Introduction aux codes correcteurs d'erreurs

Pierre Abbrugiati

23 janvier 2006

II

Table des matières

Chapitre 1

Introduction

Par codes, on peut entendre plusieurs concepts bien distincts : cryptographie (RSA,...) ; codes de compression (Huffman,...) ; codes correcteurs d'erreurs. Dans ce cours, on s'intéresse aux codes correcteurs d'erreur ; plus précisément à la famille des codes en bloc.

Lorsqu'on envoie un message à travers un canal de transmission des données (par exemple : en téléchargeant ce cours sur internet), des erreurs de transmission peuvent se produire. Le but est d'arriver à détecter, voire corriger des erreurs.

Notre modèle est le suivant :
- On considère que le message est une suite de bits...
- ... regroupés en blocs de k bits ($k = 1$, ou 4, ou 8, ou 256, etc.)
- ... et chaque bit a une probabilité $p \ll \frac{1}{2}$ donnée d'être inversé.

On se propose de "coder" chaque bloc du message initial en un bloc plus gros (avec des redondances d'information).

Exemple 1.1 *Code par adjonction d'un bit de parité* $(8, 9)$
On découpe notre message initial en blocs de 8 bits.
On transforme ensuite chaque bloc en un bloc de 9 bits en ajoutant un bit à la fin de chaque bloc de telle sorte que la somme des bits des nouveaux blocs soit toujours paire.

Si **une** erreur se produit, on peut la détecter, mais pas la localiser : on ne peut pas corriger notre bloc, il faut recommencer la transmission. Si d'avantage d'erreurs se produisent, on n'est même pas sûr de détecter le problème.

Et donc... qu'attend-on d'un bon code ?

1. L'information ne doit pas être trop diluée.
2. On doit pouvoir **détecter** et **corriger** un nombre raisonnable d'erreurs.
3. L'algorithme de codage doit être suffisamment rapide.
4. L'algorithme de décodage (corrections incluses) doit être suffisamment rapide.

Chapitre 2

Les trois principaux paramètres d'un code

2.1 Dimension et longueur d'un code

Terminologie et notations préliminaires :

Un bloc de k bits sera indifféremment appelé **bloc**, **mot** ou **vecteur**.
L'ensemble des mots de k bits sera noté $\{0,1\}^k$.
On parlera indifféremment de **bits** ou de **lettres**.
Un mot m de k bits sera noté $m_1m_2...m_k$, ou éventuellement $\begin{pmatrix} m_1 \\ \vdots \\ m_k \end{pmatrix}$.

Remarque préliminaire :
Il y a 2^k mots de k bits.

Le principe du codage est le suivant : après avoir découpé notre message en blocs de k bits, on va appliquer un même algorithme sur chaque bloc :
a) ou bien en rajoutant des bits de contrôle à la fin de chaque bloc
b) ou bien en modifiant complêtement les blocs, mais en évitant que deux blocs différents soient transformés en un même bloc.

D'où la définition suivante :

Définition 2.1 *(codes (k,n))*
Un code est une application injective $\phi : \{0,1\}^k \to \{0,1\}^n$
*Le paramêtre k est appelé la **dimension** du code ϕ et le paramêtre n est appelé la **longueur** du code : on dit que ϕ est un code de paramètres (k,n).*
Si de plus pour tout mot m de $\{0,1\}^k$, m est un préfixe de $\phi(m)$ (c'est à dire si l'application de ϕ consiste seulement à rajouter des bits de contrôle), on dira que ϕ est un code systématique.

Sauf mention contraire, tous les codes étudiés par la suite seront systématiques.

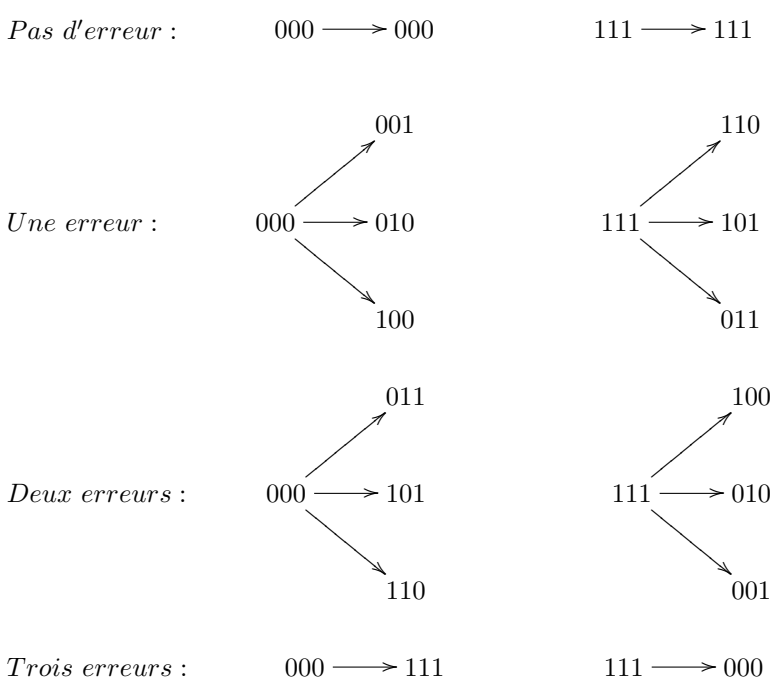
Définition 2.2 *(image d'un code)*
L'ensemble $C = \{\phi(m),\ m \in \{0,1\}^k\}$ *est appelé* ***l'image*** *du code* ϕ.
Les éléments de C *sont appelés les* ***mots de code*** *de* ϕ *(en opposition aux éléments "originels" de* $\{0,1\}^k$ *qui sont appelés mots de source).*
Deux codes ayant la même image sont dits ***équivalents****.*

Exemple 2.3
a) *Le code parité présenté dans l'introduction est un code de paramètres* $(8,9)$.
On peut aussi former des codes de parité de paramètres $(k, k+1)$ *pour tout entier strictement positif* k.
b) *Exemple de code de paramètres* $(1,3)$ *: l'application* $\begin{cases} 0 \mapsto 000 \\ 1 \mapsto 111 \end{cases}$ *(logiquement appelé code de répétition pure* $(1,3)$*)*

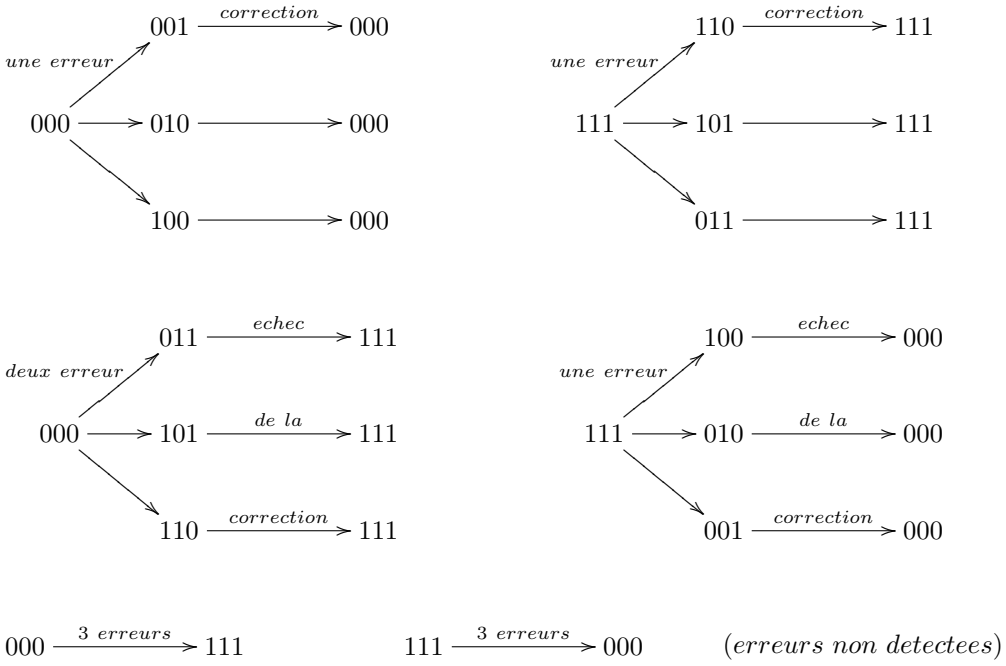
2.2 L'algorithme de décodage naïf

Attardons-nous sur le code de répétition pure $(1,3)$.

Que peuvent devenir les mots de code après erreur ?



Si une, ou même deux erreurs se produisent, le mot reçu n'est pas un mot de code, l'erreur est donc détectée.
Comment corriger ? Si le mot reçu n'est pas un mot de code, la probabilité qu'il se soit produit une erreur est plus importante que la probabilité que deux erreurs aient eu lieu. Il est donc plus raisonnable de corriger par le mot de code le plus "proche". On peut alors corriger une erreur, mais pas deux :



En résumé, le code de répétition pure $(1, 3)$ est très satisfaisant vis à vis des points **2** et **3**, et satisfaisant vis à vis du point **4**, mais inacceptable vis à vis du point **1**. Généralisons maintenant l'algorithme de décodage qu'on a mis en oeuvre. On est amené à définir proprement cette notion de "proximité" et d"́éloignement".

Définition 2.4 *(Poids et distance de Hamming)*
Soient m et m' deux mots de $\{0,1\}^k$.
On appelle ***distance de Hamming*** *entre m et m', et on note $d(m, m')$ le nombre de lettres distinctes de m et m'.*
On appelle ***poids*** *de m, et on note $w(n)$ le nombre de lettres non nulles de m (donc le nombre de bits de m égux à 1).*
Anecdotiquement, on a les deux relations suivantes : $d(m, m') = w(m + m')$ et donc $w(m) = d(m, 00...0)$, où $+$ désigne l'addition bit à bit (avec $1 + 1 = 0 \ldots$)

Algorithme général de décodage (naïf) :
Etant donné le mot reçu r, on cherche le mot de code m qui réalise le minimum de $d(r, m)$, et on décode r par m

Cet algorithme soulève **deux problèmes** :
a) Sa complexité est en $O(n.2^k)$. C'est acceptable si k est petit comme dans le cas du code de répétition pure $(1, 3)$, mais c'est impraticable dès que k est grand. Or, pour satisfaire le point **1**, il est clair que k doit être grand.
b) A priori, rien ne garantit que le mot de code qui réalise le minimum de $d(r, m)$ soit **unique**. C'est le cas du code de répétition pure $(1, 3)$, mais les codes ayant cette propriété (codes parfaits) sont très rares.

Dans les prochains cours, on élaborera donc des stratégies afin d'obtenir des codes pour lesquels on a des algorithmes de décodage plus rapides.

2.3 Distance minimale d'un code

Définition 2.5
Soit ϕ un code d'image C.
On appelle capacité de détection de ϕ le plus grand entier e_d tel qu'on soit toujours capable de détecter e_d erreurs ou moins.
On appelle capacité de correction de ϕ le plus grand entier e_c tel qu'on soit toujours capable de corriger e_c erreurs ou moins.
On appelle distance minimale de ϕ et on note d_ϕ (ou d_C) la plus petite distance non nulle entre deux mots de code.
On a $e_c = d_\phi - 1$ et $e_c = \lfloor \frac{d_\phi - 1}{2} \rfloor$

Notation :
La distance minimale d'un code quantifie donc sa qualité vis à vis du point **1**. C'est un paramètre important. En abrégé, "un code de dimension k, de longueur n et de distance minimale d" se dira "un code de paramètres (k, n, d)" ou même "un code (k, n, d)".

Exemple 2.6 *(code de répétition pure $(1, 3)$)*
Prenons l'exemple où ϕ est le code de répétition pure $(1, 3)$.
Son image est $C = \{000, 111\}$ donc sa distance minimale d_ϕ est $d(000, 111) = 3$
On retrouve bien $e_d = d_\phi - 1 = 2$ et $e_c = \lfloor \frac{d_\phi - 1}{2} \rfloor = 1$ comme attendu.

Exemple 2.7 *(code par bit de parité* $(8,9)$*)*
Prenons maintenant l'exemple où ϕ *est le code par bit de parité* $(8,9)$*.*
Son image C *a* $2^8 = 256$ *éléments. Il est donc exclu de comparer tous les mots de code distincts : il nous faudrait faire* $9 * C_{256}^2 = 293760$ *comparaisons !*
Néanmoins, on peut établir que sa distance minimale d_ϕ *est* 2*.*

Démonstration :
· 000000000 et 000000011 appartiennent à C (ce sont les deux premiers mots de code) donc la distance minimale d_ϕ vérifie $d_\phi \leq d(000000000, 000000011) = 2$.
· Prenons deux mots de code distincts. Ils s'écrivent $a_1...a_8p$ et $a'_1...a'_8p'$ avec $p = a_1 + ... + a_8$ et $p' = a'_1 + ... + a'_8$. Alors de deux choses l'une :
- ou bien $d(a_1...a_8, a'_1...a'_8) \geq 2$ et alors *a fortiori* $d(a_1...a_8p, a'_1...a'_8p') \geq 2$
- ou bien $d(a_1...a_8, a'_1...a'_8) = 1$, c'est à dire qu'il y a exactement **une** lettre de différence entre les mots $a_1...a_8$ et$a'_1...a'_8$, et donc que $p = a_1 + ... + a_8 \neq a'_1 + ... + a'_8 = p'$; il s'ensuit alors que $d(a_1...a_8p, a'_1...a'_8p') = 2$

Cette analyse montre que dans tous les cas on a $d(a_1...a_8p, a'_1...a'_8p') \geq 2$, et donc que la distance minimale d_ϕ vérifie $d_\phi \geq 2$.
· En conclusion, $d_\phi = 2$.
On retouve alors comme prévu $e_d = d_\phi - 1 = 1$ et $e_c = \lfloor \frac{d_\phi - 1}{2} \rfloor = 0$

Chapitre 3

Généralités sur les codes linéaires

3.1 Définitions d'un code linéaire

Définition 3.1 *(code linéaire)*

Un code ϕ de paramètres (k, n) est dit ***linéaire*** *s'il existe une matrice $G \in \mathcal{M}_{n,k}(\mathbb{F}_2)$ (c'est à dire avec n lignes, k colonnes, à coefficients dans $\{0,1\}$), de rang k, telle que $\forall m \in \{0,1\}^k,\ \phi(m) = G \times m$.*

La multiplication matricielle est à comprendre dans $\mathbb{F}_2$ (c'est à dire modulo 2) et le mot m est ici considéré comme un vecteur colonne. La condition sur le rang traduit l'injectivité de ϕ.

La matrice G est appelée ***matrice génératrice*** *de ϕ.*

Enfin, dire que le code ϕ est systématique, c'est dire que la matrice G est le la forme $\begin{pmatrix} I_k \\ G' \end{pmatrix}$, où I_k est la matrice identité d'ordre k.

Exemple 3.2

a) *Considérons ϕ, code linéaire de matrice génératrice* $\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix}$

C'est un code de dimension 2, de longueur 4, donné par le tableau :

x	$\phi(x)$
00	*0000*
01	*0101*
10	*1011*
11	*1110*

b) *Tous les codes rencontrés jusqu'à présent, exception faite du code du td 1, exercice 3, sont linéaires.*

. *Pour le code de bit de parité* $(8,9)$*, la matrice génératrice est* $\begin{pmatrix} & I_8 & \\ 1 & \dots & 1 \end{pmatrix}$

. *Pour le code de répétition pure* $(1,3)$*, la matrice génératrice est* $\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$

. *Pour le code du td 1, exercice 4, la matrice génératrice est* $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$

Le caractère systématique de ces codes se lit sur la matrice génératrice.

c) *Un* ***code de Hamming systématique*** *de paramètres* $(4,7)$ *est le code linéaire* $a_1a_2a_3a_4 \mapsto a_1a_2a_3a_4b_5b_6b_7$ *avec* $b_5 = a_1 + a_2 + a_3$*,* $b_6 = a_1 + a_2 + a_4$*,* $b_4 = a_1 + a_3 + a_4$*. Comme on le verra par la suite, il y a d'autres codes de Hamming systématiques de paramètres* $(4,7)$ *: un tel code consite en fait à rajouter à* $a_1a_2a_3a_4$ *trois bits de parité correspondant à trois choix distincts de* 3 *éléments parmi* a_1, a_2, a_3, a_4*. La matrice génératrice de celui-ci est :*

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

On peut déjà remarquer que les codes linéaires se comportent de façon satisfaisante (mais sans plus) vis à vis du point **3** : l'algorithme de codage est en effet une multiplication matricielle, dont le coût est en $O(n \times k)$, et même en $O((n-k) \times k)$ dans le cas d'un code systématique.

3.2 Distance minimale d'un code linéaire

Les codes linéaires sont également intéressants car on dispose d'informations sur leur distance minimale. Tout d'abord, par définition, dire qu'un code ϕ de paramêtres (k,n) est linéaire, c'est exactement dire que ϕ est une application linéaire injective de $\{0,1\}^k$ dans $\{0,1\}^n$. D'où la propriété suivante :

Remarque 3.3
Soit ϕ *un code linéaire. Alors son image* C *est un sous-espace vectoriel de* $\{0,1\}^n$
Autrement dit $\overrightarrow{0} \in C$ *et* $\forall m, m' \in C, m + m' \in C$*.*

On en déduit le sympatique corollaire suivant :

Proposition 3.4 *(distance minimale d'un code linéaire)*
Soit ϕ un code linéaire d'image C.
Alors la distance minimale de ϕ d_ϕ est égale au plus petit poids non nul d'un mot de C.

Démonstration : La distance minimale est le plus petit élément non nul de l'ensemble des distances entre deux mots de code. Il suffit donc de montrer que cet ensemble coïncide avec l'ensemble des poids des mots de code. Tout d'abord, tout poids est une distance car $w(m) = d(m, 0...0)$. Ensuite, toute distance est un poids car $d(m, m') = w(m + m')$ et si $m, m' \in C$ on a aussi $m + m' \in C$.

Dans le cas d'un code de dimension 3 ou 4, ce critère permet de calculer beaucoup plus facilement la distance minimale. Par exemple, on peut voir que le code de Hamming proposé précédemment est de distance minimale 3, donc 1-correcteur. Mais ce calcul reste coûteux en général.

On dispose d'autres critères sur la distance minimale des codes linéaires. Celui qui suit donne la limite de ce qu'on peut espérer :

Définition 3.5 *(borne de Singleton)*
La distance minimale d d'un code linéaire de dimension k et de longueur n vérifie $d \leq n + 1 - k$.
Un code pour lequel on a égalité est dit MDS (Maximum Distance Separable).

Démonstration : (un peu technique, n'est pas à comprendre)
D'après 3.4, il suffit d'exhiber dans C un vecteur non nul de poids inférieur ou égal à $n + 1 - k$. Par exemple, s'il existe dans C un mot non nul dont les $k - 1$ dernières composantes sont nulles, le résultat est acquis.
Mais justement ! Considérons l'ensemble E des mots dont les $k - 1$ dernières composantes sont nulles : c'est un sous-espace vectoriel de dimension $n - k - 1$ de $\{0, 1\}^n$. Pour sa part, C est un sous-espace vectoriel de dimension k. On a donc $dim(C) + dim(E) = n + 1 > n$ et donc $C \cap E$ contient au moins un élément non nul.

On peut encore avoir un autre critère sur la distance minimale d'un code linéaire : il est donné par la matrice de contrôle du code.

3.3 Matrice de contôle d'un code linéaire

L'intérêt principal de ne considérer que des codes linéaires est qu'ils disposent de meilleurs algorithmes de décodage. On utilise pour cela les matrices de contrôle.

Définition 3.6 *(Matrice de contrôle)*
Soit ϕ un code linéaire (k, n) de matrice génératrice G.
On appelle ***matrice de contrôle*** *de ϕ toute matrice $H \in \mathcal{M}_{n-k,n}$ (c'est à dire avec n colonnes et $n-k$ lignes) telle que $H.m = \overrightarrow{0} \Leftrightarrow m \in C$.*

L'existence de matrices de contrôle pour n'importe quel code linéaire ϕ n'étonnera pas les spécialistes de l'algèbre linéaire : H n'est rien d'autre qu'une matrice dont le noyau est l'image de G. Il est clair que de telles matrices existent toujours. Au passage, comme le noyau de H est C, H contient assez d'information pour reconstituer ϕ à équivalence près.
En revanche, on remarque que H n'est pas unique en général (par exemple, en permutant deux lignes d'une matrice de contrôle, on obtient encore une matrice de contrôle).

Etant donné la matrice génératrice G d'un code ϕ, se donner une matrice de contrôle H, c'est se donner une base de l'orthogonal de C dans $\{0,1\}^n$, ce qui n'est pas évident.

L'opération inverse est plus facile : étant donné une matrice de contrôle H, se donner la matrice génératrice G d'un code ϕ correspondant revient à se donner une base du noyau de H.

Le théorème suivant donne un critère très simple pour déterminer **une** matrice de contrôle d'un code systématique.

Théorème 3.7 *(matrice de contrôle d'un code systématique)*
Soit ϕ un code systématique de matrice génératrice $\begin{pmatrix} I_k \\ G' \end{pmatrix}$.
Alors la matrice $H = \begin{pmatrix} G' & I_{n-k} \end{pmatrix}$ est une matrice de contrôle de G.

Exemple 3.8
a) *Une matrice de contrôle du code de bit de parité $(8, 9)$ est $\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$.*
b) *Une matrice de contrôle du code de répétition pure $(1, 3)$ est $\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$*
c) *Une matrice de contrôle du code du code de Hamming systématique présenté à la première section est $\begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$. On pourra remarquer que cette matrice consiste simplement à écrire en colonne tous les mots de 3 bits. Une autre énumétation aurait correspondu à un autre code de Hamming $(7, 4)$.*

Proposition 3.9 *(distance minimale et matrice de contrôle)*
Soit ϕ un code linéaire de matrice de contrôle H.
Alors d_ϕ est le nombre minimal de colonnes de H linéairement dépendantes.
Par exemple :
- *Si H a une colonne nulle, on a $d_\phi = 1$*
- *Sinon, et si H a deux colonnes identiques, on a $d_\phi = 2$*
- *Sinon, et si une colonne de H est égale à la somme de deux autres, on a $d_\phi = 3$, etc.*

Démonstration : Il suffit de se rappeler que les colonnes de H forment une base de son image, donc de $ker(G) = C$, et d'invoquer 3.4.

Remarquons qu'avec ce lemme on peut retrouver que le code de Hamming présenté plus haut a bien pour distance minimale 3.

3.4 Décodage d'un code linéaire

Définition 3.10 *(Mot erreur et syndrome)*
Soit ϕ un code de paramètres (k, n), de matrice génératrice G et de matrice de contrôle H. On se fixe un mot de source X (mot de longueur k). Le mot de code correspondant sera $\phi(X) = Y$. Il y a éventuellement des erreurs durant la transmission et on reçoit le mot Z.
*On appelle **mot erreur** associé à Z le mot $E = Z + Y$.*
*On appelle **syndrome** de Z le mot $H \times Z$.*
On note E_i le mot ne contenant que des 0, sauf en position i.

Quelques commentaires sur cette définition :
Comme $E = Y + Z$, on a $Z = Y + E$, et donc E quantifie bien les erreurs survenues sur Y : par exemple, $w(E)$ est le nombre d'erreurs survenues. Le problème du décodage peut se reformuler : "trouver E", car alors on déduit Y.
Bien sûr, ce "trouver E" est à comprendre dans le sens "trouver E le plus probable", c'est à dire "trouver E de poids minimal".
Dire que le syndrome de Z est nul, c'est dire que Z est un mot du code : dans ce cas le mot erreur le plus probable est le mot nul.

Remarque 3.11
Soit Z un mot de syndrome S.
Alors E est aussi de syndrome S car $H \times E = H \times (Z + Y) = H \times Z + H \times Y$ et Y est un mot du code.
*L'ensemble des mots de syndrome S est appellé **classe lattérale** de Z.*
Le problème de décodage se reformule donc : "trouver le mot de plus petit poids dans la classe lattérale de Z".

... s'il existe !

Par exemple :

. Dire que le mot nul est dans la classe lattérale de Z, c'est dire que le syndrome $S = H \times Z$ est nul. Dans ce cas, le mot de poids minimal dans la classe lattérale de Z est le mot nul, il faudra donc corriger Z par $\phi^{-1}(Z)$.

. Supposons donc que le syndrome de $S = H \times Z$ est non nul. Dire qu'il existe un mot de poids 1 dans la classe lattérale de Z, c'est dire que le syndrome S est égal à une colonne C_i de H. Dans ce cas, s'il existe une seule colonne C_i de H égale au syndrome, le mot de poids minimal dans la classe lattérale de Z est E_i, il faudra donc corriger Z par $\phi^{-1}(Z + E_i)$, où E_i est le mot ne contenant que des zéros, sauf en position i.

. Supposons donc que le syndrome de $S = H \times Z$ est non nul, et distinct de toutes les colonnes de H. Dire qu'il existe un mot de poids 2 dans la classe lattérale de Z, c'est dire que le syndrome de Z est égal à la somme de deux colonnes $C_{i_1} + C_{i_2}$ de H. Dans ce cas, et si c'est la seule façon d'obtenir le syndrome comme somme de deux colonnes de H, le mot de poids minimal dans la classe lattérale de Z est $E_{i_1} + E_{i_2}$, il faudra donc corriger Z par $\phi^{-1}(E_{i_1} + E_{i_2})$.

... un algorithme prend forme !

Dans ce qui précède, le point délicat est l'unicité "du" mot de poids minimal dans la classe lattérale de Z. Par exemple, si deux colonnes C_i et C_j de H sont égales à $S = H \times Z$, on ne sait pas corriger. Pour commencer, faisons l'hypothèse que ce cas de figure ne se produira jamais. Il suffit pour cela de renoncer à corriger au delà de la capacité de correction, car un mot de poids $p \leq e_c$ peut s'écrire au plus d'une seule façon comme somme de colonnes de H.

Algorithme de décodage des codes linéaires, 1^{ere} version :

C'est un algorithme de décodage **en deçà de la capacité de correction**, donc non optimal (sauf si le code est parfait).

Etape 1) : on calcule $S = H \times Z$

Etape 2) : si $S = \overrightarrow{0}$, on décode Z par $\phi^{-1}(Z)$, sinon, on initialise p à 1

Etape 3) : . Si il existe p colonnes de H C_{i_1}, C_{i_1}, ..., C_{i_p} telles que $S = C_{i_1} + C_{i_1} + \ldots + C_{i_p}$, on décode par $\phi^{-1}(Z + E_{i_1} + E_{i_1} + \ldots + E_{i_p})$
. Sinon, et qu'on a $p < e_c$, on incrémente p et on recommence l'étape 3)
. Sinon, et donc qu'on a $p = e_c$, on passe à l'étape 4)

Etape 4) : échec du décodage

Remarque : Les étapes 2) et 3) peuvent fusionner (on initialise p à 0).

L'étape 1) a un coût en $O(n \times (n-k))$.
L'étape 2) a un coût en $O(n)$.
A p fixé, l'étape 3) a un coût en $O(C_p^n \times p \times (n-k))$, donc en $O(p \times (n-k) \times n^p)$.

Finalement, la complexité de cet algorithme est en $O(e_c \times n^{e_c+1} \times (n-k))$, ce qui est loin d'être terrible, même si c'est déjà beaucoup mieux que la complexité exponentielle de l'algorithme naïf (si k est grand).

Comment souvent, on peut faire mieux en convertissant une partie de la complexité temporelle en complexité spatiale. En effet, on peut une bonne fois pour toutes calculer, pour tout $p \leq e_c$, toutes les sommes possibles de p colonnes de H, les trier par ordre lexicographique et leur associer le mot erreur E correspondant. Le calcul de toutes les sommes dans l'étape 3) est alors remplacé par une recherche dichotomique, de coût $O(p)$.

Algorithme de décodage des codes linéaires, 2^{eme} version :

Le même, mais avec un prétraitement consitant à calculer pour tout $p \leq e_c$ toutes les sommes possibles de p colonnes de H, à les trier par ordre lexicographique et leur associer le mot erreur E correspondant (par exemple à la somme de C_1 et C_3 est associé $E = 10100...$).

Le coût du prétraitement est en $O(e_c \times n^{e_c+1} \times (n-k))$.
L'algorithme a de plus une complexité en espace en $O(e_c \times n^{e_c+1} \times (n-k))$
La complexité temporelle de l'algorithme lui-même est en $O(n \times (n-k))$

Le défaut non réglé est le caractère non optimal de ces algorithmes. Une troisième version, dont la complexité en espace est en $O(2^{n-k})$, règle ce problème.

Algorithme de décodage des codes linéaires, 3^{eme} version :

Précalcul : pour tous les syndromes possibles (les 2^{n-k} mots de longueur $n-k$), on calcule le mot de poids minimal de la classe lattérale associée, ce qui nous donne un tableau de taille (2^{n-k}) (le i^e élément est le mot de poids minimal de la classe lattérale dont le syndrome est i écrit en base 2, s'il existe ; et un symbole d'échec sinon).

Etape 1) : on calcule $S = H \times Z$ et i dont S est l'écriture en base 2

Etape 2) : on selectionne le i^e élément E du tableau

Etape 4) : on corrige par $\phi^{-1}(Z + E)$

Le coût de cet algorithme est en $O(n \times (n-k))$, mais la complexité en espace est très importante, et surtout le précalcul n'est pas réalisable si n est vraiment grand, car il est en $O(n.2^n)$.

Cet algorithme peut être optimisé, en proposant par exemple que s'il y a plusieurs mots de poids minimal dans une classe lattérale, on choisisse celui, s'il existe, pour lequel les erreurs sont le mieux groupées.

Chapitre 4

Codes parfaits

4.1 Plusieurs définitions équivalentes

Définition 4.1 *(Sphères, boules)*

Soit $m \in \{0,1\}^n$ et $k \in \mathbb{N}$.

On appelle ***sphère*** *de centre m et de rayon k, et on note $S(m,k)$, l'ensemble des mots de $\{0,1\}^n$ à distance k de m : $S(m,k) := \{r \in \{0,1\}^n,\ d(r,m) = k\}$.*

On appelle ***boule*** *de centre m et de rayon k, et on note $B(m,k)$, l'ensemble des mots de $\{0,1\}^n$ à distance inférieure ou égale à k de m :*
$B(m,k) := \{r \in \{0,1\}^n,\ d(r,m) \leq k\}$.

Proposition 4.2 *(Cardinal d'une sphère, d'une boule)*
Le nombre d'éléments d'une sphère et d'une boule sont donnés par les formules

$$|S(m,k)| = C_n^k \quad et \quad |B(m,k)| = \sum_{i=0}^{k} C_n^i$$

Remarque : Soit ϕ un code de capacité de correction e_c
▪ Les boules centrées en les mots du code, de rayon e_c, sont disjointes
▪ e_c est par définition le plus grand entier tel que les boules centrées en les mots du code de rayon e_c soient disjointes, ou autrement dit : des boules centrées en les mots du code de rayon k sont disjointes si et seulement si $k \leq e_c$.

Proposition 4.3 *(Inégalité de Hamming)*
Soit ϕ un code de paramètres (k,n), d'image C, de capacité de correction e_c. Alors on a :

$$\sum_{i=0}^{e_c} C_n^i \leq 2^{n-k}$$

*Et on appelle cette propriété l'**inégalité de Hamming**.*

Démonstration : On utilise la remarque précédente. L'union des boules de rayon e_c centrées en des mots de code est incluse dans $\{0,1\}^n$, et cette réunion est disjointe, d'où $\sum_{m\in C} |B(m,e_c)| \leq |\{0,1\}^n|$, enfin, toutes les 2^k boules on le même nombre d'éléments $\sum_{i=0}^{e_c} C_n^i$, ce qui donne $2^k \sum_{i=0}^{e_c} C_n^i \leq 2^n$, d'où l'inégalité cherchée.

Définition 4.4 *(Codes parfaits)*
*Soit ϕ un code de paramètres (k,n), de capacité de correction e_c. On dit que ϕ est un **code parfait** s'il vérifie une des trois caractérisations équivalentes suivantes :*
- *Les boules de rayon e_c de centre les mots du code forment une partition de $\{0,1\}^n$*
- *ϕ vérifie le cas d'égalité de l'inégalité de Hamming $\sum_{i=0}^{e_c} C_n^i = 2^{n-k}$ (on dit que ϕ vérifie **l'égalité de Hamming**)*
- *Pour tout mot $r \in \{0,1\}^n$, il existe un unique mot de code m qui réalise le minimum de $d(r,m)$*

4.2 Caractérisation des codes parfaits linéaires

Comme promis, montrons que les codes parfaits sont rares.
Commençons par un exemple.

Définition 4.5 *(Codes de Hamming)*
Soit $m \leq 2$ un entier.
*Un **code de Hamming** est un code de paramètres $(2^m - m - 1, 2^m - 1)$ dont une matrice de contrôle est obtenu par n'importe quelle énumération en colonne de tous les mots de m bits non nuls.*

Je sais que vous allez rigoler, mais le minitel (vous savez, ce vieux truc qu'utilisaient nos trisaïeuls) code ses données avec un code de Hamming étendu par bit de parité de paramètres $(2^7 - 7 - 1, 2^7 - 1 + 1) = (120, 128)$ (on code 15 octets à l'aide d'un octet supplèmentaire).

Proposition 4.6 *(Distance minimale d'une code de Hamming)*
Un code de Hamming a toujours pour distance minimale 3.

Démonstration : Par définition, la matrice de contrôle d'un tel code n'a pas de colonne nulle donc $d \geq 2$ et n'a pas deux colonnes identiques donc $d \geq 3$; enfin en faisant la somme de deux colonnes contenant exactement un seul 1 on obtient un vecteur contenant exactement deux 1, donc une autre colonne, d'où $d = 3$

Remarque : Un code linéaire de paramètres $(2^m - 1, 2^m - m - 1, 3)$ est nécessairement un code de Hamming. En effet, une matrice de contrôle H d'un tel code doit avoir 2^m lignes et $m - 1$ colonnes, donc contient en colonne des vecteurs de $\mathbb{F}_2^m$. Comme $d > 1$, aucun vecteur colonne de H n'est nul, comme $d > 2$, aucun vecteur colonne n'apparaît deux fois. Finalement, il nous faut placer en colonne $m - 1$ vecteurs non nuls et distincts de $\mathbb{F}_2^m$. Comme il y en a précisément $m - 1$, il faut tous les mettre et le code considéré est un code de Hamming.

Posons-nous maintenant la question de trouver tous les codes parfaits de capacité de correction 1.

Comme les codes parfaits vérifient l'égalité de Hamming, les paramètres d'un code parfait de capacité de correction 1 doivent vérifier $\sum_{i=0}^{1} C_n^i = 2^{n-k}$ c'est à dire $1 + n = 2^{n-k}$. Posons $m := n - k$, on a alors $1 + n = 2^m$, donc $n = 2^m - 1$ et $k = n - m = 2^m - m - 1$.

En définitive un code parfait de capacité de correction 1 a **nécessairement** les paramètres d'un code de Hamming ; et un code parfait linéaire de capacité de correction 1 est toujours un code de Hamming (selon la remarque).

Terminons enfin ce chapitre avec un théorème (à admettre...) qui clos le débat sur les codes parfaits.

Théorème 4.7 *(Codes parfaits linéaires)*
Les seuls codes parfaits linéaires (binaires) sont :
- *les codes de répétition pure* $(1, 2e_c + 1)$
- *les codes de Hamming*
- *le code de Golay* G_{23}

Le code de Golay est un code de paramètres $(12, 23, 7)$ sur lequel nous reviendrons.

(Véridique : il est tombé à l'examen).

Chapitre 5

Généralités sur les codes cycliques

5.1 (R)appels sur les polynômes

Définition 5.1 *(Polynômes à coefficients dans $\mathbb{F}_2$)*

*Un **polynôme** à coefficients dans $\mathbb{F}_2$ est une fonction de la forme $P(X) = a_0 + a_1X + a_2X^2 + ... + a_nX^n$ avec $\forall i \in \{0, ..n\}, a_i \in \mathbb{F}_2$*
*Si $a_n \neq 0$, l'entier n est appelé le **degré** du polynôme P et noté $deg(P)$; les entiers a_i sont appelés les coefficients de P ; par convention le polynôme nul est considéré comme étant de degré $-\infty$.*

Remarque : (identité remarquable des maternelles)
Le fait de travailler dans F_2 nous simplifie grandement la vie.
Par exemple, on a toujours $(a+b)^2 = a^2 + b^2$

Proposition 5.2
Soit P un polynôme à coefficients dans $\mathbb{F}_2$.
Alors $P(X^2) = P(X)^2$

Définition 5.3 *(racines)*
Soit P un polynôme à coefficients dans $\mathbb{F}_2$ et $a \in \mathbb{F}_2$.
*On dit que a est une **racine** de P lorsque $P(a) = 0$*

Exemple 5.4
. *Le polynôme $X^2 + X$ a pour racines 0 et 1*
. *Le polynôme $X^8 + 1$ a pour racine 1 et peut se réécrire $(X+1)^8$*
. *Le polynôme $X^2 + X + 1$ n'a pas de racine dans $\mathbb{F}_2$. Si l'on veut à tout prix qu'il ait des racines, il faudra "imaginer" de nouveaux éléments qui ne sont pas*

dans $\mathbb{F}_2$, de la même façon qu'on construit le corps des nombres complexes en "imaginant" un nouveau nombre $\boldsymbol{i}$ tel que $\boldsymbol{i}^2 = -1$

Définition 5.5 *(factorisation, irreductibilité)*
Soit P un polynôme à coefficients dans $\mathbb{F}_2$.
*. S'il existe deux polynômes P_1 et P_2 tels que $P = P_1P_2$, on dira que P_1 et P_2 **divisent** P, ou encore que P_1 et P_2 sont des **diviseurs** de P. Dans ce cas on a nécessairement $deg(P_1) + deg(P_2) = deg(P)$*
*. S'il existe deux polynômes P_1 et P_2 tels que $deg(P_1) \geq 1$, $deg(P_2) \geq 1$ et $P = P_1P_2$ alors on dit que P_1P_2 est une **factorisation** de P.*
*. Si P n'a pas de factorisation, P est dit **irréductible***

Proposition 5.6 *(division euclidienne)*
Soient P_1 et P_2 deux polynômes à coefficients dans $\mathbb{F}_2$.
Alors il existe deux polynômes à coefficients dans $\mathbb{F}_2$ Q et R, uniques, tels que $P_1 = P_2 \times Q + R$ et $deg(R) < deg(P_2)$
*Q est appelé le **qutient** de la division euclidienne de P_1 par P_2 et R le **reste**.*

5.2 Définitions d'un code cyclique

Attention ! A partir de maintenant, les codes considérés ne seront plus nécessairement systématiques (même si on s'efforcera de repérer et d'étudier ceux qui le sont). De plus, on considèrera désormais les codes "à équivalence près", c'est à dire qu'on identifiera deux codes qui ont la même image, c'est à dire qu'on identifiera un code à son image.

Définition 5.7 *(code cyclique)*
Soit C l'image d'un code de paramètres (k, n).
*Le code est dit **cyclique** si l'ensemble des mots du code est stable par décalage circulaire.*
En d'autres termes, notons $\sigma(m_1m_2...m_{n-2}m_{n-1}m_n) = m_nm_1m_2...m_{n-2}m_{n-1}$.
Un code (d'image) C est cyclique si $\forall m \in C,\ \sigma(m) \in C$.

Exemple 5.8 *(Fort heureusement...)*
La plupart des principaux codes étudiés jusqu'à présent sont cycliques.
Explicitement :
. Les codes de répétition pure (k, n) sont cycliques.
. Le code par bit de parité est cyclique.
. Le code de Hamming systématique $(7, 4)$ du chapitre 3 est cyclique.
. Plus généralement il existe des codes de Hamming $(2^m - m - 1, 2^m - 1)$ cycliques pour tout $m \leq 2$.
. De même il existe des codes simplexes $(k, 2^k - 1, 2^{k-1})$ cycliques pour tout $k \leq 1$.

Par contre...
. *Le code de Hamming étendu* $(4, 8)$ *n'est pas cyclique !*

Définition 5.9 *(code cyclique engendré par un mot)*
Soit $m \in \{0,1\}^n$*. Notons* C *le plus petit sous-espace vectoriel de* $\{0,1\}^n$ *stable par décalage circulaire. Alors* C *est (l'image d') un code cyclique qu'on appelle* ***code cyclique engendré par*** m*.*
Explicitement, le code cyclique engendré par m *est l'ensemble des mots qu'on peut obtenir en faisant des sommes finies de décalés circulaires itérés de* m*, auquel il faut ajouter le mot nul.*

Exemple 5.10
. *Le code cyclique engendré par* 111 *est (équivalent à) notre code de répétition pure* $(1, 3)$
. *Le code cyclique engendré par* 110000000 *est (équivalent à) notre code de bit de parité* $(8, 9)$
. *Le code cyclique engendré par* 1101000 *est (équivalent à) un code de Hamming* $(4, 7)$
. *Le code cyclique engendré par* 1011100 *est (équivalent à) un code Simplexe* $(3, 7)$

Théorème 5.11 *(Théorème fondamental : générateur d'un code cyclique)*
Soit C *(l'image d') un code cyclique* (k, n)*.*
Alors il existe un unique polynôme $g(X) = a_0 + a_1X + ... + a_{n-k}X^{n-k}$ *(*$a_{n-k} = 1$*) tel que :*
. $g(X)$ *est un diviseur de* $X^n + 1$
. C *est le code cyclique engendré par* $m = a_0a_1...a_{n-k}0...0$ *(*$k-1$ *zéros)*
. *Les mots* $m = a_0...a_{n-k}0...00$ *;* $\sigma(m) = 0a_0...a_{n-k}0...0$ *; ... ;* $\sigma^{k-1}(m) = 0...0a_0...a_{n-k}$ *forment une base de* C*. Ce qui signifie qu'une matrice génératrice de* C *est donnée par :*

$$\begin{pmatrix} a_0 & 0 & & 0 \\ a_1 & a_0 & & \vdots \\ \vdots & \vdots & & 0 \\ a_{n-k} & a_{n-k-1} & \dots & 0 \\ 0 & a_{n-k} & & a_0 \\ 0 & 0 & & a_1 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & & a_{n-k} \end{pmatrix}$$

Idée de la démonstration (seulement si vous n'avez pas froid aux yeux) :
. Point 1 : construction de $g(X)$
Considérons un mot m de C non nul contenant un nombre maximal de zéros "à

la fin", et notons ce mot $m = a_0...a_r0...0$, avec $a_r \neq 0$, c'est à dire $a_r = 1$.
Un tel mot est nécessairement unique car la différence de deux tels mots distincts donnerait un mot non nul avec un nombre strictement supérieur de zéros "à la fin".
Enfin, les $n-r$ mots m, $\sigma(m)$, ..., $sigma^{n-r-1}(m)$ sont linéairement indépendant car ils sont tous de longueurs distinctes.
On note $g(X) = a_0 + a_1X + ... + a_rX^r$

. Point 2 : $r = n - k$
Il s'agit maintenant de montrer que les mots m, $\sigma(m)$, ..., $\sigma^{n-r-1}(m)$ engendrent bel et bien C. Une fois ce point acquis, comme ils engendrent un espace de dimension $n - r$ et que C est de dimension k, il s'avèrera de plus que $r = n - k$
Considérons un mot m' dans C. On lui associe sa représentation polynomiale $P(X)$ (voir section suivante). En d'autres termes, on écrit $m' = a'_0a'_1...a'_n$ et on note $P(X) = a'_0 + a'_1X + ... + a'_nX^n$
Opérons la division euclidienne de $P(X)$ par $g(X)$:
$P(X) = (b_0 + b_1X + ... + b_{n-r-1}X^{n-r-1})g(X) + R(X)$ avec $deg(R(X)) < r$
Cette relation se traduit sur les mots : $m' = (b_0m + b_1\sigma(m) + ...) + m''$
Enfin comme m' et m sont des mots de C, et que notre code est linéaire, m'' est aussi un mot de C. Mais comme $deg(R(X)) < r$, m'' a plus de zéros "à la fin" que m, et donc c'est le mot nul.
Finalement on a pu écrire notre mot m' comme somme de décalés circulaires de m. Nos mots m, $\sigma(m)$, ..., $\sigma^{n-r-1}(m)$ engendrent donc bien C, et donc en particulier $r = n - k$

. Point 3 : $g(X)$ divise $X^n + 1$
Considérons maintenant le mot $m' = \sigma^{n-r}(m) = \sigma^k(m) = a_r0...0a_0...a_{r-1}$, et opérons la division euclidienne de sa représentation polynomiale par $g(X)$, comme précédement. Comme la représentation polynomiale de ce mot est :
$X^kg(X) + a_r - X^n = X^kg(X) + (X^n + 1)$, on trouve $X^kg(X) + (X^n + 1) = (b_0 + b_1X + ... + b_{k-1}X^{k-1})g(X) + R(X)$ avec $R(X) = 0$ comme on l'a vu précédemment.
Une fois réécrite convenablement, cette relation devient $X^n + 1 = (b_0 + b_1X + ... + b_{k-1}X^{k-1} + X^k)g(X) = h(X)g(X)$ et on obtient bien le résultat souhaité.

Et donc, finalement, on peut voir un code cyclique (k, n) comme un polynôme à coefficients dans $\mathbb{F}_2$) qui divise le polynôme $X^n + 1$. D'où les conventions suivantes :

Définition 5.12 *(représentation polynomiale)*
. *Soit $m = a_0a_1...a_n$ un mot de longueur n. On appelle* ***représentation polynomiale*** *de m le polynôme $a_0 + a_1X + ... + a_nX^n$. Dorénavant, on identifiera systématiquement un mot avec sa représentation polynomiale.*
. *Soit C un code cyclique (k, n). On appelle* ***polynome générateur*** *de C le polynôme $g(X)$ défini par le théorème fondamental 5.11. Dorénavant, on identifiera un code cyclique avec son polynôme générateur.*

Exemple 5.13
. *Le polynôme générateur du code de répétition pure* $(1, n)$ *est* $1 + X + X^2 + ... + X^{n-1}$
. *Le polynôme générateur du code par bit de parité* $(n-1, n)$ *est* $1 + X$
. *Le polynôme* $1 + X + X^3$ *est le polynôme générateur d'un code de Hamming* $(4, 7)$
. *Le polynôme* $1 + X^2 + X^3$ *est le polynôme générateur... d'un autre code de Hamming* $(4, 7)$ *!*

Finalement, on peut hiérarchiser les codes en fonction de l'occupation mémoire de leur description :
. Les codes en blocs quelconques sont *a priori* décrits par un tableau de taille $O(2^k)$
. Les codes linéaires quelconques sont *a priori* décrits par leur matrice génératrice, de taille $O(n \times k)$
. Les codes cycliques peuvent être décrits par un polynôme de taille $O(n - k)$

5.3 Codes cycliques vs codes systématiques

Donnons-nous un code cyclique par son polynôme générateur $g(X)$. Dans cette section, on montre comment coder et décoder sans avoir à utiliser de matrice génératrice ni retrouver de matrice de contrôle.

Proposition 5.14 *(Algorithme de codage pour un code cyclique)*
Soit C (l'image d') un code cyclique de polynôme générateur $g(X)$
Soit m un mot de source, de représentation polynomiale $P_m(X)$*. Alors le mot image correspondant pour représentation polynomiale* $g(X) \times P_m(X)$*.*

Pour un code cyclique, il existe donc des algorithmes de codage en $O(n \times lg(n))$.

Remarque : Soit C (l'image d') un code cyclique (k, n). Alors un mot de longueur n est un mot de code si et seulement si sa représentation polynômiale est un multiple de $g(X)$, ce qui revient à dire que le reste de la division euclidienne de sa représentation polynomiale par $g(X)$ est nul.

Proposition 5.15 *(Matrice de contrôle d'un code systématique)*
Soit C un code cyclique de polynôme générateur $g(X) = a_0 + a_1X + ... + a_{n-k}X^{n-k}$.

Alors (théorème fondamental) une matrice génératice de ce code est :

$$\begin{pmatrix} a_0 & 0 & & 0 \\ a_1 & a_0 & & \vdots \\ \vdots & \vdots & & 0 \\ a_{n-k} & a_{n-k-1} & \dots & 0 \\ 0 & a_{n-k} & & a_0 \\ 0 & 0 & & a_1 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & & a_{n-k} \end{pmatrix}$$

Et une matrice de contrôle associée à cette matrice génératrice est :

$$\begin{pmatrix} b_k & b_{k-1} & \dots & b_0 & 0 & \dots & 0 \\ 0 & b_k & b_{k-1} & \dots & b_0 & \dots & 0 \\ & & & \vdots & & & \\ 0 & \dots & 0 & b_k & b_{k-1} & \dots & b_0 \end{pmatrix}$$

Où $h(X) = b_0 + b_1X + ... + b_kX^k$ est le polynôme défini par $h(X) = \dfrac{X^n + 1}{g(X)}$.

On peut donc décoder un code cyclique aussi bien qu'un code linéaire systématique. En fait, dans le cas d'un code cyclique quelconque, il existe des algorithmes de décodage meilleurs que celui du chapitre 4 : un exemple assez peu évolué d'un tel algorithme est présenté dans le td 7.

Chapitre 6

Codes BCH

6.1 Détermination des codes cycliques de longueur impaire

Si $n = 2p$ est pair, l'identité remarquable des maternelles donne $X^n + 1 = (X^p+1)^2$; il y a alors deux types de diviseurs de X^n+1 : les diviseurs de X^p+1, et les produits de deux de ces précédents. Pour cette raison (et d'autres), l'étude des diviseurs de $X^n + 1$ devient alors plus compliquée. Ainsi, dans cette partie, **on suppose désormais que n est impair**.

La question qu'on se pose maintenant est la suivante : pour n (impair) donné, quels sont les codes cycliques de longueur n ? Comme le montre l'exemple, il existe toujours un code cyclique $(1, n)$ (le code de répétition pure) et un code cyclique $(n-1, n)$ (le code par bit de parité). Ces deux-là ne sont pas transcendants. L'exemple du code de Hamming montre qu'il peut exister d'autres codes cycliques qui sont de "bons" codes, du moins pour $n = 7$. La question est donc de généraliser au cas n impair quelconque, en espérant une réponse positive. Or, la recherche d'un code cyclique de longueur n n'est autre, d'après le théorème fondamental, que la recherche de diviseurs de $X^n + 1$.

Soit n entier impair.
Pour déterminer tous les diviseurs de $X^n + 1$, il suffit de trouver la factorisation complête de $X^n + 1$, c'est à dire d'écrire $X^n + 1 = Q_1Q_2...Q_s$ avec les Q_i irréductibles. Comme on l'a vu, la seule racine de $X^n + 1$ dans $\mathbb{F}_2$ est 1, et le polynôme $X^n + 1$ se factorise $X^n + 1 = (X + 1)(X^{n-1} + X^{n-2} + ... + X + 1)$, mais parfois le polynôme $X^{n-1} + ... + X + 1$ est lui-même factoriable bien qu'il n'ait pas de racines.
Au passage, remarquons que si $X^{n-1} + ... + X + 1$ avait des racines, la situation infiniment moins complexe ; mais $X^{n-1} + ... + X + 1$ n'a pas de racine. Pour remédier à ce soucis, on va "imaginer" de telles racines.

Définition 6.1 *(racine primitive n^e de l'unité)*
On appelle ***racine primitive*** n^e ***de l'unité*** *un nouveau nombre, α, tel que $\alpha \neq 1$, $\alpha^2 \neq 1$, ..., $\alpha^{n-1} \neq 1$ et $\alpha^n = 1$.*
Ce nouveau nombre n'est ni 0 ni 1, il n'appartient donc pas à $\mathbb{F}_2$.

Remarque : Les constatations suivantes s'imposent :
. α est une racine (imaginée) de $X^n + 1$
. Mais alors, α^2, α^3, ..., $\alpha^n = 1$ sont aussi des racines de $X^n + 1$!

Proposition 6.2 *(factorisation de $X^n + 1$ dans $\mathbb{F}_2(\alpha)[X]$)*
D'après la remarque qui précède, $X^n+1 = (X+1)(X+\alpha)(X+\alpha^2)...(X+\alpha^{n-1})$

Finalement, on a bien une factorisation extème de $X^n + 1$, mais le problème c'est que les coefficients des polynômes obtenus ne sont pas dans $\mathbb{F}_2$. Notre nouveau problème est donc : comment regrouper les $(X + \alpha^i)$ pour obtenir des polynômes qui eux, seront dans $\mathbb{F}_2[X]$?

Théorème 6.3
Soit $P(x)$ un polynôme à coefficients dans $\mathbb{F}_2(\alpha)$, avec α une racine primitive n^e de l'unité.
Si $P(X^2) = P(X)^2$, alors $P(X)$ est en réalité à coefficients dans $\mathbb{F}_2$!

D'où le corollaire suivant :

Théorème 6.4
Soit $P(x) = \prod_{i\in\Sigma}(X - \alpha^i)$, avec α une racine primitive n^e de l'unité.
Alors $P(X) \in \mathbb{F}_2[X]$ si et seulement si Σ est stable par multiplication par 2 modulo n.

Dans un soucis de simplification du vocabulaire, on appelera **partie stable** une partie de $\{0, 1, ..., n\}$ stable par multiplication par 2 modulo n.

Définition 6.5 *(Classes cyclotomiques)*
On appelle ***classe cyclotomique*** *de n une partie de $\{0, .., n-1\}$ stable par multiplication par 2 et minimale pour l'inclusion ; donc un ensemble de la forme $\{j, 2j, 4j, ..., 2^{s-1}j\}$ avec $2^s j \equiv j \ mod \ n$.*

Explicitement : la classe cyclotomique $\{j, 2j, 4j, ..., 2^{s-1}j\}$ (avec $2^s j \equiv j \ mod \ n$) sera appelée **classe cyclotomique engendrée par** j.

Les classes cyclotomiques permettent de factoriser complêtement $X^n + 1$ dans $\mathbb{F}_2[X]$; en effet $X^n + 1$ est le produit des polynômes obtenus en simplifiant $\prod_{i\in\Sigma}(X - \alpha^i)$ pour chaque classe cyclotomique Σ. Le problème qui reste est maintenant d'effectuer cette simplification. Pour cela on peut :
. disposer d'une table d'addition dans $\mathbb{F}_2(\alpha)$ qu'un mathématicien au grand coeur a mis à notre disposition et l'utiliser
. tester tous les diviseurs irréductibles de degré convenable ; en effet le degré d'un facteur irréductible de $X^n + 1$ est le nombre d'élément de la classe cyclotomique correspondante.
. bibouiller.

Exemple 6.6
. $X^7 + 1 = (X + 1)(X^3 + X^2 + 1)(X^3 + X + 1)$ *et donc les codes cycliques de longueur* 7 *sont* $(1, 7)$, $(3, 7)$, $(4, 7)$ *et* $(6, 7)$.
. $X^9 + 1 = (X + 1)(X^2 + X + 1)(X^6 + X^3 + 1)$ *et donc les codes cycliques de longueur* 9 *sont* $(1, 9)$, $(2, 9)$, $(3, 9)$, $(6, 9)$, $(7, 9)$ *et* $(8, 9)$.
. $X^1 + 1 = (X + 1)(X^{10} + X^9 + ... + X + 1)$ *et donc les seuls codes cycliques de longueur* 11 *sont triviaux.*

6.2 Les codes BCH primitifs stricts

Si vous tenez à briller en société quand votre belle-mère vous invitera à dîner, sachez que les codes BCH tiennent leur nom (très originalement) des initiales de leurs inventeurs : Bose, Chaudhuri, Hocquenghem. Dans le cas contraire vous pouvez oublier ce détail tout de suite.

Dorénavant, on se donne un entier r et on pose $n = 2^r - 1$ (en particulier n est impair). On se donne aussi une racine primitive n^e de l'unité qu'on note α.

Commençons avec un théorème qui tue tout, à savoir **enfin une minoration de la distance minimale d'un code** !

Théorème 6.7 *(Théorème BCH)*
Notons C le code cyclique de associé à la partie stable $\Sigma = \{i_1, ..., i_{n-k}\}$ (c'est à dire de polynôme générateur $\prod_{i\in\Sigma}(X - \alpha^i)$).
*S'il existe des entiers a et s tels que Σ contienne $a + 1$,$a + 2$,...,$a + s$, **alors la distance minimale de C est supérieure ou égale à $s+1$.***

On dispose enfin d'un moyen effectif de construire des codes (cycliques) de distance minimale aussi grande que souhaitée. On continuera à se limiter à $n = 2^r - 1$ et on prendra $a = 0$ (d'où les qualificatifs primitif et strict).

Définition 6.8 *(Code BCH)*
Soit t un nombre entier et $\delta = 2t + 1$
On appelle ***code BCH primitif strict associé à*** *δ (ou plus simplement le code BCH associé à δ) le code C associé à la plus petite partie stable contenant $1, 2, ..., \delta$. L'entier δ vérifie $\delta \leq d$ (inégalité parfois stricte) et on l'appelle la* ***distance assignée*** *à C.*

Exemple 6.9 *(Codes BCH de longueur 15)*
Prenons $r = 4$, donc $n = 15$.
Les classes cyclotomiques sont $\{0\}$, $\{1, 2, 4, 8\}$, $\{3, 6, 9, 12\}$, $\{5, 10\}$ et $\{7, 11, 13, 14\}$.
On obtient donc quatre codes BCH qui sont les suivants :

t	Σ	k	δ	d	$g(X)$
1	$\{1, 2, 4, 8\}$	11	3	3	$g_1(X)$
2	$\{1, 2, 3, 4, 6, 8, 9, 12\}$	7	5	5	$g_1(X)g_3(X)$
3	$\{1, 2, 3, 4, 5, 6, 8, 9, 10, 12\}$	5	7	7	$g_1(X)g_3(X)g_5(X)$
4 à 7	$\{1, ..., 14\}$	1	9 à 15	15	$1 + ... + X^{14}$

Avec $g_1(X) = 1 + X + X^4$
$g_3(X) = 1 + X + X^2 + X^3 + X^4$ *donc* $g_1(X)g_3(X) = 1 + X^4 + X^6 + X^7 + X^8$
$g_5(X) = 1+X+X^2$ *donc* $g_1(X)g_3(X)g_5(X) = 1+X+X^2+X^4+X^5+X^8+X^{10}$

Au passage : le premier est un code de Hamming, le dernier le code de répétition pure.

6.3 Un algorithme de décodage des codes BCH

Pour les codes BCH, on dispose d'algorithmes de décodage **algébriques**. En général, ces algorithmes permettent la correction de t erreurs ou moins. Dans le cas où $\delta = d$, c'est donc un algorithme de décodage en deçà de la capacité de correction. Si $\delta < d$, on renonce même à corriger des mots contenant moins d'erreurs que la capacité de correction. La contrepartie est que ces algorithmes sont très efficaces (de plus, t est donné par construction alors que e_c est en général difficile à déterminer). Pour conclure ce cours, présentons un tel algorithme de décodage, dû à Peterson, Gorenstein et Zierler.

Notons $P(X)$ la représentation polynômiale du mot reçu.

Etape 1) : on calcule $S_1 = P(\alpha)$, $S_2 = P(\alpha 2)$, ..., $S_{2t} = P(\alpha^{2t})$

Etape 2) : calcul du nombre d'erreurs : on note ν le rang de la matrice $\begin{pmatrix} S_1 & \dots & S_t \\ S_2 & \dots & S_{t+1} \\ & \vdots & \\ S_t & \dots & S_{2t-1} \end{pmatrix}$.

Etape 3) : On résout le système $\begin{pmatrix} S_1 & \dots & S_\nu \\ S_2 & \dots & S_{\nu+1} \\ & \vdots & \\ S_\nu & \dots & S_{2\nu-1} \end{pmatrix} \times \begin{pmatrix} \varsigma_\nu \\ \varsigma_{\nu-1} \\ \vdots \\ \varsigma_1 \end{pmatrix} = \begin{pmatrix} S_{\nu+1} \\ S_{\nu+2} \\ \vdots \\ S_{2\nu} \end{pmatrix}$.

Etape 4) : On détermine les ν racines α^{-i_1},...,α^{-i_ν} du polynôme $\varsigma_\nu X^\nu + \ldots + \varsigma_1 X + 1$.

Etape 5) : Les erreurs ont eu lieu en postition i_1, ..., i_ν, on inverse ces bits.

Prenons deux cas simples.

Premier cas : $t = 1$ (c'est à dire que le code considéré est un code de Hamming).
Dans ce cas l'algorithme est très dégéné :
Notons $P(X)$ la représentation polynômiale du mot reçu.
Etape 1) : on calcule $S = P(\alpha)$
Etape 2) : correction. Si $S = 0$, pas d'erreur.
Sinon, on peut écrire $S = \alpha^i$. L'erreur a eu lieu en position i, on inverse ce bit.

Second cas : $t = 2$.
Dans ce cas l'algorithme est le suivant :
Etape 1) : on calcule $S_1 = P(\alpha)$, $S_2 = P(\alpha^2)$, $S_3 = P(\alpha^3)$, $S_4 = P(\alpha^4)$.
Etape 2) : calcul du nombre d'erreurs.
▪ Si $S_1 = 0$, pas d'erreur, l'algorithme s'arrête.
▪ Si $S_1 \times S_3 = S_2^2$, une erreur. On peut noter $S_1 = \alpha^i$. On inverse le bit en position i et la correction est terminée, l'algorithme s'arrête.
▪ Sinon on passe à l'étape suivante.
Etape 3) : On résout le système $\begin{cases} S_1 \times \varsigma_2 + S_2 \times \varsigma_1 = S_3 \\ S_2 \times \varsigma_2 + S_3 \times \varsigma_1 = S_4 \end{cases}$.
Etape 4) : On détrermine les 2 racines α^{-i} et α^{-j} du polynôme $\varsigma_2 X 2 + \varsigma_1 X + 1$.
Etape 5) : Les erreurs ont eu lieu en postition i et j, on inverse ces bits.

Prenons deux exemples très simples avec $n = 7$.

Un mathématicien généreux a mis à notre disposition cette table d'addition de $\mathbb{F}_2(\alpha)$ pour $n = 7$:

$+$	1	α	α^2	α^3	α^4	α^5	α^6
1	0	α^3	α^6	α	α^5	α^4	α^2
α	α^3	0	α^4	1	α^2	α^6	α^5
α^2	α^6	α^4	0	α^5	α	α^3	1
α^3	α	1	α^5	0	α^6	α^2	α^4
α^4	α^5	α^2	α	α^6	0	1	α^3
α^5	α^4	α^6	α^3	α^2	1	0	α
α^6	α^2	α^5	1	α^4	α^3	α	0

Les deux seuls codes BCH avec $n = 7$ sont le code de Hamming $(4, 7)$ $(t = 1)$ et le code de répétition pure $(1, 7)$ $(t = 3)$

Prenons l'exemple du code de Hamming $(4, 7)$. Son polynôme générateur est $g(X) = 1 + X + X^3$
On reçoit le mot $1 + X + X^2$.
Etape 1 : $S = 1 + \alpha + \alpha^2 = \alpha^3 + \alpha^2 = \alpha^5$ (en utilisant deux fois la table).
Etape 2 : L'erreur a eu lieu en position 5, le mot envoyé était $1+X+X^2+X^5 = (1 + X + X^3)(1 + X^2)$, on décode donc en 1010.

Prenons l'exemple du code de répétition pure $(1, 7)$. Au passage : il est clair que dans ce cas l'algorithe de décodage naïf serait plus efficace ! Son polynôme générateur est $g(X) = 1 + X + X^2 + X^3 + X^4 + X^5 + X^6$
On reçoit le mot $1 + X$.

Etape 1 :
$S_1 = \alpha + 1 = \alpha^3$,
$S_2 = \alpha^2 + 1 = \alpha^6$,
$S_3 = \alpha^3 + 1 = \alpha$,
$S_4 = \alpha^4 + 1 = \alpha^5$,
$S_5 = \alpha^5 + 1 = \alpha^4$,
$S_6 = \alpha^6 + 1 = \alpha^2$,

Etape 2 :
- $\begin{pmatrix} \alpha^3 \end{pmatrix}$ est inversible donc il y a des erreurs.
- $\begin{pmatrix} \alpha^3 & \alpha^6 \\ \alpha^6 & \alpha \end{pmatrix}$ est inversible donc $\nu \leq 2$
- $\begin{pmatrix} \alpha^3 & \alpha^6 & \alpha \\ \alpha^6 & \alpha & \alpha^5 \\ \alpha & \alpha^5 & \alpha^4 \end{pmatrix}$ n'est pas inversible donc $\nu = 2$

Etape 3 : On utilise la formule $\begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \frac{1}{ad+bc} \begin{pmatrix} d & b \\ c & a \end{pmatrix}$, on trouve $\begin{pmatrix} \varsigma_2 \\ \varsigma_1 \end{pmatrix} = \begin{pmatrix} \alpha & \alpha^6 \\ \alpha^6 & \alpha^3 \end{pmatrix} \times \begin{pmatrix} \alpha \\ \alpha^5 \end{pmatrix} = \begin{pmatrix} \alpha \\ \alpha^3 \end{pmatrix}$

Etape 4 : On cherche les racines de $\alpha X^2 + \alpha^3 X + 1$. On obtient $1 = \alpha^0$ et $\alpha^6 = \alpha^{-1}$; les erreurs sont en positions 0 et 1, le mot envoyé est nul, le mot de source est 0.