$$

2. Pour tout $\alpha\in\mathbb{F}_2^m,\alpha\neq 0$, trouver la fonction linéaire

$$b_\alpha(x)=b_\alpha.x^t=\sum_{i=1}^{m}(b_\alpha)_i\ x_i$$

qui maximise la fonctionnelle

$$|\Big(Z^\alpha,C(x)\Big)|=|\sum_{i=1}^{n}Z_i^\alpha.(-1)^{C.\alpha_i}|\ ,\quad C\in\mathbb{F}_2^m$$

On note $b_\alpha=\Big(\ (b_\alpha)_1,\cdots,(b_\alpha)_m\Big)$, $R_\alpha=|\Big(Z^\alpha,b_\alpha(x)\Big)|$, et $R_0=0$.

3. Pour $i=1\cdots m$, calculer les fonctions linéaires $B_i(x)=B_i.x^t=\sum_{j=1}^{m}(B_i)_jx_j$, avec $(B_i)_i=0$ qui maximisent la fonctionnelle

$$T_i(a(x))=\sum_{j=1}^{n}R_{\alpha_j}(-1)^{a.{\alpha_j}^t+(b_{\alpha_j})_i}$$

tel que a est une fonction linéaire de m variables $a(x)=a.x^t$
Calculer $d_i=T_i(B_i(x))$, et noter que

$$B_i=((B_i)_1,\cdots,(B_i)_{i-1},0,(B_i)_{i+1},\cdots,(B_i)_m)$$

4. Trouver a_{ij}, avec

$$a_{ij}=\begin{cases}(B_i)_j & si\quad d_i\geq d_j\\ (B_j)_i & si\quad d_i<d_j\end{cases}$$

ces valeurs définissent $\pi(x)=\sum_{i<j}a_{ij}x_ix_j$.

5. Chercher la fonction linéaire $l_0(x)=\sum_{i=1}^{m}a_ix_i+a_0$ qui maximise la fonctionnelle $(Y,\pi(x)+l(x))$.
Le résultat de décodage est le vecteur

$$a=(a_0,a_1,\cdots,a_m,a_{12},a_{13},\cdots,a_{1m},a_{23},\cdots,a_{2m},\cdots,a_{m-1.m})$$

tel que : $f_a(x)=\pi(x)+l_0(x)$

4.4 Analyse de l'algorithme SP

Nous allons, dans un premier temps, faire l'analyse de l'algorithme. Cette analyse est similaire à celle qui est faite dans 3.2.2.2. Ensuite nous calculons la probabilité de correction en fonction du poids de vecteur d'erreur.

Nous transmettons $ev(f_a)$ et soit $y = (y_{\alpha_1}, \cdots, y_{\alpha_n})$ le mot reçu. Le décodage c'est retrouver les bits d'informations $(a_0, a_1, \cdots, a_m, a_{12}, \cdots, a_{1m}, a_{23}, \cdots, a_{m-1,m})$ à partir du vecteur y. La fonction f_y associée au vecteur y est définie de la façon suivante :

$$\begin{array}{rcl} f_y : \mathbb{F}_2^m & \rightarrow & \mathbb{F}_2 \\ \alpha & \mapsto & y_\alpha \end{array}$$

Définissons l'opérateur U suivant :

$$\begin{array}{rcl} U : \mathcal{F}_m & \rightarrow & \{-1,1\}^n \\ f & \mapsto & U_f = \Big(\ (-1)^{f(\alpha_1)}, \cdots, (-1)^{f(\alpha_n)} \Big) \end{array}$$

Et comme $\mathcal{F}_m \equiv \mathbb{F}_2^n$, nous donnons la définition de l'opérateur U sur l'espace vectoriel $\mathbb{F}_2^n$:

$$\begin{array}{rcl} U : \mathbb{F}_2^n & \rightarrow & \{-1,1\}^n \\ a & \mapsto & U_a = \Big(\ (-1)^{a_{\alpha_1}}, \cdots, (-1)^{a_{\alpha_n}} \Big) \end{array}$$

- L'entrée de l'algorithme est le vecteur $Y = U_{f_y}$.
- La première étape de l'algorithme c'est le calcul de Z^α, pour tout $\alpha \in \mathbb{F}_2^m$.
  le vecteur reçu $y = f_y$ est le vecteur f_a sur lequel des erreurs de transmission ont été rajoutées.

  $$f_y = f_a + e, \quad e = (e_{\alpha_1}, \cdots, e_{\alpha_n})$$

  donc :

  $$\begin{array}{lcl} U_{f_y} & = & U_{f_a+e} = U_{f_a}.U_e \\ D_\alpha(e) & = & (e_{\alpha_1} + e_{\alpha_1+\alpha}, \cdots, e_{\alpha_n} + e_{\alpha_n+\alpha}) \\ D_\alpha(f_a)(x) & = & t_\alpha + \alpha.B.x^t \ , \ t_\alpha \in \mathbb{F}_2, \ l_\alpha(x) = \alpha_B.x^t \end{array}$$

  et le vecteur Z^α s'écrit alors :

  $$\begin{array}{rcl} Z^\alpha & = & U_{f_y(x)+f_y(x+\alpha)} \\ & = & U_{D_\alpha(f_y)} \\ & = & U_{D_\alpha(f_a)+D_\alpha(e)} \\ & = & U_{D_\alpha(f_a)+\hat{e}} \ , \quad \hat{e} = D_\alpha(e) \\ & = & (-1)^{t_\alpha} \ U_{b_\alpha(x)}.U_{\hat{e}} \ , \quad b_\alpha(x) = \alpha.B.x^t \end{array}$$

- La deuxième étape qui consiste à maximiser la fonctionnelle $|(Z^\alpha, C(x))|$, pour $C(x) = C.x$ une fonction linéaire. Ce calcul est identique à celui de la transformée d'Hadamard[2] et nous avons :

  $$\begin{array}{rcl} \Big(\ Z^\alpha, C(x) \Big) & = & \sum_{\beta \in \mathbb{F}_2^m} Z^\alpha_\beta.(-1)^{C.\beta} \\ & = & \widehat{Z^\alpha}(C) \end{array}$$

[2]La transformée d'Hadamard est définie dans 3.2.2.2

le vecteur Z^α est le vecteur erroné du vecteur $U_{D_\alpha(f_a)}$,
Ce qui implique :

$$\max_{C\in\mathbb{F}_2^m} |(Z^\alpha, C(x))| = \max_{C\in\mathbb{F}_2^m} |\widehat{Z^\alpha}(C)|$$

Par conséquence, le résultat noté $b_\alpha(x)$ est la fonction linéaire à distance minimale de y
Notons que le terme R_α dans l'algorithme signifie :

$$R_\alpha = |\widehat{Z^\alpha}(b_\alpha)| = |n - 2.d\Big(\ Z^\alpha(x), b_\alpha(x)\Big)\ |$$

Dans le cas de transmission sans erreur $e = 0$ la valeur de R_α sera le plus élevée puisque $\forall C \in \mathbb{F}_2^m,\ \widehat{Z^\alpha}(\alpha.B) \geq \widehat{Z^\alpha}(C)$, dans ce cas $R_\alpha = n$

- L'expression utilisée dans la troisième étape pour décoder les m mots

$$\Big(\ (b_{\alpha_1})_i, \cdots, (b_{\alpha_n})_i\Big)\quad ,\ i = 1\cdots m$$

est pondérée par les coefficients de sûreté $(R_{\alpha_1}, \ldots, R_{\alpha_n})$

$$T_i(a(x)) = \sum_{j=1}^{n} R_{\alpha_j}(-1)^{a(\alpha_j)+(b_{\alpha_j})_i}\ ,\ a(x) = a.x^t$$

- Ensuite nous trouvons dans la troisième étape les estimations des bits d'informations (a_{ij}).
- Et dans les deux dernières étapes Nous retrouvons les estimations des $a_0, \cdots, a_{m-1}$, avec l'algorithme de décodage à distance minimale.

Nous allons maintenant étudier la complexité de cet algorithme ainsi que sa capacité de correction, qui est évaluée par le calcul de la borne de décodage asymptotique.

Théorème 4.4.1 *Pour le code de Reed-Muller d'ordre deux $RM(2, m)$, de grande longueur $n = 2^m$, l'algorithme $SP(., m)$ a une borne de décodage asymptotique :*

$$SP_2 = \frac{n}{2}\left(1 - \sqrt[4]{4\ln(2)\frac{m}{n}}\ \right)$$

Ce qui signifie : La probabilité que l'algorithme décode un mot du code erroné avec une erreur de poids $t < SP_2$, tend vers 1.

▷ Preuve : Soit $p = \frac{1-\epsilon}{2}, \epsilon > 0$, la probabilité d'erreur dans un canal binaire symétrique, $f(x) = \sum_{i<j} a_{ij}.x_i.x_j + \sum_i a_i.x_i + a_0\ , i, j = 1\cdots m$, $ev(f) = \Big(f(\alpha_1), \cdots, f(\alpha_n)\Big), f \in RM(2, m)$, après transmission de ce vecteur sur un canal BSC, nous recevons le vecteur

$$y = f + e, e = (e_{\alpha_1}, \cdots, e_{\alpha_n})$$

tel que :

$$\Pr[e_\alpha = 1] = p\ ,\ \forall\alpha \in \mathbb{F}_2^m$$

Notons $\widehat{e} = D_\alpha(e) = (e_{\alpha_1} + e_{\alpha_1+\alpha}, \cdots, e_{\alpha_n} + e_{\alpha_n+\alpha})$, et nous obtenons ainsi

$$D_\alpha(y) = D_\alpha(f) + \widehat{e}$$

Pour $\alpha \in \mathbb{F}_2^m$, $\alpha \neq 0$, soit $w = w(\widehat{e})$, .
Nous allons calculer l'espérance et la variance de la variable aléatoire w. les e_{α_i} est une réalisation d'une variable aléatoire X suivant la loi de $Bernoulli(p)$, Ainsi

$$\forall \alpha \in \mathbb{F}_2^m \ , \ \Pr[e_\alpha = 1] = \Pr[X = 1] = p$$

Pour $\alpha \neq 0$, les deux réalisations e_β et $e_{\beta+\alpha}$, de la variable aléatoire X sont indépendantes, et nous pouvons exprimer la probabilité d'erreur dans le vecteur dérivé $D_\alpha(e)$ en fonction de p :

$$\begin{array}{rcl} \widehat{p} & = & \Pr[\widehat{e}_\beta = 1] \\ & = & \Pr[e_\beta + e_{\beta+\alpha} = 1] \\ & = & \Pr[e_\beta = 1]\Pr[e_{\beta+\alpha} = 0] + \Pr[e_\beta = 0]\Pr[e_{\beta+\alpha} = 1] \\ & = & 2p(1-p) \\ & = & \frac{1}{2}(1-\epsilon^2) \end{array}$$

Ainsi l'erreur dans le vecteur $D_\alpha(e)$ est la réalisation d'une variable aléatoire, qu'on note $\widehat{X}$ suivant la loi de Bernoulli avec le paramètre $p = \frac{1}{2}(1-\epsilon^2)$.

Soit t le poids du vecteur d'erreur e $(wt(e) = t)$, donc

$$0 \leq w = wt(\widehat{e}) \leq 2t$$

Puisque nous avons l'égalité suivante :

$$e_{\alpha_i} + e_{\alpha_i+\alpha} = e_{(\alpha_i+\alpha)} + e_{(\alpha_i+\alpha)+\alpha}$$

Soit $w = wt(\widehat{e})$, donc $\frac{w}{2}$ est la somme de $\frac{n}{2}$ variables aléatoires indépendantes, ce qui implique que $\frac{w}{2}$ suit la loi binomiale (voir Annexe-A), la moyenne μ_w et la variance σ_w^2 valent :

$$\begin{array}{rcl} \mu_w & = & E(2\widehat{X}) \\ & = & 2\ E(\widehat{X}) \\ & = & \frac{n}{2}\ \widehat{p} \\ & = & \frac{n}{2}(1-\epsilon^2) \\ \sigma_w^2 & = & Var(2\widehat{X}) \\ & = & 4\ Var(\widehat{X}) \\ & = & 4\ \frac{n}{2}\widehat{p}(1-\widehat{p}) \\ & = & \frac{n}{2}(1-\epsilon^4) \end{array}$$

Le vecteur dérivé du mot reçu y, est décoder avec un algorithme MDD de $RM(1,m)$ (algorithme-4), c'est un algorithme de décodage à distance minimale, alors d'après le théorème-3.2.1 (voir définition-3.2.2), nous savons que la borne de décodage asymptotique du code de Reed-Muller d'ordre-1 vaut :

$$ML_1 = \frac{n}{2}\left(1 - 2\sqrt{\ln(2)\frac{m}{n}}\right)$$

Calculons, maintenant, la probabilité que le poids du vecteur d'erreur w dans le vecteur dérivé reste borné, quand $m \mapsto \infty$:

$$\begin{aligned}\Pr[w \geq ML_1] &= \Pr[w - \mu_w \geq ML_1 - \mu_w] \\ &= \Pr\left[w - E(w) \geq \underbrace{\frac{n}{2}\left(\epsilon^2 - 2\sqrt{\ln(2)\frac{m}{n}}\right)}_{\delta}\right]\end{aligned}$$

pour $\epsilon = \sqrt{\ell\sqrt{\frac{m}{n}}}$, où, ℓ est une constante, $\ell > 2\sqrt{\ln(2)}$, en utilisant l'inégalité de Chebyshev (voir Annexe-A), nous obtenons :

$$\begin{aligned}\Pr[w \geq ML_1] &\leq (\tfrac{\sigma_w^2}{\delta^2}) \\ &\leq \tfrac{1}{2}\,\frac{\frac{n}{2}(1-\epsilon^4)}{(\frac{n}{2}(\epsilon^2 - 2\sqrt{\ln(2)\frac{m}{n}}))^2} \\ &\leq \frac{1}{m\left(\ell - 2\sqrt{\ln(2)}\right)^2} \xrightarrow[m\to\infty]{} 0\end{aligned}$$

Nous avons en résultat qu'en choisissant α aléatoirement dans $\mathbb{F}_2^m$, avec une grande probabilité nous aurons un vecteur $D_\alpha(y)$ où le poids d'erreur dans ce vecteur est plus petit que la borne de décodage asymptotique de l'algorithme FHT, et il sera correctement décodé et nous arrivons à retrouver tous les bits d'informations d'ordre deux $a_{ij}, 1 \leq i < j \leq m$.
Ensuite nous décodons le vecteur $y + ev(\sum_{i<j} a_{ij}.x_i.x_j)$ avec l'algorithme FHT, dans ce vecteur le poids de l'erreur $wt(e) = \frac{n}{2}(1 - \sqrt{\ell\sqrt{\frac{m}{n}}}) < ML_1$. alors il est corrigible, et nous retrouvons tous les bits d'informations.
C.Q.F.D ◁

Et en ce qui concerne la complexité de l'algorithme nous avons le lemme suivant.

Lemme 4.4.1 *Pour le code $RM(2,m)$, $m > 2$, $n = 2^m$, l'algorithme de décodage $SP(.,m)$ a la complexité suivante :*

$$\begin{aligned}|SP(.,m)| &\leq n^2(2\log_2(n) + 1) + n(\log_2(n))^2 \\ &\approx O(n^2\log_2(n))\end{aligned}$$

▷ Preuve : La première étape de l'algorithme coûte n^2 additions modulo deux pour calculer les vecteurs Z_α, $\forall\ \alpha \in \mathbb{F}_2^m$.
Dans la deuxième étape nous utilisons l'algorithme de décodage en maximum de vraisemblance pour décoder les Z_α, cela revient à calculer le max de la fonctionnelle $|(Z(\alpha), C(x))|$, En conséquence le coût pour chaque α est de $n\log_2(n)$
La troisième, nécessite $n^2\log_2(n)$
Dans la quatrième nous faisons $\frac{m(m-1)}{2}$ comparaisons pour sélectionner les a_{ij}. et dans la dernière nous avons besoin d'encoder $\pi(x)$ qui vaut $n\ \frac{m(m-1)}{2}$ et n additions pour former le vecteur $y + \pi$ et enfin $n\ \log_2(n)$ opérations pour décoder $y + \pi$ et retrouver

tous les bits d'informations.

$$|SP(.,m)| = n^2 + n^2 \ \log_2(n) + n^2 \log_2(n) + (n+1)\frac{m(m-1)}{2} + n + n\log_2(n)$$

la complexité se réduit à $O(m\ n^2)$.

$\triangleleft$

Dans le calcul de la borne de décodage asymptotique nous avons utilisé le fait que dans la dérivée du mot reçu, l'erreur ne dépasse pas la borne de décodage de l'algorithme FHT du code $RM(1,m)$, ainsi nous pouvons atteindre la même borne avec un algorithme qui calcule un nombre suffisant de dérivées (en l'occurrence m) et c'est ce que nous allons voir tout de suite.

Il est très logique de croire que l'algorithme SP corrige plus loin que cette borne mais il n'est pas évident de prouver ce fait. Les résultats expérimentaux appuient notre intuition.

4.5 Les Variantes de l'algorithme de décodage

Nous allons exposer, dans ce qui suit, une optimisation possible pour cet algorithme pour réduire la complexité à $O(n\,m^2)$. L'algorithme de décodage $SP(.,m)$, revient à décoder n mots de $RM(1,m)$ (les dérivées du vecteur reçu Z_α, pour tout α dans $\mathbb{F}_2^m$). Nous savons que dans le calcul même de la borne de décodage asymptotique, pour tout mot erroné avec une erreur de poids plus faible que cette borne, nous retrouvons correctement toutes les dérivées. De ce fait, pas besoin de calculer toutes les dérivées, nous pouvons nous satisfaire d'une base de l'espace $\mathbb{F}_2^m$.

4.5.1 La base canonique de $\mathbb{F}_2^m$

Prenons $(e_i)_{i=1\cdots m}$ la base canonique de l'espace $\mathbb{F}_2^m$, pour une fonction d'ordre deux :

$$f(x) = \sum_{i<j} a_{ij}.x_i.x_j + \sum_i a_i.x_i + a_0\ ,\ i,j = 1\cdots m, a_{ij}, a_i \in \mathbb{F}_2.$$

Soit

$$B = \begin{pmatrix} 0 & a_{12} & \cdots & a_{1m} \\ a_{12} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & a_{m-1,m} \\ a_{1m} & \cdots & a_{m-1,m} & 0 \end{pmatrix}$$

et B_i^t la colonne i de la matrice B, $B_i = e_i B$ tel que $e_i = (\underbrace{0,\cdots,0}_{i-1},1,\underbrace{0,\cdots,0}_{m-i})$.

Par conséquent :

$$\begin{aligned} D_{e_i}(f)(x) &= a_i + e_i.B.x^t \\ &= a_i + B_i.x^t \end{aligned}$$

Donc pour un mot reçu $y = f_a + e$, le vecteur $D_{\alpha_i}(y)$, est un mot erroné du code $RM(1,m)$, nous décodons ce vecteur avec un algorithme de décodage à distance minimale, cela nous donne des estimations des bits d'informations a_{ij}.

Maintenant, nous allons retrouver la matrice B à partir des B_i. Nous obtenons les (a_i) avec un décodage à distance minimale ensuite.

D'où l'algorithme suivant :

Algorithme 11 SPS(y,m).

1. Pour $i = 1 \cdots m$, calculer $Z_{e_i} = (Y_{\alpha_1+e_i}Y_{\alpha_1}, \cdots\cdots, Y_{\alpha_n+e_i}Y_{\alpha_n})$
2. Pour $i = 1 \cdots m$, $B_i = FHT(Z_{e_i})$.
   Noter $B_i = ((B_i)_1, \cdots, (B_i)_m)$, $R_i = d_l(Z_{e_i}, ev(B_i.x^t))$ donnée aussi par l'algorithme FHT.
3. Trouver a_{ij}, avec
$$a_{ij} = \begin{cases} (B_i)_j & si \quad R_i \geq R_j \\ (B_j)_i & si \quad R_i < R_j \end{cases}$$
   ces valeurs définissent $\pi(x) = \sum_{i<j} a_{ij}x_ix_j$.
4. Chercher la fonction linéaire $l_0(x) = \sum_{i=1}^{m} a_ix_i + a_0$ qui maximise la fonction $(Y, \pi(x) + l(x))$.
   Le résultat de décodage est le vecteur
$$a = (a_0, a_1, \cdots, a_m, a_{12}, a_{13}, \cdots, a_{1m}, a_{23}, \cdots, a_{2m}, \cdots, a_{m-1.m})$$
   tel que : $f_a(x) = \pi(x) + l_0(x)$

La complexité de cet algorithme est $m^2n + m^2 + mn = O(m^2n)$.

Note 10 *La deuxième et la troisième étape de l'algorithme 10 ont été remplacées en une seule étape dans cet algorithme, dans cette deuxième étape nous trouvons deux valeurs probables pour chaque a_{ij} et nous garderons la valeur la plus probable en comparant les R_i.*

Théorème 4.5.1 *Pour le code de Reed-Muller d'ordre deux $RM(2,m)$, de longueur $n = 2^m$, l'algorithme $SPS(.,m)$ a la même borne de décodage asymptotique que celle de SP (algorithme-10) :*

$$SP_2 = \frac{n}{2}\left(1 - \sqrt{2\sqrt{\ln(2)}}\sqrt[4]{\frac{m}{n}}\right)$$

et l'algorithme corrige avec une grande probabilité toute erreur e de poids inférieur à cette borne $wt(e) < SP_2$.

▷ Preuve : Nous procédons de la même façon que dans le théorème-4.4.1, pour une transmission sur un CBS avec une probabilité d'erreur $p = \frac{1}{2}(1 - l\sqrt[4]{\frac{m}{n}})$ pour $l > \sqrt{2\sqrt{\ln(2)}}$. Nous avons vu que pour tout α dans $\mathbb{F}_2^m$, avec une grande probabilité nous aurons un vecteur $D_\alpha(y)$ où le poids d'erreur dans ce vecteur est plus petit que la borne de décodage asymptotique de l'algorithme FHT, et il sera correctement décodé et nous arrivons à retrouver tous les bits d'informations d'ordre deux $a_{ij}, 1 \leq i < j \leq m$. Ensuite nous décodons le vecteur $y + ev(\sum_{i<j} a_{ij}.x_i.x_j)$ avec l'algorithme FHT, dans ce vecteur le poids de l'erreur $wt(e) < ML_1$. alors il est corrigible, et nous retrouvons tous les bits d'informations.
◁

4.5.2 Algorithme de décodage paramétré

L'algorithme de décodage $SP(.,m)$ résout le problème de décodage d'un mot de $RM(2,m)$ en effectuant $n+m$ fois le décodage dans le code du premier ordre. Le code $RM(1,m)$ de longueur $n = 2^m$ est décodé avec l'algorithme-4 FHT, qui nous fournit pas seulement le mot le plus proche (ou le plus probable pour les canaux BSC, $AWGN$), mais aussi la distance à tous les mots du code. Nous pouvons donc sélectionner une liste des mots les plus proches.
Quand le poids du vecteur d'erreur est assez important dans la dérivée, le mot le plus proche du vecteur dérivé erroné n'est pas nécessairement la bonne valeur. Comme l'algorithme FHT peut nous donner la liste des mots les plus proches, la question se pose de comment pouvoir utiliser cette liste pour améliorer le décodage. Ainsi l'algorithme paramétré $SP(.,s,h,r)$ tient compte de la liste des s-mots les plus proches de chaque dérivée.
Soit $f_a = \sum_{i<j} a_{ij}.x_i.x_j + \sum_i a_i.x_i + a_0$ un mot du code en transmettant $c = ev(f_a)$ nous recevons le vecteur erroné $y = c + e$. Nous avons :

$$D_\alpha f_a(x) = t_\alpha + \alpha.B.x^t.$$

avec une constante, $t_\alpha \in \mathbb{F}_2$, et une matrice $B \in \mathcal{M}_{m\times m}$ composée des bits d'informations d'ordre deux $a_{ij}, 1 \leq i,j \leq m$. dans le mot dérivé, la partie linéaire $\alpha.B$ est essentielle à

l'algorithme de décodage, c'est une combinaison linéaire des bits d'informations. Notons l'équation de linéarité vérifiée par les parties linéaires des dérivées :

$$\alpha.B + \beta.B + (\alpha+\beta).B = 0 \quad , \alpha, \beta \ \in \ \mathbb{F}_2^m. \tag{4.1}$$

où la somme + est l'addition (composante à composante) dans l'espace vectoriel $\mathbb{F}_2^m$. C'est intéressant de pouvoir utiliser cette équation à fin d'améliorer le décodage. Cette équation sera utilisée pour augmenter ou diminuer la valeur de sûreté associée à chaque dérivée.

Nous allons maintenant décrire l'algorithme de décodage paramétré pour un mot reçu $y = c + e$, avec les deux paramètres $s \geq 1$ et $h \geq 0$:

1. pour une transmission sur un canal BSC, nous considérons en entrée le vecteur réel $Y = U(y)$ (voir équation-4.4 pour la définition).
2. Pour toutes les directions possibles $\alpha \in \mathbb{F}_2^m$, nous calculons la dérivée selon chaque direction $D_\alpha(Y)$, nous décodons ce vecteur avec l'algorithme FHT en prenant en compte les s mots les plus proches que nous notons $b_\alpha^1, \ldots, b_\alpha^s$ avec leurs valeurs de sûreté respective $R_\alpha^1, \ldots, R_\alpha^s$, rappelons que la valeur de sûreté vaut :

$$R_\alpha = |n - 2.d\Big(\ D_\alpha(y)(x), b_\alpha(x)\Big) \ |$$

3. Soit $D_0 \ < \ 1 \ < D_1$ deux constantes, effectuer h fois un raffinement sur les $R_\alpha^1, \ldots, R_\alpha^s$ qui dépend de la façon dont les mots b_α^i vérifient l'équation de linéarité 4.1, de la façon suivante :

$$\left| \begin{array}{l} R_\alpha^i(\beta) = \left\{ \begin{array}{l} D_0 R_\alpha^i \ \text{ si } \ \nexists 1 \leq j,k \leq m, \beta \ \in \ \mathbb{F}_2^m \ \mid \ b_\alpha^i + b_\beta^j + b_{\alpha+\beta}^k = 0 \\ D_1 \ \max\left\{ \frac{R_\alpha + R_\beta^j + R_{\alpha+\beta}^k}{3} \mid b_\alpha^i + b_\beta^j + b_{\alpha+\beta}^k = 0 \ , \ 1 \leq j,k \leq s \right\} \ \text{ sinon} \end{array} \right. \\ R_\alpha^i = (n-2)^{-1} \sum_{\beta \in \mathbb{F}_2^m / \{0,\alpha\}} \ R_\alpha^i(\beta) \end{array} \right.$$

   le but de cette étape dans l'algorithme est d'accroître la valeur de sûreté des b_α vérifiant l'équation de linéarité 4.1.
4. Choisir les b_α avec un maximum de sûreté :
   $b_\alpha = b_\alpha^i \ , R_\alpha = R_\alpha^i \mid R_\alpha^i \geq R_\alpha^j \ , \ 1 \leq j \leq s.$
5. Continuer le décodage comme dans l'algorithme-10 SP

Le schéma-4.2, montre le processus de décodage.

Les auteurs de l'article [50] n'ont pas analysé cet algorithme. Notez bien que pour des paramètres $s = 1, h = 0$ nous obtenons l'algorithme sous sa forme simplifiée présenté auparavant (algorithme-10) pour lequel nous avons analysé la capacité de correction. Le coût de l'algorithme $SP(., s, h, m)$ est $O\left((m + hs^3)n^2\right)$.

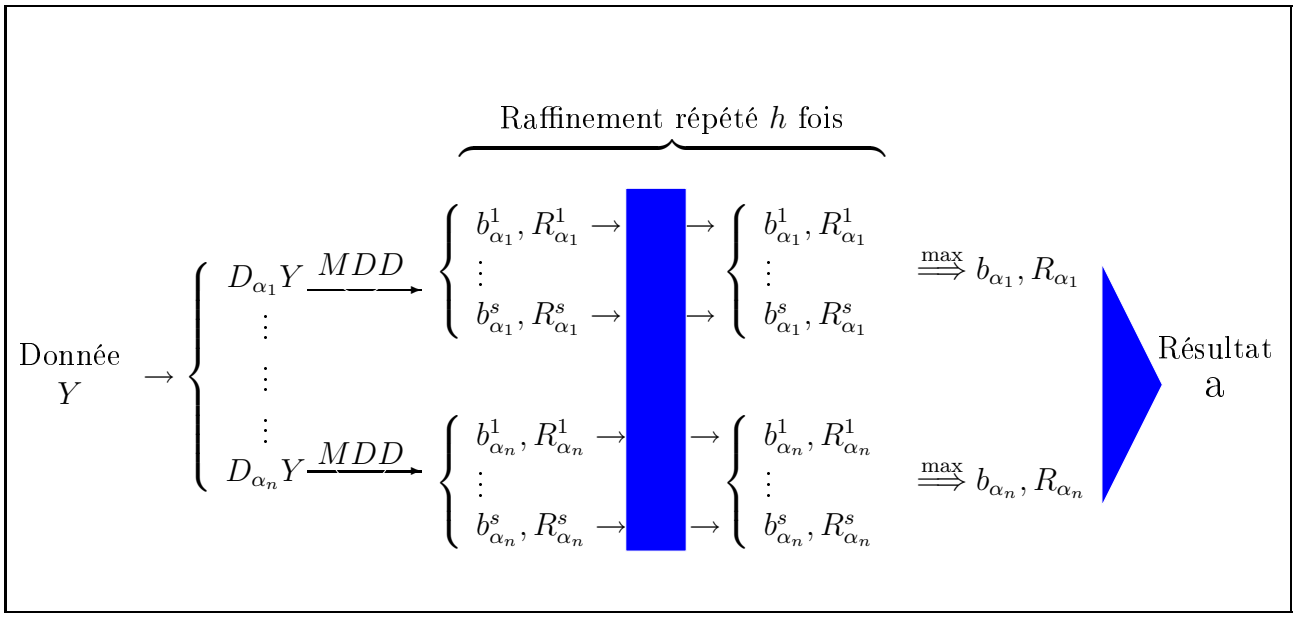


FIG. 4.2 – Schéma de l'algorithme paramétré $SP(., s, h, r)$ de décodage du code $RM(2, m)$

Chapitre 5

Algorithme de Sidel'nikov et Pershakov Modifié

Contenu

Ce chapitre a fait l'objet de deux publications [33] qui furent présentées à International Workshop on Algebraic and Combinatorial Coding Theory (ACCT) en Juin 2004, et [45] accepté à ITW2005 (IEEE ITSOC Information Theory Workshop 2005 on Coding and Complexity). Nous allons commenter le décodage des codes de Reed-Muller d'ordre deux pour donner une borne sur la probabilité de bien décoder dans $RM(2,m)$ une dérivée d'un vecteur au delà de la borne de décodage asymptotique des algorithmes récursifs. Nous ferons ensuite une étude sur le décodage souple. Nous définissons ensuite un code non binaire qui nous servira pour commenter l'algorithme SP, et nous décrivons notre algorithme en donnant une explication à la modification apportée à l'algorithme d'origine. Nous donnons une preuve que les algorithmes SP et le nôtre font du décodage borné (jusqu'à capacité de correction).

5.1 Commentaire général sur le décodage de $RM(2,m)$

Nous avons présenté dans ce manuscrit tous les algorithmes existant jusqu'à maintenant pour le décodage du code $RM(2,m)$. Ils sont basés tous sur la dérivation du vecteur à décoder, cette opération fait augmenter le bruit d'un facteur carré (voir FIG.5.1). Nous établissons le lemme suivant :

Lemme 5.1.1 *Pour une transmission sur un canal BSC* $\left(p=\frac{1}{2}(1-\epsilon)\right)$ *avec* $\epsilon \leq \sqrt[4]{\ell\frac{m}{n}}$ *où* $\ell < 4\ln(2)$. *La probabilité de bien décoder dans* $RM(1,m)$ *la dérivée du vecteur reçu tend vers zéro quand la longueur du code tend vers l'infini.*

▷ Preuve : La borne du décodage asymptotique du code $RM(1,m)$ est $ML_1 = \frac{n}{2}\left(1-\sqrt{4\ln(2)\frac{m}{n}}\right)$ (définition-3.2.2). Soit w la variable aléatoire représentant le poids de l'erreur dans la dérivée, elle suit la loi binomiale avec $\mu_w = \frac{n}{2}(1-\epsilon^2)$ et $\sigma_w^2 = \frac{n}{2}(1-\epsilon^4)$, nous avons :

$$\begin{aligned}
\Pr[w < ML_1] &= \Pr[\mu_w - w > \mu_w - ML_1] \\
&< \Pr[|w-\mu_w| \geq \mu_w - ML_1] \\
&\leq \frac{\sigma^2}{(\mu_w - ML_1)^2} \\
&\leq \frac{n}{2|\mu_w - ML_1|^2} \\
&\leq \frac{1}{cm}, \quad c = \frac{(\sqrt{4\ln(2)}-\sqrt{\ell})^2}{2}
\end{aligned}$$

Une dérivée est décodée dans le code $RM(1,m)$ et elle sera correctement décodée avec un algorithme de décodage à maximum de vraisemblance si $w < ML_1$, ainsi la probabilité de bien décoder une dérivée vaut au plus $\frac{1}{cm}$, ce qui tend vers zéro quand $m \to +\infty$ ◁

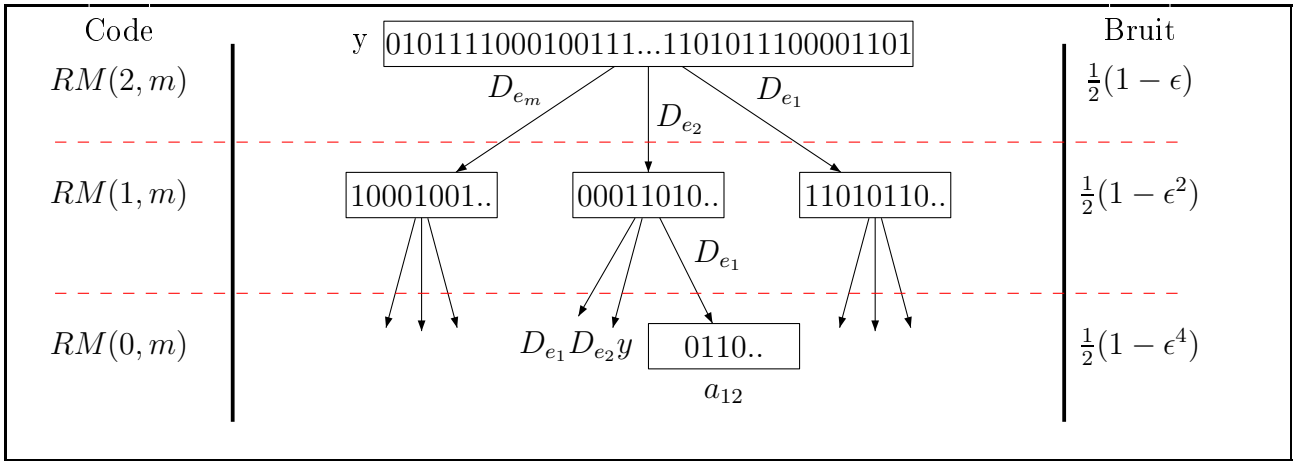


Fig. 5.1 – Décodage des codes de $RM(2,m)$

Ce lemme nous indique que l'algorithme de Dumer (décodage récursif) ne pourra pas décoder au delà de la capacité de correction des algorithmes SP. Le théorème-3.4.2 donne

$\frac{n}{2}(1-\sqrt[4]{16\frac{m}{n}})$ comme borne de décodage asymptotique pour le code $RM(2,m)$, mais ne nous précise pas si cette borne est stricte. Le lemme-5.1.1 donne une majoration de la borne de décodage asymptotique stricte de décodage récursif de l'algorithme-9 noté D_2 :

$$D_2 \leq \frac{n}{2}\left(1-\sqrt[4]{4\ln(2)\frac{m}{n}}\right)$$

5.2 Préliminaire sur le décodage souple

Nous considérons ici un canal de transmission probabiliste sans mémoire, que nous notons CTP, tel que à l'entrée nous avons des symboles binaires x_i, et la variable de sortie p_i prend ses valeurs dans un intervalle $[0,1]$. Pour chaque valeur reçue nous faisons le choix suivant :

- si $p_i \geq \frac{1}{2}$ alors $y_i = 1$, $p_1(y_i) = \Pr[Y = 1 \mid X = 1] = p_i$
- si $p_i < \frac{1}{2}$ alors $y_i = 0$, $p_0(y_i) = \Pr[Y = 0 \mid X = 0] = 1 - p_i$

Soit X,Y les variables aléatoires représentant l'entrée et la sortie respectivement :

$$\begin{array}{rcl} CTP : X & \longmapsto & Y \\ x_i & \longrightarrow & y_i, \quad p_{y_i} = \Pr[Y = y_i \mid X = y_i] \geq \frac{1}{2} \end{array} \tag{5.1}$$

Une fois que nous avons la probabilité que le bit reçu vaut un, nous avons la deuxième probabilité, puisque la probabilité est définie telle que $\Pr[y_i|1] + \Pr[y_i|0] = 1$. Nous notons $p_0(y_i) = \Pr[y_i|0]$ et $p_1(y_i) = \Pr[y_i|1]$.

Soient $\mathcal{C}$ un $[n,k]$-code binaire linéaire de longueur $n = 2^m$ et $\{\beta_1, \ldots, \beta_n\} = \mathbb{F}_2^m$. Après transmission d'un mot du code sur un canal CTP, nous recevons le vecteur $y = (y_{\beta_1}, \ldots, y_{\beta_n})$ avec les probabilités associées. Le décodage optimal (MLD) consiste à trouver un mot du code $c = (c_{\beta_1}, \ldots, c_{\beta_n})$ qui maximise la probabilité $\Pr[y|c]$ (voir 3.2) avec

$$\Pr[y|c] = \prod_{\beta \in \mathbb{F}_2^m}^{n} \Pr[y_\beta | c_\beta]$$

De manière similaire à l'étude du décodage souple fait dans [4], maximiser $\Pr[y|c]$ est équivalent à maximiser l'expression suivante :

$$\begin{array}{rcl} M(c) & = & a \sum_{\beta \in \mathbb{F}_2^m} \log \Pr[y_\beta|c_\beta] \; + b \\ & = & a \sum_{\beta \in \Lambda_0} \log p_0(y_\beta) + a \sum_{\beta \in \Lambda_1} \log p_1(y_\beta) + b \end{array}$$

Où a est un réel positif et b est un réel, et les deux ensemble $\Lambda_0 = \{\beta \; : \; c_\beta = 0\}$ et $\Lambda_1 = \{\beta \; : \; c_\beta = 1\}$.
Nous choisissons les valeurs des constantes de la façon suivante : $a = 2$ et

$$b = -\sum_{\beta \in \mathbb{F}_2^m} \Big(\; \log p_0(y_\beta) + \log p_1(y_\beta) \Big)$$

b est une constante par rapport à c. Ainsi nous écrivons l'expression à maximiser :

$$\begin{aligned} M(c) &= 2\sum_{\beta\in\Lambda_0}\log p_0(y_\beta) + 2\sum_{\beta\in\Lambda_1}\log p_1(y_\beta) - \\ &\quad \sum_{\beta\in\mathbb{F}_2^m}\Big(\ \log p_0(y_\beta) + \log p_1(y_\beta)\Big) \\ &= \sum_{\beta\in\Lambda_0}\log p_0(y_\beta) - \sum_{\beta\in\Lambda_1}\log p_0(y_\beta) + \\ &\quad \sum_{\beta\in\Lambda_1}\log p_1(y_\beta) - \sum_{\beta\in\Lambda_0}\log p_1(y_\beta) \\ &= \sum_{\beta\in\Lambda_0}\log\frac{p_0(y_\beta)}{p_1(y_\beta)} + \sum_{\beta\in\Lambda_1}\log\frac{p_1(y_\beta)}{p_0(y_\beta)} \end{aligned}$$

D'où :

$$M(c) = \sum_{\beta\in\mathbb{F}_2^m}\log\frac{p_0(y_\beta)}{p_1(y_\beta)}\,(-1)^{c_\beta} \tag{5.2}$$

Dans le canal CTP cette expression s'écrit :

$$M(c) = \sum_{\beta\in\mathbb{F}_2^m}\log\left(\frac{p_{y_\beta}}{1-p_{y_\beta}}\right)(-1)^{y_\beta}\,(-1)^{c_\beta} \tag{5.3}$$

Pour le canal CTP, nous avons $p_{y_\beta}\geq\frac{1}{2}$, donc $\log\left(\frac{p_{y_\beta}}{1-p_{y_\beta}}\right)\geq 0$
Soit $y=(y_{\beta_1},\ldots,y_{\beta_n})$ un vecteur reçu, et $p_y=(p_{y_{\beta_1}},\ldots,p_{y_{\beta_n}})$ les probabilités associées. Nous définissons la fonction f_y :

$$\begin{aligned} f_y:\mathbb{F}_2^m &\to \mathbb{R} \\ \beta &\mapsto \log\frac{p_0(y_\beta)}{p_1(y_\beta)} = \log\left(\frac{p_{y_\beta}}{1-p_{y_\beta}}\right)(-1)^{y_\beta} \end{aligned}$$

La transformée d'Hadamard de f_y (voir 3.2.2.2), notée $\widehat{f_y}$:

$$\begin{aligned} \widehat{f_y}(\alpha) &= \sum_{\beta\in\mathbb{F}_2^m} f_y(\beta)\,(-1)^{<\alpha,\beta>}, \text{ où } <\alpha,\beta>=\alpha.\beta^t=\beta_1\alpha_1+\cdots+\beta_m v_m \\ &= \sum_{\beta\in\mathbb{F}_2^m}\log\frac{p_0(y_\beta)}{p_1(y_\beta)}\,(-1)^{<\beta,\alpha>} \end{aligned}$$

Si nous cherchons la forme linéaire $C(x)=\alpha.x^t, C_\alpha = ev(C(x))$ qui maximise la probabilité $\Pr[y|C_\alpha]$, c'est aussi le mot qui maximise l'expression 5.2 ; et à l'aide de la transformée d'Hadamard nous écrivons la fonction de recherche de la forme linéaire la plus proche notée MLD_ℓ sur un canal CTP.

Algorithme 12 MLD_ℓ.
- Entrée : $y = (y_1, \ldots, y_n), y_i \in \mathbb{F}_2,\ P_y = (p_{y_1}, \ldots, p_{y_n}),\ p_{y_i} \in]0, 1[$.
- Sortie : $\alpha = (\alpha_1, \ldots, \alpha_m) \in \mathbb{F}_2^m, R_\alpha \in \mathbb{R}$

1. Calculer $Y = (Y_1, \ldots, Y_n),\ Y_i = \log\left(\frac{p_{y_i}}{1-p_{y_i}}\right)(-1)^{y_i}$
2. Calculer $\widehat{F}$
   pour $i = 1$ à m faire $Y = M_m Y^t$.

$$M_m = \begin{bmatrix}
1 & 1 & 0 & 0 & 0 & \ldots & \ldots & \ldots & \ldots & \ldots \\
0 & 0 & 1 & 1 & 0 & \ldots & \ldots & \ldots & \ldots & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 1 & 1 & 0 & 0 \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 0 & 0 & 1 & 1 \\
1 & -1 & 0 & 0 & 0 & \ldots & \ldots & \ldots & \ldots & \ldots \\
0 & 0 & 1 & -1 & 0 & \ldots & \ldots & \ldots & \ldots & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 1 & -1 & 0 & 0 \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & 0 & 0 & 1 & -1
\end{bmatrix}$$

3. Choisir α tel que Y_α soit maximum, et soit $(\alpha_1, \ldots, \alpha_m)$ l'écriture binaire de α, $R_\alpha = Y_\alpha$

La complexité de l'algorithme-12 est $O(m \times n)$.

5.2.1 MLD sur les canaux BSC et $AWGN$

Décoder à maximum de vraisemblance sur un canal probabiliste CTP c'est trouver le mot du code c qui maximise la probabilité de recevoir le vecteur y sachant que le mot c a été transmis. Nous avons démontré précédemment que c'est équivalent à maximiser la fonctionnelle $M(y, c)$, telle que :

$$M(y, c) = \sum_{\beta \in \mathbb{F}_2^m} \mu(y_\beta)\ (-1)^{c_\beta}$$

Avec $\mu(y_\beta) = \log \frac{p_0(y_\beta)}{p_1(y_\beta)}$. Mais les canaux BSC et $AWGN$ sont des canaux de transmission de type CTP, et dans ces deux cas nous allons décrire la fonctionnelle à maximiser.

5.2.1.1 Canal BSC

Pour une probabilité d'erreur $0 < p < \frac{1}{2}$, nous avons $p_0(1) = \Pr[1 \mid 0] = p$, $p_1(0) = p$ et $p_0(0) = p_1(1) = 1-p$ et ainsi $\mu(0) = -\mu(1) = \log\frac{1-p}{p}$. La fonctionnelle $M(y,c)$ s'écrit :

$$M(y,c) = \log\frac{1-p}{p} \sum_{\beta\in\mathbb{F}_2^m} (-1)^{y_\beta}\ (-1)^{c_\beta}$$

Par conséquent Le décodage MLD pour une transmission sur un canal $BSC(p)$, $p < \frac{1}{2}$ c'est maximiser la fonctionnelle $M(y,c) = \sum_{\beta\in\mathbb{F}_2^m}(-1)^{y_\beta}\ (-1)^{c_\beta}$. Pour les codes $RM(1,m)$ l'algorithme FHT (4) décrit en 3.2.2.2 est basée sur la maximisation de la fonctionnelle $M(y,c)$ et c'est un algorithme de décodage en maximum de vraisemblance.

5.2.1.2 Canal $AWGN$

Dans le cas d'un canal de transmission sans mémoire avec une sortie continue, les probabilités de transition sont $f_0(y_i), f_1(y_i)$, $y_i \in \mathbb{R}$ avec f la densité de probabilité. Le décodage MLD consiste à maximiser la probabilité $f(y \mid c) \stackrel{\triangle}{=} \prod_{\beta\in\mathbb{F}_2^m} f(y_\beta \mid c_\beta)$, et par la même méthode utilisée dans 5.2 nous trouvons que c'est équivalent à maximiser la fonctionnelle $M(y,c)$, pour $\mu(y_\beta) = \log\frac{f_0(y_\beta)}{f_1(y_\beta)}$.
Pour le canal gaussien (1.2.2) la densité de probabilité f est la suivante :

$$\begin{aligned} f_0(y) &= \tfrac{1}{\sigma\sqrt{2\,\pi}}\ \mathrm{e}^{-\frac{(y-1)^2}{2\sigma^2}} \\ f_1(y) &= \tfrac{1}{\sigma\sqrt{2\,\pi}}\ \mathrm{e}^{-\frac{(y+1)^2}{2\sigma^2}} \end{aligned}$$

Ainsi nous avons :

$$\log\frac{f_0(y_\beta)}{f_1(y_\beta)} = \frac{2}{\sigma^2}\ y_\beta$$

D'où le décodage en maximum de vraisemblance sur ce canal consiste à maximiser la fonctionnelle :

$$M(y,c) = \sum_{\beta\in\mathbb{F}_2^m} y_\beta\ (-1)^{c_\beta}$$

L'algorithme 4 de décodage FHT pour le code $RM(1,m)$ est MLD algorithme pour une transmission sur un canal $AWGN$ en considérant en entrée le vecteur reçu.

5.3 Décodage de $RM(2,m)$

Nous allons analyser l'algorithme de décodage SP décrit dans le chapitre précédent, Nous commençons par décrire un code C_ℓ, ses caractéristiques et le décodage dans ce code, ensuite nous montrons que décoder dans $RM(2,M)$ c'est décoder dans C_ℓ. Nous montrerons ce que signifie le terme "sûreté" utilisé dans l'algorithme SP. Ayant décrit le code C_ℓ nous décrivons la modification proposée pour pouvoir décoder plus loin en utilisant les caractéristiques de ce code.

5.3.1 code additif C_ℓ

Nous définissons le code C_ℓ suivant :

$$C_\ell = \{(\alpha_1 B, \ldots, \alpha_n B) \mid B \in \mathcal{M}_{m\times m}, B = B^t,\ \{\alpha_1, \ldots, \alpha_n\} = \mathbb{F}_2^m\} \tag{5.4}$$

L'alphabet de ce code c'est $\mathbb{F}_2^m$, il est de longueur n et de distance minimale $3\frac{n}{4}$ (puisque les matrices B sont symétriques), donc sa capacité de correction est $t_c = 3\frac{n}{8} - 1$. Ainsi, Il est possible de décoder correctement une erreur de poids $\leq t_c$, ce qui signifie que pour le code $RM(2,m)$ si le nombre de dérivées non corrigées ne dépasse pas t_c, il est possible de décoder correctement le mot. Nous n'avons pas songé à trouver un algorithme qui décode jusqu'à la capacité de correction de ce code (t_c), mais un moyen simple de faire le décodage c'est de se rendre compte que les m coordonnées de ce code sont des mots de $RM(1,m)$:

$$B = (B_1^t, \ldots, B_m^t)\ \ \forall\ i,\ 1 \leq i \leq m\ (\alpha_1.B_i^t, \ldots, \alpha_n.B_i^t) \in RM(1,m)$$

et avec les mêmes notations $\alpha B \in \mathbb{F}_2^m,\ \alpha B = (\alpha.B_1^t, \ldots, \alpha B_m^t)$

Soit $B \in \mathcal{M}_{m\times m}(\mathbb{F}_2)$, l'encodage de B dans le code C_ℓ est le vecteur $E_{C_\ell}(B) = (\alpha_1 B, \ldots, \alpha_n B) \in (\mathbb{F}_2^m)^n$. Ainsi la coordonnée α est l'image de $\alpha \in F_2^m$ par l'action de la transformation linéaire définie par la matrice B, d'où nous avons la propriété suivante.

Équation de linéarité : $\alpha B + \beta B + (\alpha + \beta)B = 0, \forall \alpha, \beta \in \mathbb{F}_2^m$.

5.3.2 Lien entre le décodage dans $RM(2,m)$ et dans C_ℓ

L'algorithme-10 SP présenté au chapitre précédent transforme le problème de décoder un mot dans le code de Reed-Muller d'ordre deux en plusieurs problèmes de décodage de mots dans des codes de Reed-Muller d'ordre un en utilisant la propriété-3.1.1 de diminution d'ordre de l'opérateur dérivée.
Un mot de $RM(2,m)$ correspond à une fonction $f(x)$ de degré deux de la forme :

$$\begin{array}{rcl} f_a(x) & = & f_a(x_1, \cdots, x_m) \\ & = & \sum_{i<j} a_{ij} x_i x_j + \sum_i a_i.x_i + a_0\ ,\ i,j = 1 \cdots m, a_{ij}, a_i \in \mathbb{F}_2. \end{array}$$

Les bits d'informations sont les coefficients a_0, a_i, a_{ij}, et le mot du code correspondant est l'évaluation de le fonction f_a (i.e $c = ev(f_a)$). Nous avons vu au chapitre précédent que la dérivée de la fonction f_a selon une direction $\alpha \in \mathbb{F}_2^m$ est un mot du $RM(1,m)$:

$$D_\alpha(f)(x) = t_\alpha + \alpha.B.x^t$$

avec

$$B = \begin{pmatrix} \overbrace{0}^{B_1^t} & \overbrace{a_{12}}^{B_2^t} & \cdots & \overbrace{a_{1m}}^{B_m^t} \\ a_{12} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & a_{m-1,m} \\ a_{1m} & \cdots & a_{m-1,m} & 0 \end{pmatrix}$$

$$\alpha = (\alpha_1, \cdots, \alpha_m) \quad x = (x_1, \cdots, x_m)$$

où $t_\alpha = a_0 + f(\alpha) \in \mathbb{F}_2$, et $\alpha.B = (\alpha.B_1^t, \ldots, \alpha.B_m^t) \in \mathbb{F}_2^m$. Le vecteur $D_\alpha(f) = t_\alpha + \alpha.B.x^t$, est un mot de code $RM(1,m)$, Nous nous intéressons à la partie linéaire des dérivées de la fonction $(\alpha.B)$.

Soit B_i la ligne i de la matrice B, la matrice $\Big((\alpha_1.B^t)^t, (\alpha_2.B^t)^t, \ldots, (\alpha_n.B^t)^t \Big) \in \mathcal{M}_{m\times n}(\mathbb{F}_2)$ a la forme suivante :

$$\begin{array}{c} \overbrace{}^{(\alpha_1.B)^t} \quad \overbrace{}^{(\alpha_2.B)^t} \quad \cdots \quad \overbrace{}^{(\alpha_n.B)^t} \\ \left(\begin{array}{cccc} \alpha_1.B_1^t & \alpha_2.B_1^t & \cdots & \alpha_n.B_1^t \\ \alpha_1.B_2^t & \alpha_2.B_2^t & \cdots & \alpha_n.B_2^t \\ \vdots & \vdots & \cdots & \vdots \\ \alpha_1.B_m^t & \alpha_2.B_m^t & \cdots & \alpha_n.B_m^t \end{array} \right) \end{array} \begin{array}{l} \\ \leftrightarrow ev\,(f_1(x) = x.B_1^t) \\ \leftrightarrow ev\,(f_2(x) = x.B_2^t) \\ \vdots \\ \leftrightarrow ev\,(f_m(x) = x.B_m^t) \end{array} \tag{5.5}$$

Les lignes de cette matrice sont des mots du code Reed-Muller d'ordre-1. Soit $b = (b_{\alpha_1}, \ldots, b_{\alpha_1})$ un vecteur erroné d'un mot de C_ℓ, avec $b_\alpha = ((b_\alpha)_1, \ldots, (b_\alpha)_m) \in \mathbb{F}_2^m$, donc selon 5.5 les vecteurs $b_i = ((b_{\alpha_1})_i, \ldots, (b_{\alpha_1})_i),\ i = 1 \ldots m$ peuvent être décodés dans le code $RM(1,m)$.

Remarque 3 *Pour décoder le vecteur b, nous pouvons envisager de diminuer le bruit dans les vecteurs b_i. Dans l'article [1] un algorithme de décodage pour les codes $RM(1,m)$ est décrit, c'est un décodeur MAP*[1]*, il minimise la probabilité d'erreur pour les symboles reçus séparément. Pour le vecteur b_i, MAP permet de trouver un vecteur $\widetilde{b_i}$ avec un bruit atténué. Nous avons implémenté cet algorithme et nous l'avons utilisé pour améliorer les performances de l'algorithme SP, mais les résultats expérimentaux ont montré que l'utilisation n'a pas amélioré les performances de décodage.*

Rappelons que le décodage MLD du code $RM(1,m)$ dans le canal BSC consiste à maximiser l'expression :

$$M_d(b_i, c) = \sum_{\beta \in \mathbb{F}_2^m} (-1)^{(b_\beta)_i}\ (-1)^{c_\beta},\ c(x) = c.x^t$$

et pour un canal CTP (voir 5.3) c'est maximiser :

$$M_s(b_i, c) = \sum_{\beta \in \mathbb{F}_2^m} \log\left(\frac{p_{(b_\beta)_i}}{1 - p_{(b_\beta)_i}}\right) (-1)^{(b_\beta)_i}\ (-1)^{c_\beta},\ c(x) = c.x^t,\ \log\left(\frac{p_{(b_\beta)_i}}{1 - p_{(b_\beta)_i}}\right) \geq 0$$

La troisième étape de l'algorithme-10 SP, qui correspond au triangle dans le schéma de la figure-5.3, est un décodage dans le code C_ℓ. Dans cette étape nous cherchons à maximiser la fonctionnelle

$$T_i(a(x)) = \sum_{j=1}^{n} R_{\alpha_j} (-1)^{a(\alpha_j) + (b_{\alpha_j})_i}\ ,\ a(x) = a.x^t,\ R_{\alpha_j} \geq 0 \tag{5.6}$$

[1] MAP : en anglais "Maximum a posteriori decoder"

Notons que ne pas tenir compte de la sûreté R_{α_j} dans cette expression c'est faire de décodage dur (canal BSC), et le fait d'utiliser cette information supplémentaire c'est une approche pour faire de décodage souple (canal CTP). Nous allons donner l'interprétation de la valeur de sûreté dans l'algorithme de Sidel'nikov Pershakov.

Nous allons montrer que sous certaine condition maximiser 5.6 est équivalent à maximiser la fonctionnelle

$$\widetilde{M_i}(a(x)) = \sum_{\beta \in \mathbb{F}_2^m} \log p_{y_\beta} (-1)^{y_\beta} \ (-1)^{c_\beta} \ , \quad c = ev(a(x)) \ , \ y_\beta = (b_\beta)_i$$

pour une transmission sur un canal CTP. Ce n'est pas un décodage à maximum de vraisemblance (comparer cet expression avec celle de MLD : équation-5.3).

5.3.3 Signification de la sûreté

Soient $\mathcal{C}$ un code de longueur n, et z un vecteur binaire reçu après transmission d'un mot de code c. En décodant ce vecteur avec un algorithme de décodage nous obtenons un mot $\widetilde{c} \in \mathcal{C}$. Nous pouvons ensuite mesurer l'erreur dans le vecteur z par la distance $d(\widetilde{c}, z)$. Cette distance est liée à la probabilité que le résultat du décodage est bien le mot transmis. Soit $t = d(\widetilde{c}, z)$, pour une transmission sur un canal $BSC(p)$, nous avons :

$$\begin{aligned} \Pr\left[z|\widetilde{c}\right] &= p^t \ (1-p)^{n-t} \\ &= \left(\tfrac{p}{1-p}\right)^t \ (1-p)^n \end{aligned}$$

Donc pour $p < \frac{1}{2}$, nous avons :

$$\log \ \Pr\left[y|\widetilde{c}\right] = t \log\left(\frac{p}{1-p}\right) + \log\left(1-p\right)^n$$

Dans l'algorithme de décodage de $RM(2,m)$, pour un vecteur reçu y nous calculons la dérivée selon toutes les directions possibles $D_\alpha(y)$. En décodant le vecteur dérivé, nous obtenons la forme affine $b_\alpha.x^t + t_\alpha$ et la sûreté R_α selon cette direction :

$$\begin{aligned} R_\alpha &\triangleq n - 2\min\{\ d(D_\alpha(y), b_\alpha.x^t), d(D_\alpha(y), b_\alpha.x^t + 1)\ \} \\ &= n - 2d(D_\alpha(y), b_\alpha.x^t + t_\alpha) \end{aligned} \tag{5.7}$$

La sûreté atteint sa valeur maximum si les deux vecteurs $D_\alpha(y)$ et $ev(b_\alpha.x^t)$ sont identiques ou complémentaires, ce qui signifie que il n'y a pas d'erreur dans le vecteur dérivé. Elle est liée au poids de l'erreur dans la dérivée $D_\alpha(y)$.

Comme b_α est le résultat du décodage du vecteur $D_\alpha(y)$, nous pouvons donc écrire la sûreté R_α (equation-5.7), en fonction de la probabilité de recevoir le vecteur $D_\alpha(y)$ ayant transmis sur un canal BSC le vecteur $ev(b_\alpha.x^t + t_\alpha)$:

$$R_\alpha = a \log\left(\Pr\left[Y = D_\alpha(y) \mid X = b_\alpha.x^t + t_\alpha\right]\right) + b, \ a, b \in \mathbb{R}, \ a > 0$$

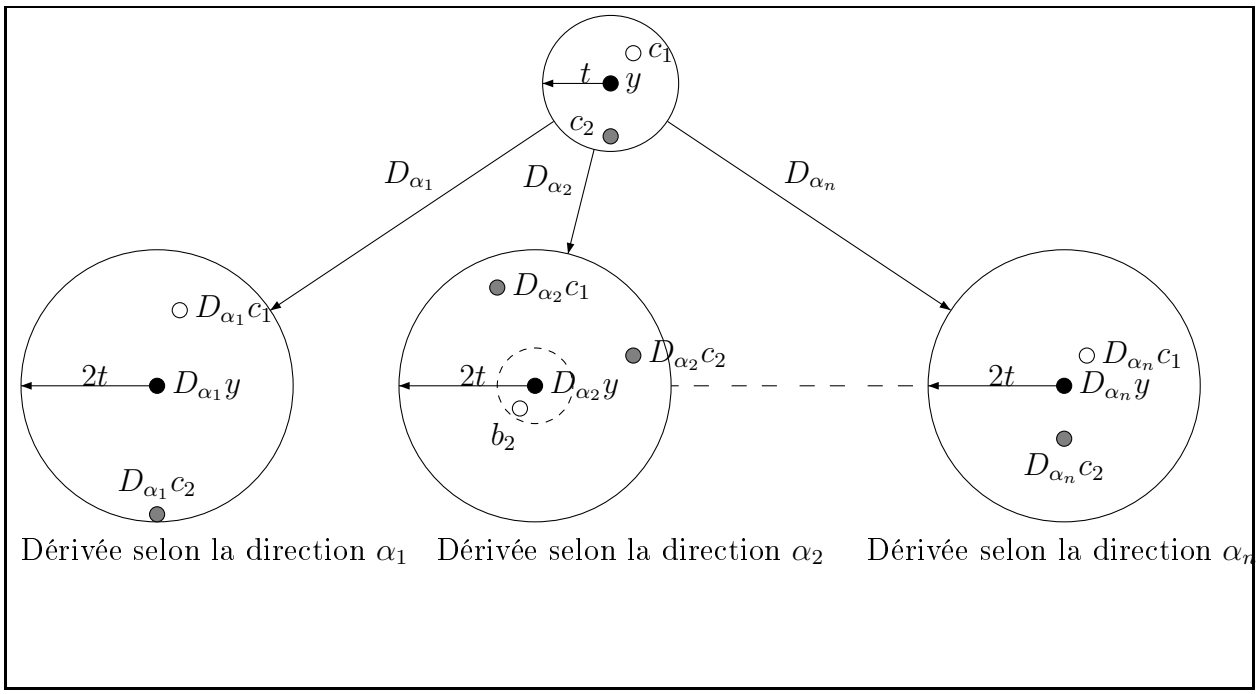


FIG. 5.2 – Dérivées selon toutes directions possibles

Nous déduisons que maximiser la fonctionnelle 5.6 revient à maximiser l'expression :

$$\widetilde{M}_i(c) = \sum_{\beta \in \mathbb{F}_2^m} \log\left(\Pr\left[D_\beta(y) \mid b_\beta.x^t + t_\beta\right]\right)(-1)^{y_\beta}\,(-1)^{c_\beta}\ ,\quad c = ev(a(x))\ ,\ y_\beta = (b_\beta)_i$$

5.3.4 Efficacité de l'utilisation de la sûreté

Dans l'algorithme de décodage SP du code $RM(2,m)$, nous calculons les dérivées selon toutes les directions possibles et nous appliquons le décodage MLD dans un code du premier ordre. Nous récupérons la partie linéaire $b_\alpha \in \mathbb{F}_2^m$. Pour certaines dérivées (voir figure-5.2) nous retrouvons les dérivées qui viennent des mots proches du y, mais pour d'autres nous retrouvons des mots qui ne sont pas dans le voisinage du y. Ayant le vecteurs $b = (b_{\alpha_1}, \ldots, b_{\alpha_n})$ qui est une version erronée du vecteur $(\alpha_1 B, \ldots, \alpha_n B)$, nous cherchons à décoder le vecteur b de manière optimale (retrouver les bits d'informations d'ordre-2 ou de manière équivalente la matrice B).

Prenons une dérivée selon une direction α. Si dans cette direction le poids de l'erreur est très important, cette dérivée peut être proche d'un mot qui n'est pas proche du mot transmis (voir figure-5.2, dérivée selon la direction α_2). Ainsi la sûreté peut être élevée pourtant ils ne provient pas d'un mot de $RM(2,m)$. Donc si le poids de l'erreur dépasse $\frac{n}{2}\left(1 - \sqrt[4]{4\ln(2)\frac{m}{n}}\right)$, le lemme-5.1.1 nous indique que pour la plupart des dérivées nous retrouvons des mots qui ne sont pas proches du mot transmis, ainsi la sûreté n'est pas suffisante pour caractériser les bonnes directions de dérivation.

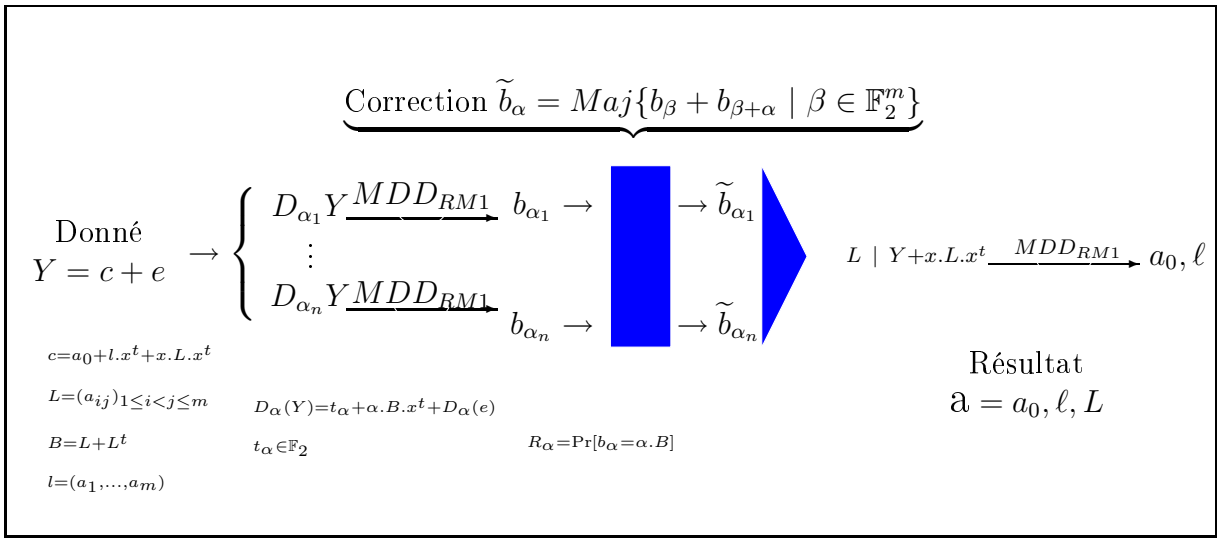


FIG. 5.3 – Schéma de l'algorithme SPM pour $RM(2,m)$

5.4 Algorithme de décodage SPM

Soit $B \in \mathcal{M}_{m\times m}(\mathbb{F}_2)$, l'encodage de B dans le code C_ℓ est le vecteur $E_{C_\ell}(B) = (\alpha_1 B, \ldots, \alpha_n B) \in (\mathbb{F}_2^m)^n$. Ainsi la coordonnée α est l'image de $\alpha \in F_2^m$ par l'action de la transformation linéaire définie par la matrice B, d'où nous avons la propriété suivante.

$$\alpha B + \beta B + (\alpha+\beta)B = 0, \forall \alpha, \beta \in \mathbb{F}_2^m. \tag{5.8}$$

La méthode de décodage utilisée dans l'algorithme-10 n'utilise aucune caractéristique du code C_ℓ. L'algorithme-4.2 paramétré $SP(.,s,h,m)$, utilise la linéarité vérifiée par les coordonnées du code C_ℓ. Mais la caractéristique la plus importante c'est que les coordonnées sont des éléments de $\mathbb{F}_2^m$ et l'erreur aussi. L'algorithme que nous allons exposer exploite la linéarité vérifiée par les coordonnées et le fait que ces coordonnées sont dans $\mathbb{F}_2^m$. Si une coordonnée est erronée alors avec une grande probabilité l'équation de linéarité ne sera plus vérifiée.

Notre modification consiste à ajouter une étape de correction par vote majoritaire à l'algorithme d'origine nous illustrons l'algorithme de décodage par le schéma 5.3.

La correction par vote utilise la propriété (5.8), $\forall \alpha \in F_2^m$ définissons $\widetilde{b}_\alpha$ par :

$$\widetilde{b}_\alpha = Maj_{\substack{\beta\in\mathbb{F}_2^m \\ \beta\neq 0, \beta\neq\alpha}} (b_{\alpha+\beta} + b_\beta) \tag{5.9}$$

Tel que Maj est la fonction qui choisit l'élément le plus fréquent. Pour montrer les motivations d'une telle correction, nous allons décrire un code qui correspond à notre situation, ensuite nous analysons l'étape de la correction.

Soit $B \in \mathcal{M}_{m\times m}$ nous l'encodons en $(\alpha_1 B, \ldots, \alpha_n B)$, $n = 2^m$, c'est un code qui contient le code C_ℓ défini par l'équation 5.4 mais on n'impose pas la symétrie sur la matrice B. Après transmission nous recevons le vecteur $(b_{\alpha_1}, \ldots, b_{\alpha_n})$, $b_\alpha \in \mathbb{F}_2^m$ où :

$$b_\alpha = \alpha B + e_\alpha$$

avec une erreur tirée aléatoirement dans $\mathbb{F}_2^m$.

Propriété 5.4.1 *Pour $B \in \mathcal{M}_{m\times m}$, $b = E_{C_\ell}(B) + e$. Soit $\#\{\alpha \mid \alpha B = b_\alpha\} = \delta\sqrt{n}$ alors, pour $\widetilde{b} \in \mathbb{F}_2^m{}_{/\{\alpha\}}$*

$$\frac{\Pr_\beta\left[b_\beta + b_{\alpha+\beta} = \alpha B\right]}{\Pr_\beta\left[b_\beta + b_{\alpha+\beta} = \widetilde{b}\right]} \geq \delta^2$$

▷ Preuve : Soit $B \in \mathcal{M}_{m\times m}$, et le vecteur $b = \{b_{\alpha_1}, \ldots, b_{\alpha_n}\}$, $b_\alpha = \alpha B + e_\alpha \in \mathbb{F}_2^m$ supposons que nous avons t éléments corrects dans le vecteur b :

$$s = \#\{\alpha \mid e_\alpha = 0\}$$

Nous allons calculer la probabilité P_0 que la somme de deux éléments ne contiendra pas d'erreur, pour un $\alpha \neq 0$ donnée nous avons :

$$\begin{aligned} P_0(\alpha) &= \Pr_{\beta\in\mathbb{F}_2^m}\left[b_\beta + b_{\beta+\alpha} = \alpha B\right] \\ &= \Pr\left[e_\beta + e_{\beta+\alpha} = 0\right] \\ &= \Pr\left[e_\beta = 0 \text{ et } e_{\alpha+\beta} = 0\right] + \Pr\left[e_\beta + e_{\beta+\alpha} = 0 \mid e_\beta \neq 0\right] \\ &\geq \frac{\binom{s}{2}}{\binom{n}{2}} \end{aligned} \tag{5.10}$$

Et la probabilité P_1 que la somme de deux éléments prend une valeur $\widetilde{b} \neq b_\alpha$, pour un α fixe nous avons :

$$\begin{aligned} P_1(\alpha) &= \Pr_{\beta\in\mathbb{F}_2^m}\left[b_\beta + b_{\beta+\alpha} = \widetilde{b}\right] \\ &= \Pr\left[e_\beta + e_{\beta+\alpha} = \widetilde{b} + \alpha B \neq 0\right] \\ &= \Pr\left[e_\beta = e_{\beta+\alpha} + \widetilde{b} + \alpha B \neq 0\right] \\ &\leq \frac{1}{n-1} \end{aligned}$$

D'où :

$$\frac{P_0(\alpha)}{P_1(\alpha)} \geq \frac{s(s-1)}{n} \approx \delta^2 \ , \quad \text{pour } s = \delta\sqrt{n}$$

◁

Cette propriété nous indique qu'ayant $\delta\sqrt{n}$ dérivées décodées correctement, l'étape de la correction permet de retrouver une dérivée selon une direction α avec une probabilité δ^2 plus que retrouver une autre valeur.

Lemme 5.4.1 *Soit $B \in \mathcal{M}_{m\times m}$ et $b = (b_{\alpha_1}, \ldots, b_{\alpha_n})$, $b_\alpha \in \mathbb{F}_2^m$, avec $b_\alpha = \alpha B + e_\alpha$ et $\mathbb{F}_2^m = \{\alpha_1, \ldots, \alpha_n\}$, où l'erreur e_α est tirée aléatoirement dans $\mathbb{F}_2^m$. Si le nombre d'element sans erreurs dans le vecteur b est $\delta\sqrt{n}$ avec $\delta^2 > \frac{m}{\log(m)}$, alors le vecteur $(\widetilde{b}_{\alpha_1}, \ldots, \widetilde{b}_{\alpha_n})$ avec les $\widetilde{b}_\alpha$ défini en 5.9 est sans erreur avec une grand probabilité.*

▷ Preuve : Nous allons étudier un événement similaire au vote d'un vecteur aléatoire : Soit X la variable aléatoire prenant ses valeurs dans $\mathbb{F}_2^m = \{0, 1, 2, \ldots, n-1\}$, nous allons l'exécuter n fois en notant les valeurs prises comme $X_1, X_2, \ldots, X_n$ et nous notons la fréquence d'apparition d'une valeur quelconque $i \in \mathbb{F}_2^m$ par

$$f_i = \#\{j | X_j = i\}$$

Il est démontré dans [6] que l'espérance de la plus grande fréquence est liée à n par la formule :

$$E(max_{\{i\in\mathbb{F}_2^m\}} f_i) = \frac{\log(n)}{\log\log(n)}$$

D'après 5.10 l'espérance de vote conduisant à une valeur correcte de la dérivée est δ^2. Nous déduisons alors que pour une fraction de bonnes dérivées équivalent à $\delta\sqrt{n}$ avec $\delta > \sqrt{\frac{m}{\log(m)}}$, l'étape de la correction par vote corrige avec une grande probabilité toutes les dérivées. ◁

Nous avons vu que le code $\mathcal{C}_\ell$ a une capacité de correction qui vaut $t_c = 3\frac{n}{8} - 1$, le lemme-5.4.1 nous indique que nous arrivons à décoder beaucoup plus loin que la capacité de correction de ce code à l'aide de l'étape de correction par vote majoritaire, puisque il nous suffit d'une fraction de $\sqrt{\frac{mn}{\log(m)}}$ correct coordonnées pour décoder un mot erroné de ce code.

Algorithme 13 $SPM(y, m)$.
Donnée : $Y = (Y_{\alpha_1}, \ldots, Y_{\alpha_n}) \in \{-1, 1\}^n$.
Résultat : $a = (a_0, a_1, \ldots, a_m, a_{12}, a_{13}, \ldots, a_{1m}, a_{23}, \ldots, a_{2m}, \ldots, a_{m-1.m})$

1. Pour tout $\alpha \in \mathbb{F}_2^m, \alpha \neq 0$, calculer

$$Z^\alpha = (Y_{\alpha_1+\alpha} Y_{\alpha_1}, \cdots\cdots, Y_{\alpha_n+\alpha} Y_{\alpha_n})$$

2. Pour tout $\alpha \in \mathbb{F}_2^m, \alpha \neq 0$, trouver la fonction linéaire

$$(t_\alpha, b_\alpha) = FHT(Z^\alpha)$$

   On note $b_\alpha = \Big(\ (b_\alpha)_1, \cdots, (b_\alpha)_m \Big) \ \in \mathbb{F}_2^m$, et $b_0 = 0$.
3. Pour tout $\alpha \in \mathbb{F}_2^m, \alpha \neq 0$:

$$b_\alpha = Maj_{\substack{\beta \in \mathbb{F}_2^m \\ \beta \neq 0, \beta \neq \alpha}} (b_{\alpha+\beta} + b_\beta)$$

4. Pour $i = 1 \cdots m$, calculer les fonctions linéaires $B_i(x) = B_i.x^t = \sum_{j=1}^m (B_i)_j x_j$, avec $(B_i)_i = 0$ qui maximisent la fonctionnelle

$$T_i(a(x)) = \sum_{j=1}^{n} (-1)^{a.{\alpha_j}^t + (b_{\alpha_j})_i}$$

   tel que a est une fonction linéaire de m variables $a(x) = a.x^t$
   Calculer $d_i = T_i(B_i(x))$, et noter que

$$B_i = ((B_i)_1, \cdots, (B_i)_{i-1}, 0, (B_i)_{i+1}, \cdots, (B_i)_m)$$

5. Trouver a_{ij}, avec

$$a_{ij} = \begin{cases} (B_i)_j & si \quad d_i \geq d_j \\ (B_j)_i & si \quad d_i < d_j \end{cases}$$

   ces valeurs définissent $\pi(x) = \sum_{i<j} a_{ij} x_i x_j$.
6. Chercher la fonction linéaire $l_0(x) = \sum_{i=1}^m a_i x_i + a_0$ qui maximise la fonctionnelle $(Y, \pi(x) + l(x))$.
   Le résultat de décodage est le vecteur

$$a = (a_0, a_1, \cdots, a_m, a_{12}, a_{13}, \cdots, a_{1m}, a_{23}, \cdots, a_{2m}, \cdots, a_{m-1.m})$$

   tel que : $f_a(x) = \pi(x) + l_0(x)$

Pour le code $RM(2,m)$ de longueur $n = 2^m$ l'algorithme SPM a une complexité comparable à SP, nous avons

$$|SPM| \leq n^2(\log_2(n)+1) + n\log_2(m)(\log_2(n)+1) = O(mn^2)$$

Lemme 5.4.2 *Les algorithmes SP, SPS, SPM réalisent le décodage à distance bornée, jusqu'à la capacité de correction* $\lfloor \frac{d-1}{2} \rfloor$ *du code* $RM(2,m)$

▷ Preuve : Le code $RM(2,m)$ est de longueur $n = 2^m$ et de distance minimale $d_2 = \frac{n}{4}$. Soit $y = c + e \in \mathbb{F}_2^n$ où $c = E(a_0, a_i, a_{ij}) \in RM(2,m)$ un mot reçu, avec un vecteur d'erreur de poids borné $wt(e) \leq \frac{n}{8} - 1$.
Pour chaque $\alpha \in \mathbb{F}_2^m$ la dérivée $D_\alpha y = D_\alpha c + D_\alpha e$ contient au plus $\frac{n}{4} - 2$ erreurs, et comme $D_\alpha c \in RM(1,m)$ où la distance minimale du code d'ordre-1 vaut $d_1 = \frac{n}{2}$ donc l'algorithme FHT corrige certainement $\frac{n}{4} - 1$ erreurs.
Ainsi les algorithmes de décodage de type SP retrouvent correctement tous les bits d'informations d'ordre deux a_{ij}.
L'étape de correction dans l'algorithme SPM, et le choix de plusieurs mots dans le décodages des dérivées dans l'algorithme paramétré $SP(.s,h,m)$ n'affecte pas les éléments b_α puisque dans ces cas nous avons $b_\alpha = \alpha B$, et les votes retournent le même vecteur b_α pour SPM. Dans $SP(.,s,h,m)$ les mots les plus proches sont les vecteurs αB avec la plus grande valeur de sûreté. Suite aux étapes du raffinement, ils gardent aussi la même valeur de sûreté qui sera la plus élevée.
Pour retrouver les bits d'informations d'ordre-1, nous décodons avec l'algorithme FHT le mot $y + \sum_{1 \leq i < j \leq m} a_{ij} x_i x_j$ qui contient $wt(e) < \frac{n}{4}$ erreurs. ◁

5.5 Analyse du décodage de $RM(2,m)$

Soit w la variable aléatoire représentant le poids d'erreur dans la dérivée selon une direction α du vecteur reçu y. Nous avons vu dans 4.4 théorème-4.4.1 que cette variable suit la loi binomiale avec $\mu_w = \frac{n}{2}(1-\epsilon^2)$ et $\sigma_w^2 = \frac{n}{2}(1-\epsilon^4)$. Sachant que la borne du décodage de MDD_{RM_1} est $ML_1 = \frac{n}{2}\left(1 - \sqrt{4\ln(2)\frac{m}{n}}\right)$ (définition-3.2.2).

Dans l'étude de la borne de décodage asymptotique de l'algorithme SP et de décodage récursif, nous avons démontré que si $\epsilon = \sqrt[4]{l\,\frac{m}{n}}$, $l > 4\ln(2)$ ces algorithmes corrigent avec une grande probabilité les mots transmis sur le canal $BSC(p = \frac{1}{2}(1-\epsilon))$, ceci est illustré par la figure-5.4. Nous constatons que la probabilité de mal décoder une dérivée est proportionnelle à $\left(\frac{\sigma}{d}\right)^2$, où d est la distance entre l'espérance de w et la borne ML_1. Pour $\epsilon = \sqrt[4]{l\,\frac{m}{n}}$, $l > 4\ln(2)$, nous avons $\sigma^2 \approx \frac{n}{2}$ et $d^2 \approx c\,nm$ avec c une constante. Donc la probabilité de mal décoder selon une direction de dérivation tends vers zéro quand m tends vers l'infini.

Nous voulons étudier la borne de décodage asymptotique pour l'algorithme-13, donc d'après le lemme-5.4.1 il nous suffit de trouver la probabilité d'erreur du canal

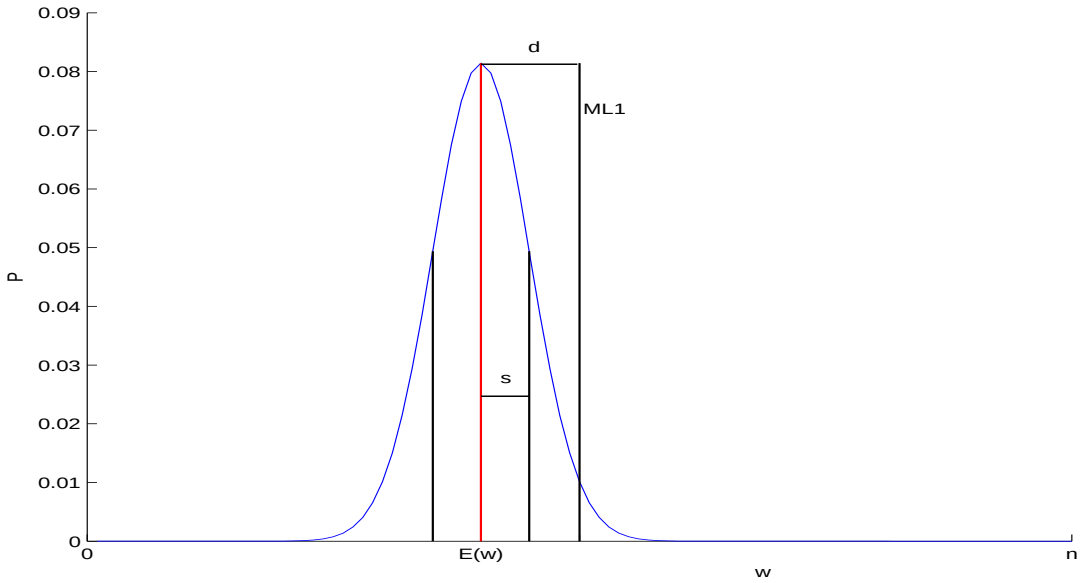


FIG. 5.4 – Nous avons $ML_1 = \frac{n}{2}(1 - 2\sqrt{\ln(2)\frac{m}{n}})$, Pour $\epsilon = \sqrt[4]{l\,\frac{m}{n}}$, $l > 4\ln(2)$, soient $s = \sigma_w$ et $d = ML_1 - E(w) > s$, nous avons $\Pr[w > ML_1] \leq \Pr[|w - E(w)| > d] \leq \left(\frac{\sigma}{d}\right)^2 = \frac{1}{c\ m}$

$BSC(p)$, $p = \frac{1}{2}(1-\epsilon)$ pour lequel l'algorithme de décodage MDD_{RM_1} du code $RM(1,m)$ (voir schema-5.3) corrige au moins $\delta\sqrt{n}$, $\delta = \sqrt{\frac{m}{\log(m)}}$. nous illustrons sur le schéma-5.5 la probabilité à calculer en utilisant l'approximation de la loi binomiale par une loi gaussienne décrit en A.3.1.

$$\Pr[w < ML_1] \geq \frac{\delta}{\sqrt{n}}$$

Pour $\epsilon\sqrt{l\frac{m}{n}}$, $l < 4\ln(2)$, donc $\mu_w = E(w) > ML_1$, En utilisant l'inégalité de Chebyshev nous avons :

$$\begin{aligned}
\Pr[w < ML_1] &\leq \Pr[|w - \mu_w| \geq |\mu_w - ML_1|] \\
&\leq \frac{\sigma^2}{|\mu_w - ML_1|^2} \\
&\leq \frac{n}{2|\mu_w - ML_1|^2} \\
&\leq \frac{1}{cm}, \quad c = \frac{(\sqrt{4\ln(2)} - \sqrt{l})^2}{2}
\end{aligned}$$

La fraction de $\frac{1}{cm}$ est largement suffisante, mais c'est malheureusement une borne supérieure, qui ne nous sera pas utile. Puisque nous ne savons toujours pas, pour quelle valeur de ϵ, cette probabilité est supérieure à $\delta/\sqrt{n}$. Nous ne savons pas actuellement borner inférieurement de façon stricte la probabilité qu'une variable aléatoire suivant la loi gaussienne ait des petites valeurs. En utilisant l'inégalité de Chernoff nous obtenons une borne supérieure aussi qui ne peut pas nous aider à déterminer la borne de décodage asymptotique de notre algorithme.
Soit $p = \frac{1}{2}(1 - \epsilon)$ la probabilité d'erreur sur le canal BSC, et $\epsilon = \sqrt[4]{l\frac{m}{n}}$, $l < 4\ln(2)$, nous notons w la variable aléatoire représentant le poids d'erreur dans la dérivée selon une direction α du vecteur reçu y. Elle suit la loi binomiale avec $\mu_w = \frac{n}{2}(1 - \epsilon^2)$ et

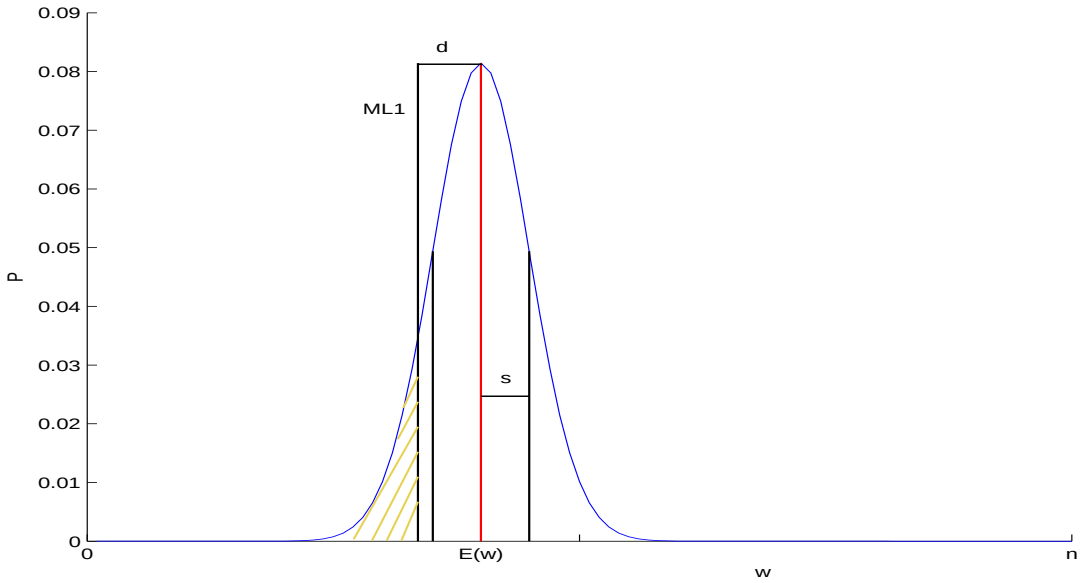


FIG. 5.5 – Nous cherchons ϵ, qui vérifie que la zone hachée soit plus grand que $\sqrt{\frac{m}{\log(m)}}\, n^{-\frac{1}{2}}$

$\sigma_w^2 = \frac{n}{2}(1-\epsilon^4)$. Nous voulons calculer la probabilité de bien décoder ($\Pr[w < ML_1]$) dans une direction de dérivation. En utilisant l'inégalité de Chernoff (A.3.3), nous écrivons :

$$\begin{array}{rcl} \Pr[w < ML_1] & \leq & \Pr[|w-\mu_w| \geq |\mu_w - ML_1|] \\ & \leq & \frac{1}{2^{\frac{|\mu_w - ML_1|^2}{3n}}} \\ & \leq & \frac{1}{2^{cm}}, \quad c = \frac{(\sqrt{4\ln(2)}-\sqrt{l})^2}{12} \\ & \leq & \frac{1}{\sqrt[6]{n}}, \quad \text{Pour } l = 0 \end{array}$$

De la même manière, nous trouvons que la borne inférieur donnée par l'inégalité de Markov ne nous donne pas de précision sur la borne de décodage asymptotique.

Chapitre 6

Simulation et Résultats expérimentaux

Ce chapitre est dédié aux résultats numériques que nous avons obtenus à l'aide des algorithmes que nous avons présentés dans les différents chapitres de ce manuscrit, il concerne donc le décodage du code $RM(2, m)$. Pour simuler le canal de transmission BSC nous avons besoin d'un générateur d'aléa, nous avons utiliser le générateur d'aléa *HAVEGE* (HArdware Volatile Entropy Gathering and Expansion)[1] mis au point par André Seznec et Nicolas Sendrier [48].

Pour mesurer numériquement la performance des algorithmes, nous tirons aléatoirement un grand nombre de mots du code, pour chaque mot nous tirons aléatoirement une erreur de poids t. Le tirage d'un vecteur de longueur n et de poids t se fait de la façon suivante :

- Pour générer un vecteur d'erreur e de poids t :
- Initialiser le vecteur $e = (e_1, \ldots, e_n), \forall i,\ e_i = 0$, nombre d'erreur $p_e = 0$.
- Tant que $p_e < t$ faire :
  - Tirer un nombre j aléatoirement avec $Havege$ entre 1 et n.
  - Tester si $e_j = 0$ faire :
    $e_j = 1$
    Incrementer(p_e).
- le vecteur e est de poids t

Ainsi $\forall i = 1 \ldots n,\ \ \Pr[e_i = 1] = \frac{t}{n}$.
Soit c un mot du code, et e le vecteur d'erreur tiré aléatoirement de la manière décrite précédemment. Le vecteur $y = c + e$ est décodé avec un algorithme de décodage D, on note $\widetilde{c} = D(y)$ le mot du code renvoyé par l'algorithme. Nous mesurons deux événements, le premier est le pourcentage des mots correctement décodés, c'est le cas où l'algorithme

[1] C'est un générateur de nombres aléatoires très performant, il exploite comme source d'incertitude les très nombreuses modifications de l'état interne de l'ordinateur. Téléchargeable en accord avec la licence *Lesser GNU General Public License* sur http ://www.irisa.fr/caps/projects/hipsor/

retrouve le mot $D(y) = c$. Le deuxième événement est quand D retrouve un mot plus proche du vecteur y que le mot du code c ($d(\widetilde{c}, y) \leq d(c, y) = wt(e)$). C'est un événement très important aussi puisqu'il nous indique le poids de l'erreur pour lequel un algorithme MDD n'arrive pas à décoder avec une grande probabilité un vecteur erroné. Notez que nous ne savons toujours pas décoder de façon optimale le code $RM(2, m)$.
Nous utilisons fréquemment dans ce chapitre les notations suivantes :

c	mot de code de $RM(2, m)$.
$\widetilde{c}$	le résultat d'un algorithme de décodage.
t	poids du vecteur d'erreur.
SP	L'algorithme de décodage de Sidel'nikov et Pershakov (Algorithme 10)
SPP	L'algorithme de décodage de Sidel'nikov et Pershakov paramétré $s = h = 3$ (Algorithme-4.5.2).
$Dumer$	L'algorithme récursif de Dumer $\Psi(., 2, m)$ algorithme-8
SPM	L'algorithme de décodage proposé (Algorithme-13)

Nous avons programmé tous les algorithmes de décodages étudiés en langage C++, et comme les algorithmes $FHT, Dumer$, SP, SPP, SPM ont besoin de manipuler des nombres réels, en fait tous les algorithmes appellent l'algorithme FHT qui manipule des réels et peut être exécuté pour des entiers vu que nous simulons un canal BSC. et pour un code $RM(2, m)$ la longueur du code est $n = 2^n$, dans le cas de 14 variables $n = 16384$, et pour un entier sur deux octets (FHT les composant du vecteur sont dans l'intervalle $[-n, n]$, la matrice génératrice de ce code contient $nk = 3Moctets$. Le temps du calcul pour une simulation de l'algorithme $SPP = 24$ minutes sur une machine AMD Athlon 1,8GHz et $SPP = 21$ minutes AMD Opteron 2GHz.

Nous rappelons dans le tableau-6 les bornes de décodage asymptotique des différents algorithmes.

Algorithme	Complexité	borne de décodage asymptotique
Vote Majoritaire	$O(m^2\ n)$	$\frac{n}{2}(1 - \sqrt[8]{4\ \frac{m}{n}})$
Dumer	$O(m\ n)$	$\frac{n}{2}(1 - \sqrt[4]{16\frac{m}{n}})$
SP	$O(m\ n^2)$	$\frac{n}{2}\left(1 - \sqrt[4]{4\ln(2)\frac{m}{n}}\right)$
MLD	$O(n^m)$	$\frac{n}{2}\left(1 - \sqrt{6\ ln(2)\frac{m^2}{n}}\right)$

TAB. 6.1 – bornes de décodage asymptotique pour le code $RM(2, m)$

Nous tenons à rappeler que les performances pour les moyennes longueurs nous intéressent beaucoup plus que les performances en asymptotique. Et nous allons voir dans la suite que les deux algorithmes SP et $Dumer$ ont la même borne asymptotique tandis que SP corrige plus loin que $Dumer$.

6.1 Les codes de petites longueurs

Nous allons comparer les algorithmes de décodage quand le nombre de variables m varie de 7 à 10. Dans le tableau 6.1, nous avons extrait quelques résultats numériques pour des

m	$n = 2^m$	$\frac{d-1}{2}$	t	$c = \widetilde{c}$			
				Dumer	SP	SPP	SPM
7	128	15	25	64	95	98	98
7	128	15	28	30	54	81	82
7	128	15	30	12	10	43	50
8	256	31	47	99	100	100	100
8	256	31	50	96	100	100	100
8	256	31	56	84	99	100	100
8	256	31	60	60	99	100	100
8	256	31	64	37	92	99	99
8	256	31	72	6	3	36	56
9	512	63	115	100	100	100	100
9	512	63	116	97	100	100	100
9	512	63	142	48	100	100	100
9	512	63	145	40	99	100	100
9	512	63	155	15	59	99	100
9	512	63	156	13	52	96	99
9	512	63	166	1	0	9	64
10	1024	127	257	100	100	100	100
10	1024	127	258	99	100	100	100
10	1024	127	310	57	100	100	100
10	1024	127	335	18	97	100	100
10	1024	127	347	6	45	99	100
10	1024	127	352	3	11	96	100
10	1024	127	359	1	0	51	99
10	1024	127	370	0	0	0	54

TAB. 6.2 – Pourcentage de mots bien décodés par différents algorithmes pour le code $RM(2, m)$, $7 \leq m \leq 10$, pour 100000 essais

codes $RM(2, m)$ de petites longueurs, nous pouvons remarquer que l'algorithme proposé SPM corrige plus loin que les autres algorithmes pour ces petites longueurs et que quand le nombre de variables m augmente l'algorithme SPM corrige exponentiellement plus loin, en fonction du nombre de variables, que les autres algorithmes cette remarque reste valide pour les longueurs moyennes et il sera présenté graphiquement. Pour les petites longueurs notre algorithme arrive à retrouver avec une probabilité $> \frac{1}{2}$ un mot $\widetilde{c}$ plus proche que le mot transmis c ce qui n'est plus le cas pour des longueurs moyennes. Nous n'avons pas encore d'explications sur cette remarque.

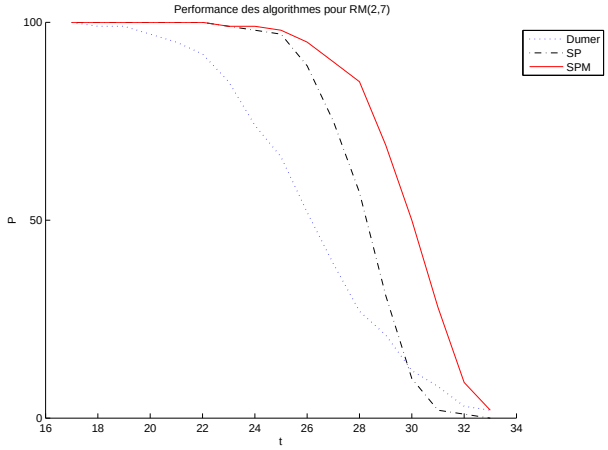


FIG. 6.1 – Performance pour $RM(2,7)$

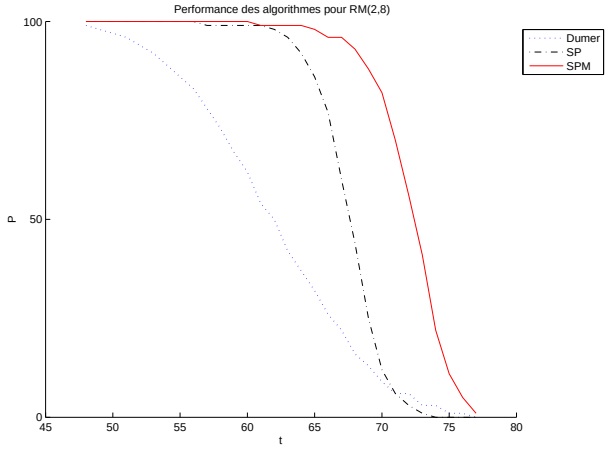


FIG. 6.2 – Performance pour $RM(2,8)$

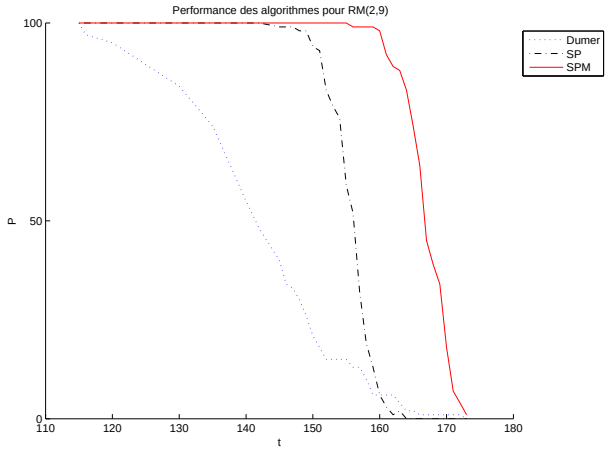


FIG. 6.3 – Performance pour $RM(2,9)$

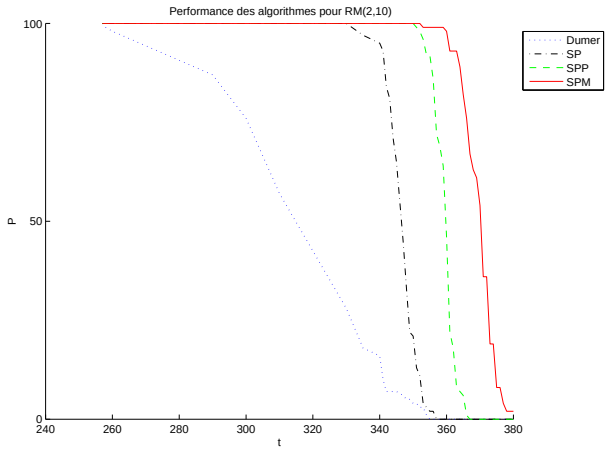


FIG. 6.4 – Performance pour $RM(2, 10)$

6.2 Les codes de longueurs moyennes

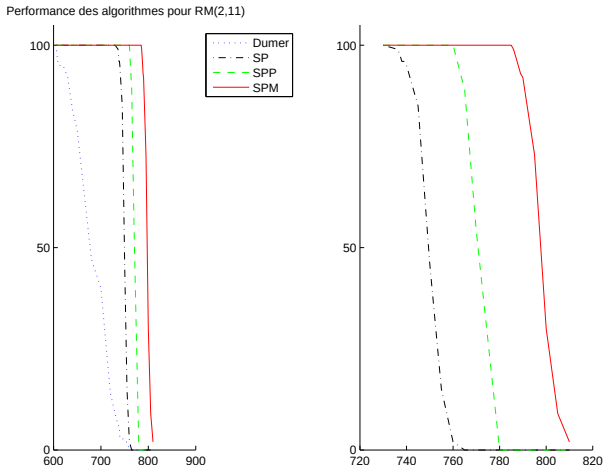


FIG. 6.5 – Performance pour $RM(2, 11)$

Soit D un algorithme de décodage des codes $RM(2, m)$, sa borne de décodage asymptotique est notée $b_D = \frac{n}{2}(1 - \epsilon_D)$ et c'est le poids de l'erreur que l'algorithme décode avec une grande probabilité. Si $\epsilon_D = \left(\ell_D \frac{m}{n}\right)^{a_D}$, donc

$$\begin{aligned} -\log \epsilon_D &= -\log\left(1 - 2\tfrac{b}{n}\right) \\ &= a_D(m - \log m) - a_D \log \ell_D \\ &= a_D(m - \log m) - b_D \end{aligned}$$

Pour les algorithmes étudiés nous avons considéré la borne de décodage asymptotique pour chaque nombre de variable $m = 7, \ldots, 14$, comme le poids de l'erreur que l'algorithme décode avec une grande probabilité (100% des tests executés), et nous avons calculé $-\log\left(1 - 2\frac{b}{n}\right)$, nous avons pris le polynôme de premier degré qui approche le plus au sens moindre carré. La figure-6.9 montre les résultats obtenus. Nous constatons que

m	$n = 2^m$	$\frac{d-1}{2}$	t	$c = \widetilde{c}$			
				Dumer	SP	SPP	SPM
11	2048	255	605	100	100	100	100
-	-	-	630	92	100	100	100
-	-	-	736	6	99	100	100
-	-	-	760	0	2	100	100
-	-	-	765	0	0	89	100
-	-	-	770	0	0	54	100
-	-	-	780	0	0	0	100
-	-	-	786	0	0	0	99
-	-	-	790	0	0	0	92
12	4096	511	1300	100	100	100	100
-	-	-	1310	99	100	100	100
-	-	-	1470	53	100	100	100
-	-	-	1550	7	100	100	100
-	-	-	1580	4	91	100	100
-	-	-	1600	0	19	100	100
-	-	-	1605	0	4	100	100
-	-	-	1610	0	3	97	100
-	-	-	1620	0	0	58	100
-	-	-	1640	0	0	0	100
-	-	-	1655	0	0	0	99
-	-	-	1670	0	0	0	55
13	8192	1023	2810	100	100	100	100
-	-	-	3000	84	100	100	100
-	-	-	3300	3	99	100	100
-	-	-	3360	0	4	90	100
-	-	-	3400	0	0	0	100
-	-	-	3435	0	0	0	99
-	-	-	3445	0	0	0	70
-	-	-	3450	0	0	0	40
14	16384	2047	6010	99	100	100	100
-	-	-	6300	80	100	100	100
-	-	-	6700	10	100	100	100
-	-	-	6870	0	95	100	100
-	-	-	6950	0	0	69	100
-	-	-	7000	0	0	0	100
-	-	-	7070	0	0	0	100
-	-	-	7075	0	0	0	90
-	-	-	7100	0	0	0	40

TAB. 6.3 – Pourcentage de mots bien décodés par différents algorithmes pour le code $RM(2, m)$, $11 \leq m \leq 14$, pour 10000 essais

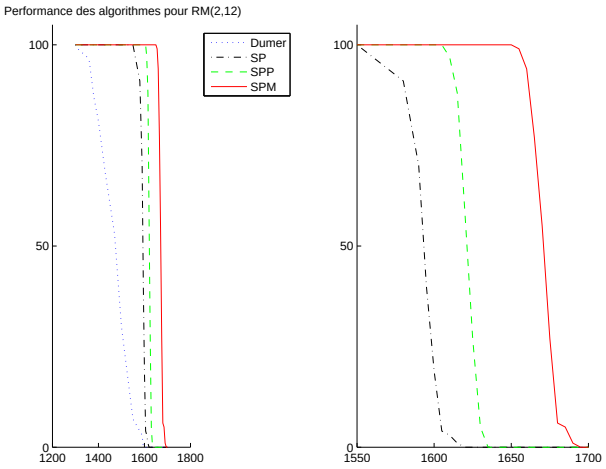


FIG. 6.6 – Performance pour $RM(2, 12)$

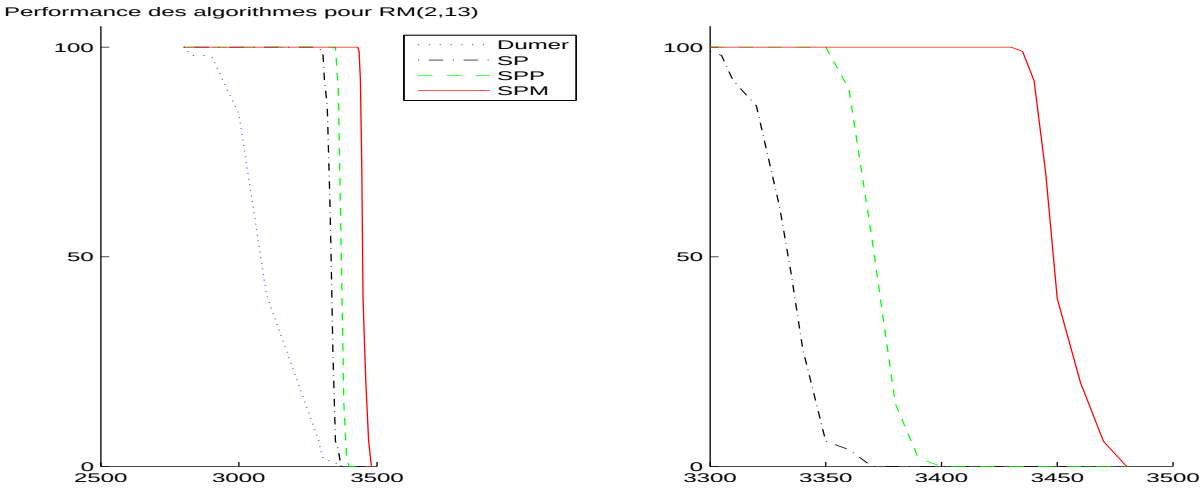


FIG. 6.7 – Performance pour $RM(2, 13)$

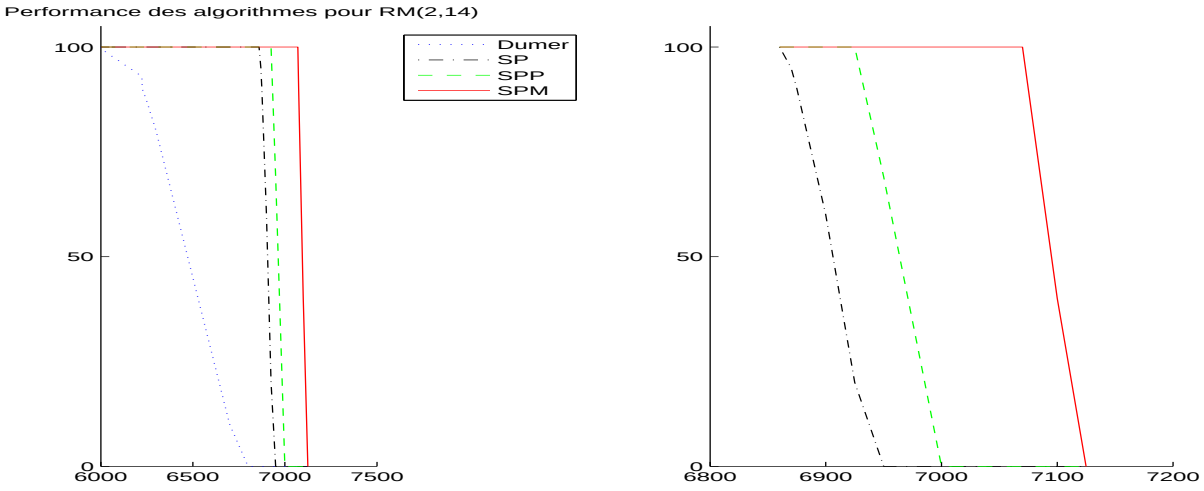


FIG. 6.8 – Performance pour $RM(2, 14)$

l'algorithme de *Dumer* a bien une pente qui vaut $0.238 \approx \frac{1}{4}$ qui correspond bien à sa borne de décodage asymptotique donnée dans le tableau 6. Pour l'algorithme SP et SPP la pente vaut 0.33 qui est plus élevée que la borne asymptotique, pour notre algorithme la pente vaut 0.37 donc ce qui montre expérimentalement que nous arrivons à décoder un bruit plus élevé de l'ordre $\epsilon = \left(\ell \frac{m}{n}\right)^a,\ a = 0.37 > \frac{1}{3}$

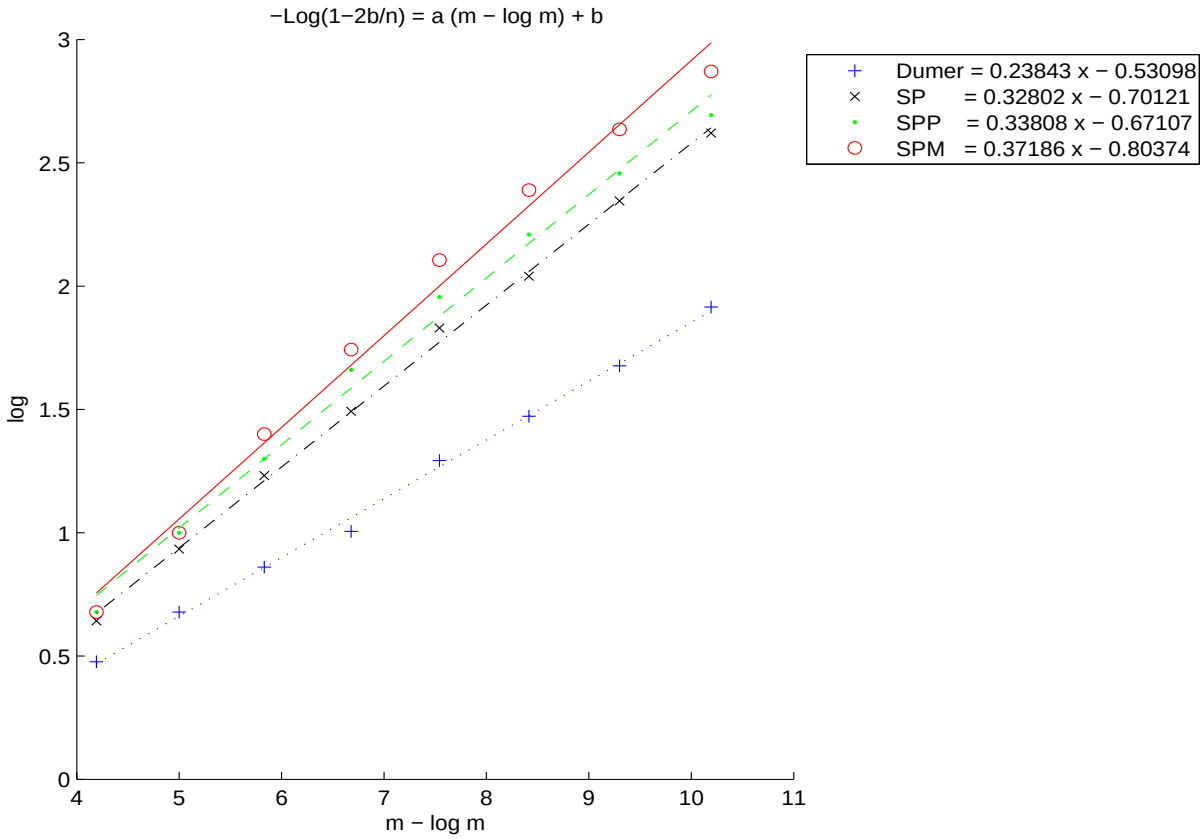


FIG. 6.9 – Comparaison, et borne de décodage asymptotique

Nous avons prouvé que la borne de décodage asymptotique stricte de l'algorithme *Dumer* est majorée par $\frac{n}{2}\left(1 - \sqrt[4]{4\ln(2)\frac{m}{n}}\right)$ (voir 5.1.1), ce qui est vérifié expérimentalement, sur la figure-6.9. Les algorithme SP et SPP peuvent décoder au delà de la borne $SP2$, et notre algorithme décode exponentiellement plus loin que tous les autres algorithmes. La borne de décodage asymptotique stricte d'un algorithme MLD (voir 3.2.2), est

$$ML_2 = \frac{n}{2}\left(1 - \sqrt{12\ ln(2)\frac{m^2}{n}}\right)$$

et sur la figure ce sera une droite d'une pente $\frac{1}{2}$.

Conclusion

Les travaux de cette thèse portent sur le décodage des codes de Reed-Muller. Nous avons décrit les codes de Reed-Muller, ainsi que leur groupe d'automorphismes, en utilisant la description de ces codes à l'aide des fonctions booléennes. La notion de borne de décodage asymptotique ainsi que la borne de décodage asymptotique stricte ont été définis. Une majoration de la borne de décodage asymptotique stricte pour les algorithmes de décodage récursif est donnée (lemme-5.1.1). Cette borne de décodage ne peut pas être franchie avec les méthodes utilisées jusqu'à présent (basées sur la dérivée).

Les travaux que j'ai effectués ont amélioré de manière significative l'efficacité de l'algorithme de Sidel'nikov-Pershakov en le simplifiant et en ajoutant une étape de correction par vote majoritaire qui permet d'accroître de manière importante les performances du décodage à complexité constante [33]. L'algorithme proposé décode plus loin que tous les autres algorithmes [45]. Si pour le moment la borne de décodage stricte de notre algorithme n'est pas déterminée, nous donnons des arguments qui nous laissent croire en l'efficacité de notre modification.

Le décodage des codes de Reed-Solomon, non binaires définis à l'aide des polynômes univariés a connu une avancé remarquable après la mise au point, par Madhu SUDAN, d'un algorithme exploitant les propriétés algébriques de ces codes. Mais cet algorithme n'est pas applicable aux codes binaires. Tous les anciens algorithmes de décodage sont basés sur la reconstruction de polynôme.

Les codes de Reed-Muller ont une description similaire des codes de Reed-Solomon, le décodage par reconstruction de polynôme des codes de Reed-Solomon développé par GOLDREICH, RUBINFELD et SUDAN [25] a donné suite à plusieurs algorithmes de décodage en liste. GOLDREICH et LEVIN [24] ont proposé un algorithme applicable aux codes de Reed-Muller binaires. Ce dernier a été bien étudié et amélioré dans les travaux récents de TAVERNIER et KABATIANSKIY [28, 27].

L'étude effectuée dans cette thèse montre que tous les algorithmes de décodage connus pour les codes RM sont basés sur la dérivée discrète. L'algorithme proposé exploite le fait

que l'ensemble des dérivées d'un mot de $RM(2, m)$ forme un groupe de codes non binaires, et décrit un moyen de décoder ce code en exploitant cette remarque. Nous avons montré que la dérivation fait augmenter le bruit d'un facteur carré, cela impose une limite difficile à franchir avec les algorithmes existants. Les résultats expérimentaux ont montré que notre algorithme arrive à décoder plus loin que cette limite, il serait intéressant d'analyser le comportement asymptotique de notre algorithme afin de voir s'il dépasse en efficacité, ne serait-ce que légèrement, les algorithmes par décodage récursif, ce qui est suggéré par les courbes tracées pour des décodages en longueur moyenne.

Perspectives générales

Il serait intéressant d'utiliser l'algorithme d'encodage récursif (algorithme-7) avec un coût $n \min(r, m - r)$ et qui ne nécessite pas le stockage de la matrice génératrice du code en mémoire. Nous pouvons ainsi faire les tests pour un nombre de variables $m > 14$, et nous pouvons mesurer les performance de notre algorithme, mais le temps de calcul croît exponentiellement en le nombre de variables.

Pour les codes d'ordre trois, nous pouvons avoir de manière similaire un algorithme de type SPS simple en choisissant le nombre exacte de dérivées nécessaires, et un algorithme de type SPM en ajoutant une étape de correction par vote majoritaire. Du point de vue théorique cela ne vas pas améliorer la borne de décodage asymptotique de ces codes, puisque nous n'avons pas pu le déterminer pour les codes d'ordre deux. Du point de vue pratique nous obtenons deux algorithmes qui décodent plus loin que les algorithmes récursifs, et avec une complexité de plus en plus élevée si nous voulons décoder plus loin.

Annexe A

Rappels de probabilités

Nous allons juste ici faire les rappels de probabilités nécessaires à la bonne compréhension du document. Le lecteur intéressé pourra se reporter à [20] pour des informations additionnelles sur le sujet. La partie fondamentale est le théorème central limite. Les éléments présentés permettent de justifier l'approximation d'une distribution observée par une loi théorique. De plus ces éléments permettent de donner l'ordre des erreurs corrigibles dans le calcul de la capacité de correction d'un algorithme de décodage.

A.1 Loi de probabilité

Une variable aléatoire est une fonction définie sur l'ensemble des résultats possibles d'une expérience aléatoire, telle qu'il soit possible de déterminer la probabilité pour qu'elle prenne une valeur donnée ou qu'elle prenne une valeur dans un intervalle donné

la variable aléatoire la plus simple c'est la variable de Bernoulli. Celle-ci peut prendre deux états, codés 1 et 0, avec les probabilités p et $1-p$. Une interprétation simple concerne un jeu de dé dans lequel on gagnerait un euro en tirant le six ($p = 1/6$). Sur une séquence de parties, la moyenne des gains tend vers p lorsque le nombre de parties tend vers l'infini.

Une loi de probabilité, ou distribution de probabilité, a commencé par décrire les répartitions typiques des fréquences d'apparition des résultats d'un phénomène aléatoire. On associe naturellement une loi de probabilité à une variable aléatoire pour décrire la répartition des valeurs qu'elle peut prendre.

Parmi l'ensemble des lois de probabilités possibles, on distingue un certain nombre de familles usuelles qui correspondent à des phénomènes aléatoires simples : lancer de dés, jeu de pile ou face, erreurs de mesures, etc. En théorie des probabilités, une loi de probabilité est une mesure positive sur un espace mesuré, de masse finie égale à 1. Lorsque le support de cette mesure est un ensemble fini ou dénombrable, par exemple $\mathbb{N}$, on parle de loi

discrète, tandis que lorsque la mesure est absolument continue, on parle de loi continue.

Définition A.1.1 *On appelle densité de probabilité d'une variable aléatoire X réelle continue une fonction f*
- *positive ou nulle sur $\mathbb{R}$*
- *intégrable sur $\mathbb{R}$*
- *vérifiant $\int_{\mathbb{R}} f(t)\,dt = 1$*

On appelle distribution de probabilité de X la fonction :

$$F(x) = \Pr[X < x] = \int_{-\infty}^{x} f(t)\,dt$$

La probabilité $\Pr[a < X < b]$ se calcule alors par la relation suivante :

$$\Pr[a < X < b] = \int_{a}^{b} f(t)\,dt = F(b) - F(a)$$

En traçant la représentation graphique de la densité de probabilité, la probabilité $\Pr[a < X < b]$ se lit comme l'aire sous la courbe sur l'intervalle $]a; b[$.

Définition A.1.2 *Espérance et Variance*
Soit X une variable aléatoire nous nous intéressons à deux quantités, l'espérance et la variance de cette variable. L'espérance, notée $E(X)$, nous indique la moyenne des valeurs prises par cette variable en le répétant un grand nombre de fois. Habituellement on note $\mu = E(X)$.
La variance, notée $Var(X)$, est une mesure servant à caractériser la dispersion d'un échantillon, définie par le carré de l'écart type. Habituellement on note $\sigma^2 = Var(X)$, ainsi σ est l'écart type.

Pour une variable aléatoire suivant une loi discrète, prenant pour valeurs $x_1, x_2, \ldots$ avec les probabilités respectives $p(x_1), p(x_2), \ldots$, l'espérance et la variance sont calculées de la façon suivante :

$$E(X) = \sum_k x_k p(x_k)$$

$$Var(X) = (\sigma_X)^2 = \sum_k p_k \left(x_k - E(X)\right)^2 = \sum_k (p_k x_k^2) - (E(X))^2$$

Dans le cas continu, la variable aléatoire X possède une densité de probabilité f, Nous avons :

$$E(X) = \int_{\mathbb{R}} x\, f(x)\,dx$$

$$Var(X) = \int_{\mathbb{R}} x^2\, f(x)\,dx - \left[\int_{\mathbb{R}} x\, f(x)\,dx\right]^2$$

Notez que nous avons toujours :

$$Var(X) = E(X^2) - (E(X))^2$$

Ces deux valeurs (μ, σ) sont caractéristique pour une loi de probabilité.

Définition A.1.3 *Soit X et Y deux variables aléatoires, nous disons qu'elles sont indépendantes si :*

$$\Pr[Y \mid X] = \Pr[Y]$$

La probabilité conditionnelle $\Pr[Y \mid X]$ *est définie par*

$$\frac{\Pr[X, Y]}{\Pr[X]}$$

A.2 Quelques lois de probabilité

On réalise une épreuve aléatoire dont la probabilité d'un succès est p. Si X est la variable aléatoire qui vaut 1 s'il y a succès, 0 sinon, alors X suit une **loi de Bernoulli** de paramètre p, notée $Bernoulli(p)$.

Note 11 $Bernoulli(p)$
- $\mu = p$
- $\sigma = \sqrt{p(1-p)}$

L'erreur de transmission sur un Canal binaire symétrique (BSC) suit la loi de Bernoulli avec un paramètre p qui dépend des caractéristique du canal.

La loi binomiale apparaît naturellement lorsqu'on considère la somme de n variables aléatoires indépendantes X_i de même loi de Bernoulli de paramètre p

$$X = X_1 + X_2 + \cdots + X_n$$

Alors X est une variable aléatoire de loi Binomiale $B(n, p)$, et X prend ses valeurs dans l'ensemble $0, 1, \cdots, n$ avec probabilité[1]

$$\Pr[X = k] = \binom{n}{k} p^k (1-p)^{n-k}$$

Note 12 $B(n, p)$

[1] Le coefficient binomial $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

- $\mu = np$
- $\sigma = \sqrt{np(1-p)}$

Exemple A.2.1 *Le tirage de boules avec remise.*

Soit une urne contenant b boules blanches et r boules rouges, on note $p = \frac{b}{b+r}$. *On effectue n tirages avec remise, alors la probabilité de l'évènement* E_k *(l'échantillon de n boules obtenu contient exactement k boules blanches) est.*

$$\Pr[E_k] = \frac{n!}{k!(n-k)!} p^k (1-p)^{n-k}$$

Espérance et variance : L'espérance d'une loi binomiale $B(n,p)$ est np et sa variance $np(1-p)$

Définition A.2.1 *La densité normale*

La fonction $g : \mathbb{R} \to \mathbb{R}^+$ *définie par :*

$$g(t) = \frac{1}{\sqrt{2\,\pi}}\ \mathrm{e}^{-\frac{t^2}{2}}$$

est une densité de probabilité : elle est continue, et son intégrale sur $\mathbb{R}$ *est égale à* 1.

Son intégrale

$$\mathcal{N}(x) = \int_{-\infty}^{x} g(t)d(t)$$

est appelée distribution normale.

On sait en effet que $\int_{-\infty}^{+\infty}\ \mathrm{e}^{-\frac{t^2}{2}}\ dt = \sqrt{2\,\pi}$ (intégrale de Gauss).

Définition A.2.2 *On appelle loi normale (ou gaussienne) centrée réduite, et on le note* $\mathcal{N}(0,1)$ *la loi définie par la densité de probabilité normale* $g(t)$

Soit X une variable aléatoire avec une densité de probabilité normale $g(t)$, nous avons :

$$E(X) = \int_{-\infty}^{+\infty} t\, g(t)dt = \frac{1}{\sqrt{2\,\pi}}\,[-\ \mathrm{e}^{-\frac{t^2}{2}}]_{-\infty}^{+\infty} = 0$$

l'espérance est nulle : la loi est centrée.

$$Var(X) = E(X^2) - (E(X))^2 = \int_{-\infty}^{+\infty} t^2\, g(t)dt = -\int_{-\infty}^{+\infty} t\, \acute{g}(t)dt = [-t\, g(t)]_{-\infty}^{+\infty} + \int_{-\infty}^{+\infty} g(t) = 1$$

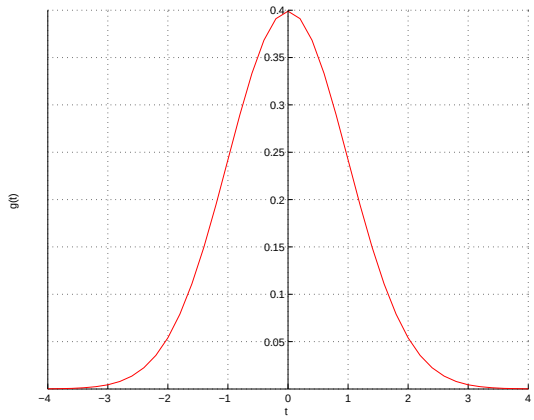


FIG. A.1 – Gaussienne centrée réduite $\mathcal{N}(0,1)$

la variance vaut 1 : la loi est réduite.

La représentation graphique de cette densité, Figure-A.1, est une courbe en cloche (ou courbe de Gauss).

Lemme A.2.1 *Soit X une variable aléatoire de loi normale centrée réduite $\mathcal{N}(0,1)$. Alors on a :*

$$(x^{-1} - x^{-3})g(x) \;<\; \Pr[X > x] \;<\; x^{-1}g(t),\ x > 0$$

▷ PREUVE : On montre aisément que

$$(1 - 2x^{-4})g(x) \;<\; g(x) \;<\; (1 + x^{-2})g(x)$$

par intégration entre 0 et l'infini, on obtient le résultat. ◁

A.2.1 La loi normale générale

Soient X une variable aléatoire suivant la loi normale centrée réduite, et deux réels μ, σ, où $\sigma > 0$.

On définit la variable aléatoire $Y = \sigma X + \mu$, dont on note F la fonction de répartition.

On a $\mathrm{E}(Y) = \sigma\,\mathrm{E}(X) + \mu = \mu$ et $\mathrm{V}(Y) = \sigma^2\,\mathrm{V}(X) = \sigma^2$ puisque $\mathrm{E}(X) = 0$ et $\mathrm{V}(X) = 1$.

Cherchons la loi de Y : pour tout $x \in \mathbb{R}$,

$$F(x) = \Pr[Y \le x] = \Pr[\sigma X + \mu \le x] = \Pr\left[X \le \frac{x-\mu}{\sigma}\right] = \mathcal{N}\left(\frac{x-\mu}{\sigma}\right),$$

puisque la fonction de répartition de X est Φ.

Ainsi, F est continûment (et même indéfiniment) dérivable : Y suit une loi à densité, et la dérivée f de F est une densité de probabilité de cette variable aléatoire ; pour tout $x \in \mathbb{R}$,

$$f(x) = \mathcal{F}'_m(x) = \frac{1}{\sigma}\acute{\mathcal{N}}\left(\frac{x-\mu}{\sigma}\right) = \frac{1}{\sigma}\,g\left(\frac{x-\mu}{\sigma}\right) = \frac{1}{\sigma\sqrt{2\,\pi}}\ \mathrm{e}^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}.$$

Nous donnons la définition générale suivante.

Définition A.2.3 *On appelle loi normale (ou gaussienne) de paramètres μ, σ^2 (où $\sigma > 0$), notée $\mathcal{N}(\mu, \sigma^2)$, la loi de probabilité définie par la densité $f : \mathbb{R} \to \mathbb{R}^+$, telle que pour tout $x \in \mathbb{R}$:*

$$f(x) = \frac{1}{\sigma}\,g\left(\frac{x-\mu}{\sigma}\right) = \frac{1}{\sigma\sqrt{2\,\pi}}\ \mathrm{e}^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}.$$

A.3 Théorème central limite

Il fut établi par Liapounoff et Lindeberg. On se place dans une situation d'épreuves répétées, caractérisées par une suite $X_1, X_2, X_3, \cdots, X_n$ de n variables aléatoires indépendantes et de même loi (espérance $E(X_i) = m$ et variance $V(X_i) = \sigma^2$). On définit ainsi une nouvelle variable aléatoire : la somme $S_n = X_1 + X_2 + \cdots + X_n$ Telles que : $E(S_n) = nm$, $V(S_n) = n\sigma^2$

Ces formules nous donne l'estimation de l'erreur sur les canaux de transmission sans mémoire (dont l'erreur est une variable aléatoire indépendante des événements passés sur le canal, d'où vient la nomination).

Théorème A.3.1 *(Théorème central limite) Soit la variable aléatoire S_n résultant de la somme de n variables, $X_1, \ldots, X_n$, aléatoires indépendantes et de même loi, avec une espérance fini $\mu = E(X_k)$ et une variance finie $\sigma^2 = Var(X_k)$. On construit la variable centrée réduite telle que :*

$$Z_n = \frac{S_n - n\mu}{\sigma\sqrt{n}}$$

Alors pour tout $x \in R$, la fonction de répartition $F_n(x) = \Pr[Z_n < x]$,a la limite suivante :

$$F_n(x) = \Pr[Z_n < x] \xrightarrow[n\to\infty]{} \mathcal{N}(x)$$

où $\mathcal{N}$ est la distribution normale (définition :A.2.1).

Une variable aléatoire résultant de la somme de plusieurs v.a. ayant même loi et mêmes paramètres est distribuée suivant une loi normale lorsque le nombre d'épreuves n

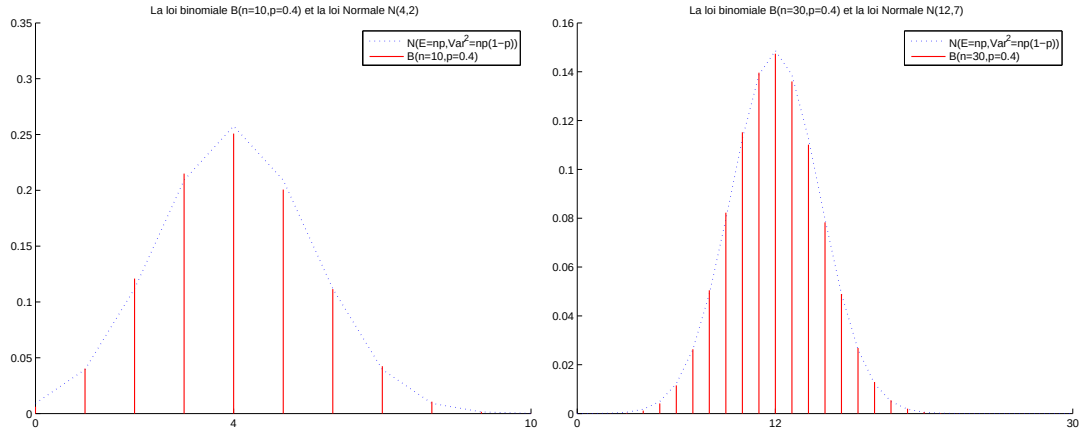


FIG. A.2 – La loi binomiale et la normale

tend vers l'infini. Le théorème central limite s'applique quelque soit la loi de probabilité suivie par les variables aléatoires discrètes ou continues, pourvu que ***les épreuves soient indépendantes***, reproductibles et en très grand nombre.
Grâce au théorème centrale limite, on peut voir que des phénomènes dont la variation est engendrée par un nombre important de causes indépendantes sont généralement susceptibles d'être représentés par une loi normale.

A.3.1 Approximation gaussienne

Soit X une variable aléatoire de loi binomiale $B(n,p)$, l'approximation gaussienne se base sur le théorème central limite qui spécifie que lorsque n tend vers l'infini, la loi de la variable

$$Z = \frac{X - \lceil np \rceil}{\sqrt{np(1-p)}}$$

tend vers la loi normale centrée réduite. Alors pour des valeurs de n assez grandes nous avons :

$$\Pr[X]\ dX \approx g(Z)\ dZ = \frac{1}{\sqrt{2\pi}}\ g\left(\frac{X - \lceil np \rceil}{\sqrt{np(1-p)}}\right) \frac{1}{\sqrt{np(1-p)}}\, dX \tag{A.1}$$

Donc une approximation de la loi binomiale est donnée par la formule suivante :

$$\Pr[X = k] = \binom{n}{k} p^k (1-p)^{n-k} \approx \frac{1}{\sqrt{2\pi np(1-p)}}\ \mathrm{e}^{\left(-\frac{(k-\lceil np \rceil)^2}{2np(1-p)}\right)} \tag{A.2}$$

Nous pouvons voir sur la figure-A.2 que même pour une petite valeur de n la loi binomiale est bien proche de la loi normale. et plus particulièrement, pour $n = 2m,\ k = m,\ p = \frac{1}{2}$ nous obtenons :

$$\binom{2m}{m} \approx \frac{2^{2m}}{\sqrt{m\pi}} \tag{A.3}$$

A.3.2 Quelques formules

L'inégalité de Markov et l'inégalité de Bienaymé-Tchébycheff s'appliquent aussi bien aux variables aléatoires discrètes ou absolument continues. Elles permettent pour une variable aléatoire X d'espérance $E(X) = m$ et de variance σ^2 d'évaluer la probabilité pour que X diffère de la moyenne d'une quantité inférieure à une valeur δ. Nous voulons démontrer que plus l'écart-type d'une variable aléatoire est faible, plus sa distribution de probabilité est concentrée autour de son espérance mathématique. Afin de démontrer cela, nous allons présenter tout d'abord l'inégalité de Markov.

Définition A.3.1 *Inégalité de Markov : Soit X une variable aléatoire positive telle que la quantité $E(|X|^r)$ existe et fini pour $r > 0$. Pour tout $\epsilon > 0$ et tout $r > 0$*

$$\Pr[X \geq \epsilon] \leq \frac{E[X^r]}{\epsilon^r}$$

où | | désigne la valeur absolue :

Dans l'analyse d'un algorithme manipulant des variables aléatoires, il est essentiel de pouvoir borner la probabilité qu'une variable aléatoire X s'éloigne de son espérance $E(X)$. En fait, lorsqu'une variable aléatoire est générée plusieurs fois, cela revient à borner la vitesse de convergence de la valeur moyenne des tirages obtenus.

Définition A.3.2 *L'inégalité de Chebyshev(Bienaymé-Tchébycheff) : Soit X une variable aléatoire telle que la quantité $E(|X|^r)$ existe et fini pour $r > 0$. Pour tout $\epsilon > 0$, et toute valeur c on a*

$$\Pr[|X - c| \geq \epsilon] \leq \frac{E(|X - c|^r)}{\epsilon^r}$$

En particulier, si $c = E(X)$, $r = 2$ alors

$$\Pr[|X - E(X)| \geq \epsilon] \leq \frac{Var(X)}{\epsilon^2}$$

L'inégalité de Chebyshev donne une majoration de la probabilité de succès de l'évènement $|X - E(X)| \geq \epsilon$. Et comme

$$\Pr[|X - E(X)| \geq \epsilon] + \Pr[|X - E(X)| < \epsilon] = 1$$

elle donne une minoration de la probabilité d'échec.

Définition A.3.3 *(Okamoto [37]) L'inégalité de Chernoff dans le cas spécial (loi binomiale) : Soit $X_1, \ldots, \mathcal{X}_n$ des variables aléatoires booléennes indépendantes de même loi pour lesquelles $\Pr[X_i = 1] = p < \frac{1}{2}$, considérons la somme $S_n = X_1 + \cdots + X_n$. Pour tout δ, $0 < \delta \leq 1 - p$,*

$$\Pr[S_n - E(S_n) \geq n\delta] \leq e^{-2n\delta^2}$$

Annexe B

Préalables mathématiques

Définition d'un groupe

Définition B.0.4 *Un groupe est un ensemble G, muni d'une loi de composition interne (lci), c'est à dire une application de $G \times G \to G$, généralement notée par la concaténation ($(x, y) \mapsto xy$), vérifiant :*

- *la loi est associative :* $\forall x, y, z \in G, (xy)z = x(yz)$
- *Il existe un élément neutre :* $\exists 1 \in G \mid \forall x,\ x.1 = 1.x = x$, *1 est dit l'élément neutre*
- *Tout élément de G possède un inverse dans G :* $\forall x, \exists x^{-1} \in G \mid xx^{-1} = x^{-1}x = 1$

Un sous ensemble H de G est dit sous groupe, si la restriction de la loi interne de G à H définit une structure de groupe sur H.

Définition B.0.5 *$H \subset G$ est un sous groupe de G, si et seulement si :*

- $1 \in H$
- $(x, y) \in H^2 \to xy \in H$
- $\forall x x^{-1} \in H$

On appelle groupe symétrique d'un ensemble G l'ensemble des permutations de cet ensemble.

Définition B.0.6 *On note $\mathcal{S}_n$, n-ième groupe symétrique standard l'ensemble des permutations de $\mathcal{I}_n = \{1, 2, ..., n\}$.*

Tous les groupes symétriques sur des ensembles de cardinal n sont isomorphes à $\mathcal{S}_n$.

Avec G un groupe et X un ensemble, on appelle action à gauche de G sur X une application α de $G \times X$ dans X telle que :

- $\alpha(\mathbf{1}, x) = x$, $\mathbf{1}$ l'élément neutre de G
- $\alpha(g, \alpha(h, x)) = (g.h, x)$

On dit aussi que G opère à gauche sur X où que G est une opération à gauche sur X. Usuellement on note plus simplement $g.x$ au lieu de $\alpha(g, x)$. Les deux conditions deviennent alors :

- 1.x=x
- (g.h).x=g.(h.x)

Notations

Nous donnons ici la liste des notations utilisées dans cette thèse ainsi que la page de leur définition.

$\stackrel{\triangle}{=}$	:	Par définition :
$\mathbb{F}_p$	:	Corps premier de caractéristique p..
$< .,. >$	:	le produit scalaire euclidien usuel : $< a, b >= \sum_{i=1}^{n} a_i b_i$
m	:	Nombre de variables d'une fonction du type $\mathbb{F}_p^m \rightarrow \mathbb{F}_p$.
α, β	:	Les éléments de l'espace vectoriel $\mathbb{F}_2^m$.
$\mathcal{C}[n, k]$	:	Code Correcteur de Longueur n et dimension k.
$\mathcal{C}^{\perp}$	:	Le code dual du code considéré.
n	:	Longueur du code considéré.
$RM(r, m)$	:	Code de Reed-Muller de longueur 2^m et d'ordre r sur l'alphabet $\mathbb{F}_2$ (p. 34)
$wt(x)$	:	Le poids d'un mot binaire x (p. 17).
$wt(f)$	:	Le poids de la fonction f.
$d(x, y)$	:	distance de Hamming entre x et y (p. 17).
$deg(f)$	:	Le degré de la fonction f.
$supp(f)$	:	Le support de la fonction f.
$(e_1, \ldots, e_m)$	:	La base canonique de l'espace vectoriel $\mathbb{F}_2^m$.
	:	
BSC	:	Le canal binaire symétrique (p. 13).
$AWGN$	:	Le canal additif gaussien blanc (p. 14).
$\mid . \mid$	:	La complexité en nombre d'opérations élémentaires d'un algorithme
$H(x)$	:	La fonction d'entropie de SHANNON
$E(.)$	:	L'espérance d'une variable aléatoire
$Var(.)$	:	La variance d'une variable aléatoire
$\Pr[(].)$	:	La probabilité d'un événement
ln	:	La logarithme népérien
log	:	La logarithme à base deux
$GL_n(\mathbb{K})$	:	l'ensemble des matrices carrées inversibles d'ordre n à coefficient dans un corps $\mathbb{K}$.
$\mathcal{S}_n$	:	Le groupe symétrique de toutes les permutations de n-symboles.

Table des figures

Liste des tableaux

Index

Bibliographie

[1] A. E. Ashikhmin and S. Litsyn. Simple MAP decoding of first-order Reed-Muller and Hamming codes. *IEEE Transactions on Information Theory*, 50(8) :1812–1818, 2004.

[2] A. E. Ashikhmin and S. N. Litsyn. Fast decoding algorithms for first order Reed-Muller and related codes. *Des. Codes Cryptography*, 7(3) :187–214, 1996.

[3] D. Augot. Introduction à l'algorithme de Sudan sur les codes de Reed-Solomon. *Gazette des Mathématiciens*, 98 :5–13, Octobre 2003.

[4] Y. Be'ery and J. Snyders. Optimal soft decision block decoders based on fast hadamard transform. *IEEE Trans. Inf. Theor.*, 32(3) :355–364, 1986.

[5] P. Charpin. *Codes idéaux de certaines algèbres modulaires*. PhD thesis, Université Paris VII, 1982.

[6] B. Chor, A. Fiat, and M. Naor. Tracing Traitors. *Lecture Notes in Computer Science*, 839 :257–270, 1994.

[7] I. Dumer. Decoding of Reed-Muller codes on Pascal triangles. *Communication Theory and Applications,*, 4 :49–62, 2000.

[8] I. Dumer. Decoding of Reed-Muller codes with polylogarithmic complexity. In *WISICT '04 : Proceedings of the winter international synposium on Information and communication technologies*, pages 1–6. Trinity College Dublin, 2004.

[9] I. Dumer. Recursive decoding and its performance for low-rate Reed-Muller codes. *IEEE Trans. Info. Theory,*, 50(5) :811–823, May 2004.

[10] I. Dumer. Soft decision decoding of Reed-Muller codes : a simplified algorithm. *IEEE Trans. Info. Theory,*, 52(3) :253–269, March 2006.

[11] I. Dumer. Recursive decoding of Reed-Muller codes. In *Proc37 Annual Allerton Conf. on commun.,control, and comp.*, pages 63–69, Sept. 22-24, 1999.

[12] I. Dumer and M. Burnashev. Error exponents for recursive decoding of Reed-Muller codes on a binary-symmetric channel. *IEEE Transactions on Information Theory*, 52(11) :4880–4891, 2006.

[13] I. Dumer and R. E. Krichevskiy. Soft-decision majority decoding of Reed-Muller codes. *IEEE Transactions on Information Theory*, 46(1) :258–264, January 2000.

[14] I. Dumer and K. Shabunov. Recursive constructions and their maximum likelihood decoding. In *Proc. 38 Allerton Conf. on Commun. Contr., and Computing*, pages 71–80, 2000.

[15] I. Dumer and K. Shabunov. Recursive list decoding for Reed-Muller codes and their subcodes. *In : Information, Coding and Mathematics,*, pages 279–298, 2002.

[16] I. Dumer and K. Shabunov. Recursive error correction for general Reed-Muller codes. *Discrete Applied Mathematics*, 154 :253–269, 2006.

[17] I. Dumer and K. Shabunov. Soft decision decoding of Reed-Muller codes : recursive lists. *IEEE Trans. Info. Theory,*, 52(3) :253–269, March 2006.

[18] P. Elias. List decoding for noisy channels. *1957-IRE WESCON Convention Record, Pt. 2,*, pages 94–104, 1957.

[19] P. Elias. Zero error capacity under list decoding. *IEEE Transactions on Information Theory*, 34(5) :1070–1074, 1988.

[20] W. Feller. *An introduction to probability theory and its applications*, volume I. John Wiley & Sons, third edition, 1950.

[21] C. Fontaine. *Contribution à la recherche de fonctions booléennes hautement non linéaires, et au marquage d'images en vue de la protection des droits d'auteur.* PhD thesis, Université Paris VI, 1998.

[22] G. D. Forney. Coset codes-I : Introduction and geometrical classification. *IEEE Transactions on Information Theory*, 34(5) :1123–1151, 1988.

[23] G. D. Forney. Coset codes-II : Binary lattices and related codes. *IEEE Transactions on Information Theory*, 34(5) :1152–1187, 1988.

[24] O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *STOC '89 : Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 25–32, New York, NY, USA, 1989. ACM Press.

[25] O. Goldreich, R. Rubinfeld, and M. Sudan. Learning polynomials with queries : the highly noisy case. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 294–303. IEEE Computer Society Press, Los Alamitos, CA, 1995.

[26] A. E. Heydtmann and T. Jakobsen. Decoding Reed-Muller codes beyond half the minimum distance,. *Proceedings of the Fifth International Conference on Finite Fields and Applications, Augsburg, Germany*, 1999.

[27] G. Kabatianskiy and C. Tavernier. List decoding of second order Reed-Muller codes. In *Proceedings of Eight International Simposium on Communication Theory and Applications*, pages 230–235, Ambelside, UK, July 2005.

[28] G. Kabatianskiy and C. Tavernier. List decoding of Reed-Muller codes of first order. In *Proc. of ACCT'9*, pages 230–235, Kranevo, Bulgaria, June 2004.

[29] G. A. Kabatianskiy. On decoding Reed-Muller codes in semicontinuous channel. In *Proc. Second Int. Workshop Algebr. and Combin. Coding Theory*, pages 87–91, 1990.

[30] R. E. Krichevskiy. On the number of reed-muller code correctable errors,. *Dokl. Soviet Acad. Sciences,*, vol. 191 :pp.541–547, 1970.

[31] V. Lint. *Introduction to Coding Theory*. Springer-Verlag, 1992.

[32] S. N. Litsyn. On decoding complexity of low-rate Reed-Muller codes,. *Proc. 9th ALL-Union Conf. on Coding Theory and Inf. Transmission*, Part 1 :pp.202–204, 1988.

[33] P. Loidreau and B. Sakkour. Modified version of sidel'nikov-peshakov decoding algorithm for binary second order Reed-Muller codes. In *Proc. of ACCT'9*, pages 266–272, Kranevo, Bulgaria, June 2004.

[34] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland,Amsterdam, 1981.

[35] M. Matsui. Linear cryptanalysis method for DES cipher. In *EUROCRYPT '93 : Workshop on the theory and application of cryptographic techniques on Advances in cryptology*, pages 386–397, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc.

[36] D. E. Muller. Application of boolean algebra to switching circuit design and to error detection. *IRE Trans. Elec. Comp,*, 3 :6–12, Sept. 1954.

[37] M. Okamoto. Some inequalities relating to partial sum of binomial probabilities. *Annals of institut of Statistical Mathematics*.

[38] R. Pellikaan and Xin-Wen Wu. List decoding of q-ary Reed-Muller codes. *IEEE Transactions on Information Theory*, 50(4) :679–682, 2004.

[39] M. Plotkin. Binary codes with specified minimum distance. *IEEE Transactions on Information Theory*, IT-6 :445–450, 1960.

[40] A. Poli and L. Huguet. *Error Correcting Codes, Theory and Applications*. Masson and Prentice Hall, 1989.

[41] V. S. Pless R. A. Brualdi and W. C. Huffman. *Handbook of Coding Theory, Volume I*. Elsevier Science Inc., North-Holland, 1998.

[42] V. S. Pless R. A. Brualdi and W. C. Huffman. *Handbook of Coding Theory, Volume II*. Elsevier Science Inc., North-Holland, 1998.

[43] I. S. Reed. A class of multiple error correcting codes and the decoding scheme,. *IEEE Trans.Info.Theory*, vol.IT-4 :pp.38–49, 1954.

[44] S. Roman. *Coding and information theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1992.

[45] B. Sakkour. Decoding of second order Reed-Muller codes with a large number of errors,. In *ITW2005 - IEEE ITSOC Information Theory Workshop 2005*, Rotoua, New Zealand, 2005.

[46] G. Schnabl and M. Bossert. Soft-decision decoding of Reed-Muller codes as generalized multiple concatenated codes. *IEEE Transactions on Information Theory*, 41(1) :304–308, 1995.

[47] G. Seroussi and A. Lempel. Maximum likelihood decoding of certain Reed-Muller codes. *IEEE Transactions on Information Theory*, 29(3) :448–450, 1983.

[48] A. Seznec and N. Sendrier. HAVEGE : A user-level software heuristic for generating empirically strong random numbers. *ACM Trans. Model. Comput. Simul.*, 13(4) :334–346, 2003.

[49] V. M. Sidel'nikov. A public-key cryptosystem based on binary Reed-Muller codes. *Discrete Applied Mathematics,*, 4(3) :253–269, 1994.

[50] V. M. Sidel'nikov and A. S. Pershakov. Decoding of Reed-Muller codes with a large number of errors. *Problemy Peredachi Informatsii*, 28 :80–94, 1992.

[51] J. Snyders and Y. Be'ery. Maximum likelihood soft decoding of binary block codes and decoders for the Golay codes. *IEEE Transactions on Information Theory*, 35(5) :963–975, 1989.

[52] M. Sudan. Decoding of Reed-Solomon codes beyond the error-correction bound. *Journal of Complexity*, 28 :13(1) :180–193, Mars 1997.

[53] T. Kløve T. Helleseth and V. Levenshtein. Bounds on error-correction capability of codes beyond half the minimum distance. In *Workshop on Coding and Cryptography 2003*, pages 243–251, Versailles, France, March 24-28, 2003.

[54] C. Tavernier. *Testeurs, problèmes de reconstruction univariés et multivariés, et application à la cryptanalyse du DES*. PhD thesis, Ecole Polytechnique, 2004.

[55] A. Vardy. Algorithmic complexity in coding theory and the minimum distance problem. In *STOC*, pages 92–109, 1997.

[56] H. Vater. Binary coding by integration of polynomials. *IEEE Transactions on Information Theory*, 40(5) :1417–1424, Sept. 1994.

[57] A.J. Viterbi. Error bounds for convolutional codes and as asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, IT-13(2) :260–269, 1967.

[58] X. M. Wang Y. X. Li, R. Deng. On the equivalence of McEliece's and Niederreiter's public key cryptosystems. *IEEE Trans. IT*, 40(1) :271–273, Jan. 1994.

Étude et amélioration du décodage des codes de Reed-Muller d'ordre deux.

Résumé : Dans cette thèse, nous étudions les codes de Reed-Muller qui constituent une des familles de codes correcteurs les plus étudiées, et les plus utilisées dans la transmission des communications numériques. Grâce à leur rapidité d'encodage et de décodage, ils furent notamment utilisés pour les transmissions satellitaires. Ils ont également un lien très fort avec les notions de fonctions booléennes. L'étude de ces dernières constitue le coeur de la réalisation et de la sécurité des systèmes de chiffrement à clé secrète, tant par blocs que par flot. Depuis l'introduction de ces codes, de très nombreux algorithmes de décodage virent le jour, et aujourd'hui encore étudier leur structure afin de construire des algorithmes de décodage constitue un fructueux domaine de recherche. Ces algorithmes de décodage peuvent être utilisés dans l'étude de la structure de systèmes de chiffrement à clé secrète.

Nous exposons un point de vue unificateur à l'ensemble des algorithmes de décodage des codes de Reed-Muller, ce point de vue étant celui de la dérivée discrète. Nous exposons un algorithme performant pour le décodage des codes d'ordre deux, que nous analysons ensuite. Nous discutons les résultats de simulations des algorithmes étudiés pour les petites et moyennes longueurs de code. Ces résultats montrent que l'algorithme proposé décode beaucoup plus loin en pratique que les autres algorithmes.

Study and improvement of the decoding of Reed-Muller codes of second order.

Abstract : In this thesis, we study the Reed-Muller codes which constitute one of the classes of error correcting codes the most studied, and most used in numerical communications. Thanks to their speed of encoding and decoding, they were in particular used for the satellite transmissions. They also have a strong bond with the concepts of Boolean functions. The study of Boolean functions constitutes the heart of the realization and the safety of secret key cryptography. Since the introduction of these codes, many decoding algorithms were introduced, and even today the study of their structure in order to build decoding algorithms constitutes an interested field of research in coding theory and in cryptography for finding good linear, quadratic etc approximations to Boolean functions used in cryptography

We expose a unifying point of view to all known decoding algorithms of this codes, this point of view is that of the discrete derivative. We expose a powerful algorithm for the decoding of the codes of order two, which we analyze then. We discuss the results of simulations of the algorithms studied for the small and average lengths of code. Simulation results show that the proposed algorithm decodes in practice much further that the other algorithms.

Discipline : Mathématiques et Informatique

Laboratoire : UMA, ENSTA, 32 Boulevard Victor, 75739 Paris Cedex 15, France.