Codes détecteurs et correcteurs d'erreurs

G. Montcouquiol

IUT Orsay

2006-2007

Un problème très concret

Contexte : omniprésence contemporaine de la transmission de données numériques (internet, téléphones portables...)

Problème : les communications ne sont jamais parfaites. Quel que soit le canal utilisé, des erreurs de transmission se produisent inévitablement.

Conséquences : en informatique, une erreur d'un seul bit peut avoir des conséquences sérieuses si elle a lieu dans un exécutable : instruction erronée, écriture au mauvais emplacement en mémoire...

Des chiffres : le taux d'erreur varie énormément suivant les canaux. En informatique, il est généralement compris entre 10^{-9} (un bit sur un milliard erroné) et 10^{-4} (un bit sur dix mille erroné).

Exemple : avec un taux d'erreur de 10^{-6} et une connexion à 1 Mo/s, en moyenne 8 bits erronés sont transmis chaque seconde...

Des solutions

On peut envisager deux types de solutions pour s'assurer de l'intégrité des données transmises.

- On peut agir sur le canal de transmission. Objectif : rendre le taux d'erreur négligeable. Implique de remplacer les infra-structures existantes → impraticable. De toutes façons le taux d'erreur nul n'existe pas.
- On peut agir sur le message transmis. Objectif : être capable de détecter si des erreurs de transmission ont eu lieu, et éventuellement les corriger. C'est le principe des **codes correcteurs d'erreurs**.

Comment détecter et/ou corriger des erreurs ?

Un exemple simple :
Supposons qu'on veuille me transmettre un numéro de téléphone. Mon correspondant à deux possibilités :

1. Il me transmet 0169336000.
   S'il y a des erreurs de transmission, par exemple si je reçois 0167336010, je ne peux pas les détecter.
2. Il me transmet `zero un soixante-neuf trente-trois soixante zero zero`
   S'il y a des erreurs de transmission, par exemple si je reçois `zero an sogxante-seuf trepte-troisksoixanqe zoro zerb`, je suis capable de corriger les erreurs et de retrouver le numéro.

Dans le premier cas, l'information est la plus concise possible.
Dans le deuxième cas au contraire, le message contient plus d'informations que nécessaire. C'est cette **redondance** qui permet la détection et la correction d'erreurs.

Exemples de redondance

L'utilisation de redondance pour améliorer une communication n'est pas vraiment une idée nouvelle... Le type le plus simple de redondance est la répétition pure et simple du message.

- Comportement habituel au téléphone : répétition des numéros, épellation d'un nom ("I comme Irma, U comme Ursule, T comme Thérèse"...)
- Alphabet radio international : chaque lettre de l'alphabet est remplacée par un mot (Alpha Bravo Charlie Delta Echo ...)
- L'orthographe traditionnelle comporte des redondances → moins sensible aux erreurs qu'une écriture phonétique (ou type SMS).

Schéma général de communication

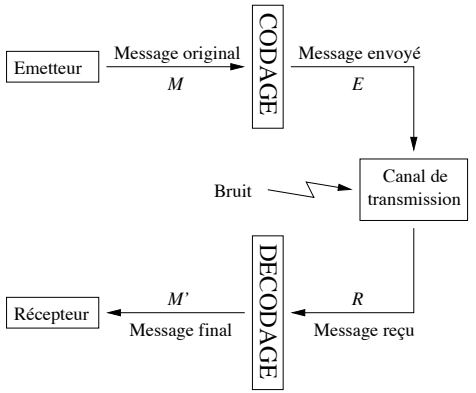


Schéma général de communication

- Le message à transmettre M est d'abord **codé** en un message E. Le codage introduit un supplément d'information (redondances).
- Le message E est envoyé sur le canal de transmission.
- A cause de l'imperfection du canal, le message reçu R n'est pas forcément identique au message envoyé E : il peut contenir des erreurs.
- Le message reçu R est ensuite **décodé** : des erreurs éventuelles peuvent être détectées et corrigées.
- Le message décodé M' est identique au message d'origine M si il n'y a pas eu trop d'erreurs dans la transmission.

Exemples de transmission

Ce schéma de communication est très général.

- transmissions inter-ordinateurs : liaisons ethernet, wifi, internet...
- transmissions intra-ordinateurs : disque dur ↔ mémoire vive ↔ processeur
- communications hertziennes (ondes électro-magnétiques) : radio, téléphones portables, satellittes, sondes spatiales...
- mais aussi la transmission de données via des supports : disquettes, CD, DVD, lettres, livres...

Définitions

Définition
Un **alphabet** *A est un ensemble fini, dont les éléments sont appelés* **symboles**.

Définition
Un **mot** *de longueur n est un élément de A^n : c'est une suite de n symboles de A.*
Un **message** *est une suite de symboles de A de longueur quelconque. On note A^* l'ensemble des messages.*

Si w est un élément de A^n (ou de A^*), on note w_i le i-ème symbole de w.
On utilisera aussi la notation $w = w_1 w_2 \dots w_n$, où les w_i sont des symboles de A.
Le plus fréquemment on aura $A = \{0, 1\}$ (on parlera alors de mots et de codes **binaires**).

Rappels sur les mots

Concaténation :
Soient $w_1 \in A^n$ un mot de longueur n et $w_2 \in A^p$ un mot de longueur p.
Le **concaténé** de w_1 et de w_2, noté $w_1.w_2$, est le mot de A^{n+p} obtenu en "écrivant à la suite" w_1 et w_2.
Exemple : $azert.yuiop = azertyuiop$

Préfixe :
Un mot $u \in A^p$ est un **préfixe** d'un mot $w \in A^n$ si u est "au début" de w, c'est-à-dire s'il existe $v \in A^{n-p}$ tel que $w = u.v$

Suffixe :
Un mot $u \in A^s$ est un **suffixe** d'un mot $w \in A^n$ si u est "à la fin" de w, c'est-à-dire s'il existe $v \in A^{n-s}$ tel que $w = v.u$

Codages

Définition
Un **codage** *est une application injective $\phi : A^* \to B^*$.*

C'est une règle spécifiant comment transformer un *message d'origine M* en un nouveau *message codé $M' = \phi(M)$*.
(Les alphabets du message d'origine et du message codé seront le plus souvent les mêmes.)

Etant donné un message $M' \in B^*$, le **décodage** consiste à rechercher un message $M \in A^*$ tel que $\phi(M) = M'$.
S'il existe, un tel message M doit être unique : c'est pour cela que l'application ϕ doit être **injective**.

Utilité

Les buts d'un codage peuvent être multiples :

- Détection et correction d'erreurs : le codage doit introduire de la redondance. C'est l'objet de ce cours.
- Compression (abréviations, code de Huffman...).
- Changement d'alphabet (code Morse, ASCII...).
- Cryptographie : le décodage doit être très difficile si on ne connaît pas ϕ.

Codage par blocs

Dans la pratique des codes correcteurs, on utilise principalement des **codages par blocs**.

- Le message à transmettre est découpé en **blocs** (mots) de taille p fixée.
- Chacun des blocs d'origine de taille p est codé séparément. Tous les blocs codés doivent avoir la même taille n.
- Les blocs codés sont transmis successivement.

Exemples : code Ascii ($p = 1$, $n = 7$ ou 8), codes vus en TD ($p = 2$ puis 1, $n = 3$).
Contre-exemple : code Morse, alphabet radio

Codage par blocs

L'étude d'un codage par blocs se restreint à l'étude du codage de mots de longueur p par des mots de longueur n.

Un tel codage est entièrement décrit par l'**application de codage**

$$\phi : A^p \to B^n$$

Remarque : L'application ϕ doit être injective, donc si $B = A$ on a nécessairement $n \geq p$.

Dans la suite de ce cours tous les codages seront des codages par blocs (sauf mention du contraire).

Terminologie

Soit $\phi : A^p \to B^n$ une application de codage.

Définition

On appelle **mot de code** *tout élément de l'image de ϕ, i.e tout mot de B^n de la forme $\phi(w)$, $w \in A^p$.*

On appelle **code** *l'ensemble des mots de code : $C = \{\phi(w),\ w \in A^p\}$.*
C'est l'image par ϕ de tous les mots de A^p.

Comme ϕ est injective, il y a autant de mots de code que de mots dans A^p. Un code est donc juste un sous-ensemble à $card(A)^p$ éléments de B^n.

Attention : ne pas confondre **code** (sous-ensemble de B^n) et **codage** (donné par l'application ϕ).

Dans le cas où $A = B$, on parle de **code de taille** $(\mathbf{p}, \mathbf{n})$
Dans la suite de ce cours A et B seront toujours les mêmes (sauf mention du contraire).

Codage systématique

Définition

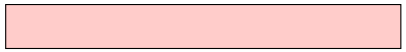
Un codage $\phi : A^p \to A^n$ est **systématique** *si le mot à coder se retrouve en tête du mot codé :*

$$\forall w \in A^p,\ w \text{ est un préfixe de } \phi(w)$$

Un codage systématique consiste juste à rajouter $k = n - p$ **symboles** (ou bits) **de contrôle** à la fin du mot à coder.
Les p premiers symboles du mot codé portent l'information.
Les k derniers symboles du mot codé portent la redondance.

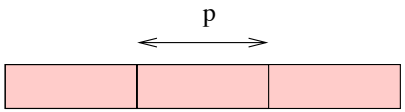
- Intérêt : décodage immédiat
- Quasiment tous les codes peuvent être rendus systématiques

Codage systématique - illustration



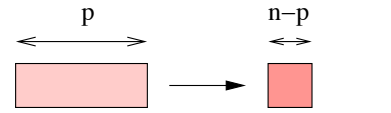
message à transmettre

Codage systématique - illustration



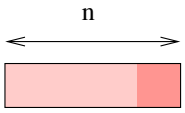
découpage en blocs

Codage systématique - illustration



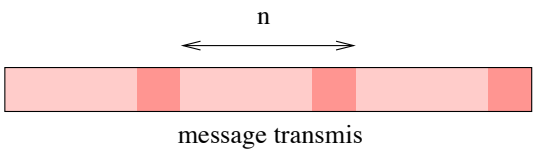
mot d'origine bits de controle

Codage systématique - illustration



mot codé

Codage systématique - illustration



Décodage et détection d'erreurs

Supposons que l'on ait reçu un mot w. Comment savoir si la transmission s'est bien déroulée ?

1. Si w n'est pas un mot du code C, on sait qu'il y a eu des erreurs dans la transmission et que le mot reçu est erroné.
2. Si w est un mot du code, on suppose qu'il n'y a pas eu d'erreurs de transmission et que le mot reçu est correct. On peut alors décoder w.

   En fait, deux cas peuvent s'être produits :
   - il n'y a pas eu d'erreur de transmission
   - il y a eu suffisamment d'erreurs de transmission, et bien positionnées, transformant le mot de code envoyé en un autre mot de code

   On rejette la deuxième possibilité comme étant la moins probable.

Décodage et détection d'erreurs

Il est bien sûr impossible de garantir avec certitude l'absence d'erreurs dans la transmission.
On peut cependant réduire autant que voulu le risque d'erreurs accidentelles, au prix d'une augmentation considérable de la longueur du message envoyé...

Codage par répétition simple

Vu en TD

C'est un des codes détecteurs d'erreurs les plus simples.

Codage de taille $(1, 3)$ sur l'alphabet $A = \{0, 1\}$.
Application de codage : $0 \mapsto 000$, $1 \mapsto 111$

C'est un codage systématique.

Les mots du code sont 000 et 111.

On a vu en TD que ce code peut détecter deux erreurs (et corriger une erreur).

Codage par bit de parité

Vu en TD

C'est un des codes détecteurs d'erreurs les plus simples, et un des plus utilisés.

Codage de taille $(n-1, n)$ sur l'alphabet $A = \{0, 1\}$
Le codage est systématique : on rajoute un bit de contrôle (**bit de parité**), de telle sorte que le nombre de 1 du mot codé soit pair.

Si le nombre de 1 du mot à coder est pair $\longrightarrow$ bit de parité à 0
Si le nombre de 1 du mot à coder est impair $\longrightarrow$ bit de parité à 1

Exemples : 0110 donne 01100, 1101 donne 11011.

Mots du code : ensemble des $w \in A^n$ dont le nombre de 1 est pair.

On a vu en TD que ce code détecte un nombre impair d'erreurs (mais ne peut pas en corriger).

Hypothèses sur les erreurs

Pour simplifier, on va supposer que :

1. les erreurs de transmission ne vont pas rajouter ou supprimer de symboles, seulement les modifier,
2. chaque symbole a la même probabilité p d'être modifié pendant la transmission,
3. si l'alphabet est de taille s, les $s-1$ possibilités de modification d'un symbole sont équiprobables,
4. les probabilités d'erreur sur chaque symbole transmis sont indépendantes.

Ces hypothèses ne modélisent pas l'ensemble des situations. On va les reprendre une par une.

Hypothèses réalistes ?

1. les erreurs de transmission ne vont pas rajouter ou supprimer de symboles, seulement les modifier

Ce n'est pas le cas si le canal est un texte écrit. Erreurs de lecture classique : ri $\longrightarrow$ n, rn $\longrightarrow$ m...

Dans le cadre des transmissions informatiques, ce sont des erreurs peu fréquentes et très faciles à détecter (problèmes de synchronisation d'horloges ou de longueur de trames).

Hypothèses réalistes ?

2. chaque symbole a la même probabilité p d'être modifié pendant la transmission,
3. si l'alphabet est de taille s, les $s-1$ possibilités de modification d'un symbole sont équiprobables

Ce n'est pas le cas si le canal est un texte écrit. Certaines confusions sont plus fréquentes : O et 0, I et l ...

C'est le cas pour une transmission informatique (en binaire, le point 3 est sans objet).

Hypothèses réalistes ?

4. les probabilités d'erreur sur chaque symbole transmis sont indépendantes.

C'est le point le plus critiquable. Dans une transmission numérique, on distingue deux types d'erreurs :

- des erreurs apparaissant de façon **aléatoire**, liées à un bruit de fond électro-magnétique. Ce sont celles que l'hypothèse modélise.
- des erreurs apparaissant **en rafale**, dont les causes peuvent être multiples. Se manifestent par beaucoup de symboles erronés à très peu d'écart → mettent en défaut les codages par blocs classiques. On procède alors à un **entrelacement** du message avant de faire le codage par blocs

Hypothèses sur les erreurs

Conclusion :

Les hypothèses faites sur les erreurs modélisent convenablement un bruit de fond aléatoire lors de la transmission d'un signal binaire.

On verra éventuellement les techniques d'entrelacement à la fin du cours.

Nombre d'erreurs - loi binômiale

Avec les hypothèses que l'on a faites, le nombre d'erreurs de transmission (noté Z) suit une **loi binômiale** $B(n,p)$, où n est la taille d'un bloc et p le **taux d'erreurs**.

$$P(Z=k) = C_n^k\, p^k (1-p)^{n-k}$$

On supposera toujours que p est suffisament faible et n suffisament petit, de telle sorte que le cas le plus probable est qu'il n'y ait pas (ou peu) d'erreurs de transmission :

$$P(Z=0) > P(Z>0)$$

Applications

On utilise un canal sur lequel le taux d'erreurs est de 10^{-2} pour envoyer des messages de 64 bits.

Probabilité d'avoir :

- aucune erreur de transmission : $P(Z{=}0) = (1-p)^n = 0{,}99^{64} \simeq 0{,}526$
- une erreur de transmission : $P(Z{=}1) = np(1-p)^{n-1} \simeq 0{,}340$
- deux erreurs de transmission : $P(Z{=}2) = \frac{n(n-1)}{2} p^2 (1-p)^{n-2} \simeq 0{,}108$
- trois erreurs de transmission : $P(Z{=}3) = C_n^3 p^3 (1-p)^{n-3} \simeq 0{,}023$
- quatre erreurs de transmission : $P(Z{=}4) = C_n^4 p^4 (1-p)^{n-4} \simeq 0{,}003$
- plus de quatre erreurs : $P(Z{>}4) < 0{,}0005$

Distance entre deux mots

Définition

Soit A un alphabet, et soient w et w' deux mots sur A de longueurs p. On appelle **distance de Hamming** *(ou simplement distance) entre w et w', et on note $d(w, w')$, le nombre de positions où w et w' ont des symboles différents :*

$$d(w, w') = card\{i,\ w_i \neq w'_i\}$$

Alternativement, c'est aussi :

- le nombre de symboles à modifier pour passer de w à w'
- le nombre d'erreurs nécessaires pour confondre w et w'

Exemples : $d(1101, 1010) = 1$, $d(IUT, DUT) = 1$, $d(toto, titi) = 2$.

Propriétés

On vérifie que la distance de Hamming est une **distance** sur A^p, c'est-à-dire qu'elle vérifie les trois propriétés suivantes :

positivité : $d(x, y) \geq 0$ quels que soient les éléments x et y de A^p, et $d(x, y) = 0$ si et seulement si $x = y$

symétrie : $d(x, y) = d(y, x)$ quels que soient les éléments x et y de A^p

inégalité triangulaire : $d(x, y) \leq d(x, z) + d(z, y)$ quels que soient les éléments x, y et z de A^p

Décodage et correction d'erreurs

Supposons que l'on ait reçu un mot w. Comment retrouver le message transmis ?

1. Si w est un mot du code, on suppose qu'il n'y a pas eu d'erreurs de transmission (cas le plus probable) et que le mot reçu est correct. On peut alors décoder w.
2. Si w n'est pas un mot du code C, on sait qu'il y a eu des erreurs dans la transmission et que le mot reçu est erroné.
   On recherche alors le mot du code **le plus proche** de w pour la distance de Hamming.
   - Si le mot de code le plus proche de w est unique, on remplace w par ce mot de code v : on a **corrigé** les erreurs de transmission.
   - Sinon, on ne peut pas choisir parmi les mots de code les plus proches : la correction d'erreur échoue.

Distance de Hamming d'un code

Définition

On appelle **distance de Hamming** *(ou simplement distance) d'un code C, et on note $d(C)$, la plus petite distance entre deux mots distincts de C :*

$$d(C) = \min\{d(x, y)\ ;\ x \in C,\ y \in C,\ x \neq y\}$$

C'est donc le nombre minimum d'erreurs permettant de transformer un mot du code en un autre mot du code.

Plus la distance de Hamming du code est grande, plus les mots du code sont "dispersés" dans A^n.

Exemples

Codage par répétition

$C = \{000, 111\}$
La distance de Hamming du code est donc $d(000, 111) = 3$.

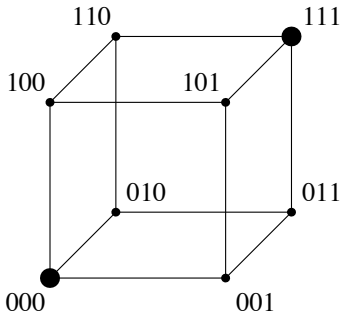
Codage par bit de parité

$C = \{000, 011, 101, 110\}$
La distance de Hamming du code est 2.

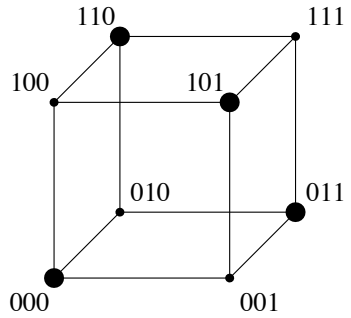
Exemples

On peut représenter les éléments $(\mathbb{Z}/2\mathbb{Z})^3$ par les sommets d'un cube ; les arêtes du cube relient les mots à distance 1 l'un de l'autre.

Codage par répétition :



Bit de parité :



La distance de Hamming du code se lit alors comme le nombre minimum d'arêtes entre deux sommets correspondant à des mots du code.

Distance d'un code et capacité de correction

La distance de Hamming d'un code est directement liée à ses performances :

Théorème

Soit C un code de distance d, et soit w' un mot reçu, comportant au plus e erreurs de transmission par rapport au mot envoyé w.

- *Si $e < d$, le code C permet de détecter si le mot reçu w' est erroné.*
- *Si $e < d/2$, le code C permet de corriger le mot reçu.*

Un code C de distance d permet donc de détecter jusqu'à $d - 1$ erreurs et de corriger jusqu'à $E(\frac{d-1}{2})$ erreurs.

Distance d'un code et capacité de correction

Démonstration

Cas $e < d$:
On suppose qu'il y a eu au moins une erreur de transmission. Le mot reçu w' est détecté comme erroné si ce n'est pas un mot du code.
Par définition, pour tout mot du code v différent de w, on a $d(w, v) \geq d$.
Or comme $d(w, w') \leq e < d$, w' ne peut pas être un mot du code.

Distance d'un code et capacité de correction

Démonstration - suite

Cas $e < d/2$ **:**
Pour tout mot du code v différent de w, on a par définition $d \leq d(w, v)$.
L'inégalité triangulaire donne : $d(w, v) \leq d(w, w') + d(w', v)$.
Maintenant par hypothèse $d(w, w') \leq e < d/2$.
Donc $d < d/2 + d(w', v)$, soit $d/2 < d(w', v)$.
Et comme $d(w, w') \leq d/2$, on en déduit que pour tout mot du code v différent de w, $d(w', w) < d(w', v)$.
Le mot reçu w' est bien corrigé, car w est le mot du code le plus proche de w'.

Distance d'un code et capacité de correction

Remarques :
Dès que le nombre d'erreurs est supérieur ou égal à d, on peut trouver des messages envoyés w tels que le message reçu soit aussi un mot du code.

Dès que le nombre d'erreurs est supérieur ou égal à $d/2$, on peut trouver des messages envoyés w tels que le message reçu soit plus proche d'un autre mot du code que de w.

Exemples :

- Codage par répétition : on a $d = 3$, donc ce code détecte jusqu'à 2 erreurs et corrige une erreur.
- Codage par bit de parité : on a $d = 2$, donc ce code détecte une erreur et ne peut pas en corriger.

Performance d'un code correcteur

La performance d'un code détecteur/correcteur d'erreurs se juge par :

- sa capacité de détection et de correction d'erreurs, mesurée par la distance de Hamming d du code, d'une part
- l'augmentation de la taille des messages, mesurée par le rapport n/p, d'autre part.

Un bon code est un code qui, à p et n fixé, maximise d (ou qui, à p et d fixé, minimise n).

Il y a de toute façon un compromis à trouver entre les deux.
Par exemple, si le taux d'erreurs est très faible et les messages faciles à réexpédier, savoir détecter une erreur peut suffire.
Par contre si le taux d'erreurs est élevé et les messages couteux à réexpédier, on va privilégier la capacité de correction au débit.

Inégalité et borne de Hamming

La taille (n, p) d'un code et sa distance de Hamming d sont reliées par l'**inégalité de Hamming** (vue en TD) :

$$\sum_{k=0}^{t} C_n^k (s-1)^k \leq s^{n-p}$$

où s est le nombre de symboles de l'alphabet A et où $t = E((d-1)/2)$ est la capacité de correction d'erreurs du code.

Cette inégalité permet de :

- majorer d en connaissant p et n
- majorer p en connaissant d et n
- minorer n en connaissant d et p

Règles de calculs dans $\mathbb{Z}/2\mathbb{Z}$

Dans la suite du cours, on se placera toujours sur l'alphabet $A = \{0, 1\}$, sauf mention du contraire.

On note $\mathbb{Z}/2\mathbb{Z}$ l'ensemble $\{0, 1\}$ muni des deux lois internes suivantes :

Addition, notée $+$, correspond au "ou exclusif" en logique booléenne :

$+$	0	1
0	0	1
1	1	0

Multiplication, notée $\times$, correspond au "et" en logique booléenne :

$\times$	0	1
0	0	0
1	0	1

Ces deux opérations ont les propriétés usuelles de l'addition et de la multiplication : associativité, commutativité, distributivité de la multiplication par rapport à l'addition.

Mathématiquement, on dit que $\mathbb{Z}/2\mathbb{Z}$ est un **corps**.

Règles de calculs dans $(\mathbb{Z}/2\mathbb{Z})^n$

La notation $(\mathbb{Z}/2\mathbb{Z})^n$ désigne l'ensemble des n-uplets d'éléments de $\mathbb{Z}/2\mathbb{Z}$, **identifié** avec l'ensemble des mots binaires de longueur n.

Le n-uplet $(0, 0, \ldots, 0) = 00\ldots0$ est noté 0_n.

Sur $(\mathbb{Z}/2\mathbb{Z})^n$, on définit deux lois :

- l'addition, noté $+$: $(\mathbb{Z}/2\mathbb{Z})^n \times (\mathbb{Z}/2\mathbb{Z})^n \to (\mathbb{Z}/2\mathbb{Z})^n$
  $(v_1, v_2, \ldots, v_n) + (w_1, w_2, \ldots, w_n) = (v_1 + w_1, v_2 + w_2 \ldots, v_n + w_n)$
- la multiplication scalaire, noté $\times$ ou $\cdot$: $\mathbb{Z}/2\mathbb{Z} \times (\mathbb{Z}/2\mathbb{Z})^n \to (\mathbb{Z}/2\mathbb{Z})^n$
  $\lambda \cdot (v_1, v_2, \ldots, v_n) = (\lambda \cdot v_1, \lambda \cdot v_2, \ldots, \lambda \cdot v_n)$

Remarque : l'addition dans $(\mathbb{Z}/2\mathbb{Z})^n$est une opération différente de la somme de deux nombres écrits en binaire. En particulier, on ne fait pas de report de retenue.

Règles de calculs dans $(\mathbb{Z}/2\mathbb{Z})^n$

L'addition et la multiplication par un scalaire ont les mêmes propriétés dans $(\mathbb{Z}/2\mathbb{Z})^n$ que dans $\mathbb{R}^n$.

Mathématiquement, on dit que $(\mathbb{Z}/2\mathbb{Z})^n$ est un $\mathbb{Z}/2\mathbb{Z}$**-espace vectoriel**.

Toutes les notions vues sur les espaces vectoriels réels s'appliquent à $(\mathbb{Z}/2\mathbb{Z})^n$ (combinaisons linéaires, sous-espaces vectoriels, bases, applications linéaires, matrices ...).

Attention

Pour tout $x \in (\mathbb{Z}/2\mathbb{Z})^n$, on a $x + x = 0_n$, c'est-à-dire $x = -x$!
En particulier, $x + y = z \Leftrightarrow x = y + z$

Matrices

On note $\mathcal{M}_{p,n}(\mathbb{Z}/2\mathbb{Z})$ l'ensemble des matrices de taille $p \times n$ à coefficients dans $\mathbb{Z}/2\mathbb{Z}$.

La multiplication matricielle $\mathcal{M}_{p,k}(\mathbb{Z}/2\mathbb{Z}) \times \mathcal{M}_{k,n}(\mathbb{Z}/2\mathbb{Z}) \to \mathcal{M}_{p,n}(\mathbb{Z}/2\mathbb{Z})$ s'effectue comme pour les matrices réelles (en utilisant les opérations de $\mathbb{Z}/2\mathbb{Z}$).

Exemple :

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

Matrices

Convention

L'usage est d'écrire les éléments de $(\mathbb{Z}/2\mathbb{Z})^n$ "en ligne", comme des mots (ex : $(0,1,1) \leftrightarrow 011$), et pas "en colonne", comme des vecteurs.

On **identifie** donc $(\mathbb{Z}/2\mathbb{Z})^n$ avec $\mathcal{M}_{1,n}(\mathbb{Z}/2\mathbb{Z})$, l'ensemble des matrices lignes de longueur n.

Avec cette convention, toute matrice $M \in \mathcal{M}_{p,n}(\mathbb{Z}/2\mathbb{Z})$ donne une application $f_M : (\mathbb{Z}/2\mathbb{Z})^p \to (\mathbb{Z}/2\mathbb{Z})^n$.

Exemple : $M = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}$

$$f_M(1101) = (1\ 1\ 0\ 1) \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix} = 111100$$

Applications linéaires

Définition

Une application $f : (\mathbb{Z}/2\mathbb{Z})^p \to (\mathbb{Z}/2\mathbb{Z})^n$ est **linéaire** *si et seulement si pour tous mots x, y de $(\mathbb{Z}/2\mathbb{Z})^p$, on a $f(x+y) = f(x) + f(y)$.*

Théorème

Pour toute application linéaire $f : (\mathbb{Z}/2\mathbb{Z})^p \to (\mathbb{Z}/2\mathbb{Z})^n$, il existe une unique matrice $M \in \mathcal{M}_{p,n}(\mathbb{Z}/2\mathbb{Z})$ telle que $f = f_M$.

Réciproquement, pour toute matrice $M \in \mathcal{M}_{p,n}(\mathbb{Z}/2\mathbb{Z})$, l'application f_M est linéaire.

Sous-espaces vectoriels

Définition

Une partie non vide E de $(\mathbb{Z}/2\mathbb{Z})^n$ est un **sous-espace vectoriel** *si elle est stable par addition :*

$$\forall x, y \in E,\ x + y \in E$$

Une **base** *d'un sous-espace vectoriel E est une famille $(e_1, e_2, \dots e_p)$ d'éléments de E, telle que tout élément de E s'écrive de façon unique comme combinaison linéaire des $e_1, \dots e_p$:*

$$\forall x \in E,\ \exists! \lambda_1, \dots, \lambda_p \in \mathbb{Z}/2\mathbb{Z},\ x = \lambda_1 e_1 + \dots + \lambda_p e_p$$

Sous-espaces vectoriels

Théorème

Soit E un sous-espace vectoriel de $(\mathbb{Z}/2\mathbb{Z})^n$.

- *Toutes les bases de E ont le même nombre d'éléments ; ce nombre est appelé la* **dimension** *de E.*
- *Si B est une base de E, l'ensemble des combinaisons linéaires des éléments de B est égal à E.*
- *Le cardinal de E est 2^p, où p est la dimension de E.*

Sous-espaces vectoriels

Exemples

- $(\mathbb{Z}/2\mathbb{Z})^n$ est un sous-espace vectoriel trivial, sa dimension est n. Une base de $(\mathbb{Z}/2\mathbb{Z})^n$ est 100...0, 010...0, 001...0, ..., 000...1.
  Exemple : $1011 = 1 \times 1000 + 0 \times 0100 + 1 \times 0010 + 1 \times 0001$
- $E = \{10, 01, 11\}$ n'est pas un sous-espace vectoriel de $(\mathbb{Z}/2\mathbb{Z})^2$: en effet $10 + 10 = 00 \notin E$.
- L'ensemble des mot binaires de longueur n ayant un nombre pair de 1 est un sous-espace vectoriel de $(\mathbb{Z}/2\mathbb{Z})^n$, de dimension $n-1$. Une base en est 100...01, 010...01, 001...01, ..., 000...11.
  Exemple : $1010 = 1 \times 1001 + 0 \times 0101 + 1 \times 0011$

Distance de Hamming entre mots binaires

Si $A = \mathbb{Z}/2\mathbb{Z}$, on vérifie que la distance de Hamming est **invariante par translation** :

$$d(x, y) = d(x + z, y + z) \quad \forall x, y, z \in (\mathbb{Z}/2\mathbb{Z})^n$$

En particulier, $d(x, y) = d(x + y, y + y) = d(x + y, 0)$

Définition

Le **poids** $p(w)$ *d'un mot binaire* w *est son nombre bits égaux à* 1.

Le poids d'un mot binaire w est égal à sa distance au mot nul : $p(w) = d(w, 0)$

On a alors pour tous mots binaires x et y de même longueur, $d(x, y) = p(x + y)$.

Codes linéaires

Définition

Un code binaire C *est* **linéaire** *si la somme de deux mots quelconques du code est encore un mot du code :*

$$\forall w_1, w_2 \in C, \ w_1 + w_2 \in C$$

Un code linéaire est donc un **sous-espace vectoriel** de $(\mathbb{Z}/2\mathbb{Z})^n$. En particulier le nombre de mots du code est 2^p, où p est la dimension de C.

Remarque : dans un code linéaire, le mot 0_n est toujours un mot du code : en effet si x est un mot du code quelconque, alors $x + x = 0_n$ est aussi un mot du code.

Distance d'un code linéaire

Théorème

Soit C *un code linéaire.*

- *La distance du code est égale au poids du mot du code non nul de plus petit poids :*

$$d(C) = \min\{p(w) \ ; \ w \in C, \ w \neq 0_n\}$$

- *Pour tout mot du code* w*, la plus petite distance entre* w *et un autre mot du code est égale à* $d(C)$.

La distance de Hamming d'un code est donc beaucoup plus facile à déterminer quand le code est linéaire.

Démonstration

On rappelle que $p(w) = d(0_n, w)$, et que 0_n est un mot du code. Donc :

$$\begin{aligned}\min\{p(w)\ ;\ w \in C,\ w \neq 0_n\} &= \min\{d(0_n, w)\ ;\ w \in C,\ w \neq 0_n\} \\ &\leq \min\{d(v, w)\ ;\ v, w \in C,\ w \neq v\} \\ &\leq d(C)\end{aligned}$$

Réciproquement, si x, y sont deux mots (distincts) du code tels que $d(x, y) = d(C) = \min\{d(v, w)\ ;\ v, w \in C,\ w \neq v\}$, alors $d(C) = d(x, y) = p(x + y)$ et $x + y$ est un mot du code non nul. Donc : $d(C) = p(x + y) \leq \min\{p(w)\ ;\ w \in C,\ w \neq 0_n\}$.

Pour le deuxième point : soit x un mot du code tel que $d(C) = p(x) = \min\{p(w)\ ;\ w \in C,\ w \neq 0_n\}$. Alors $w + x$ est un mot du code, et $d(w, w + x) = p(w + w + x) = p(x) = d(C)$.

Matrice génératrice

Définition

On appelle **matrice génératrice** *d'un code linéaire C toute matrice dont les lignes forment une base de C.*

Un code linéaire est complètement décrit par une matrice génératrice.

Exemple :
$G = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}$ est la matrice génératrice du code $\{00000, 00110, 11000, 11110, 10111, 10001, 01111, 01001\}$. La distance de ce code est 2.

Codage linéaire

Définition

Un codage est **linéaire** *si l'application de codage $\phi : A^p \to A^n$ est linéaire, c'est-à-dire si $\phi(x + y) = \phi(x) + \phi(y)$ pour tous mots x, y de A^p.*

Le code correspondant (i.e. l'image de ϕ) à un codage linéaire est un code linéaire.

L'application ϕ est linéaire donc peut s'écrire sous forme **matricielle**. La matrice correspondant à ϕ est une matrice génératrice du code.

Les codages linéaires sont donc faciles à décrire : ils sont explicités par la donnée d'une matrice.

Exemples

- Le code $\{000, 111\}$ est linéaire. Sa matrice génératrice est $G = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$
- Le code $\{101, 010\}$ n'est pas linéaire.
- Le codage par bit de parité est linéaire. La matrice génératrice correspondante (en taille $(4, 3)$) est $G = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$

Codages linéaires systématiques

Rappel : un codage est systématique si le mot d'origine est un préfixe du mot codé.

Proposition

Un codage linéaire est systématique si et seulement si la matrice génératrice G correspondante est la juxtaposition de la matrice identité I_p et d'une **matrice de parité** $P \in \mathcal{M}_{p,n-p}(\mathbb{Z}/2\mathbb{Z})$ *:* $G = (I_p \vdots P)$

Exemples :

- Codage $(3,1)$ par répétition : $G = \left(\begin{array}{c|cc} 1 & 1 & 1 \end{array}\right)$, $P = \begin{pmatrix} 1 & 1 \end{pmatrix}$
- Codage $(4,3)$ par bit de parité : $G = \left(\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{array}\right)$, $P = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$

Matrice génératrice standard

Etant donné un code linéaire C, on aimerait en connaître un codage systématique.

Cela revient à chercher une **matrice génératrice standard** du code C, c'est-à-dire une matrice génératrice de la forme $G = (I_p \vdots P)$.

Si elle existe, la matrice génératrice standard est unique. On l'obtient en appliquant l'**algorithme de Gauss**.

Matrice génératrice standard

Exemples

Soit C le code définit par sa matrice génératrice standard $G = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$. On applique l'algorithme de Gauss à G :

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix} \xrightarrow{L_2 \leftarrow L_2 + L_1} \rightarrow$$

$$\begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} \begin{array}{l} L_1 \leftarrow L_1 + L_2 \\ \\ L_3 \leftarrow L_3 + L_2 \end{array} \rightarrow \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} \xrightarrow{L_1 \leftarrow L_1 + L_3}$$

Certains codes linéaires n'admettent pas de matrices génératrices standard, par exemple $\{000, 001, 010, 011\}$.

Classes d'équivalence

Le décodage pour un code "quelconque" est fastidieux : il faut déterminer, pour chaque mot susceptible d'être reçu, le mot du code le plus proche.

La structure d'un code linéaire permet de simplifier en partie ce travail.

Soit C un code linéaire de taille (n, p). On définit sur $(\mathbb{Z}/2\mathbb{Z})^n$ la **relation binaire** $\mathcal{C}$:

$$u \mathcal{C} v \Leftrightarrow u + v \in C$$

Proposition

La relation $\mathcal{C}$ est une **relation d'équivalence**.

Cette relation possède 2^{n-p} classes d'équivalence. Le code C est la classe d'équivalence de 0_n.

Classes d'équivalence

Démonstration

- **Réflexivité :** pour tout $x \in (\mathbb{Z}/2\mathbb{Z})^n$, on a $x + x = 0_n$, et 0_n est un mot du code car le code est linéaire ; donc $x \mathcal{C} x$.
- **Symétrie :** si $x \mathcal{C} y$, alors par définition $x + y \in C$, et on a directement $y + x = x + y \in C$, soit $y \mathcal{C} x$.
- **Transitivité :** si $x \mathcal{C} y$ et $y \mathcal{C} z$, alors par définition $x + y \in C$ et $y + z \in C$. On a $x + z = x + (y + y) + z = (x + y) + (y + z)$; comme le code est linéaire et que $x + y$ et $y + z$ sont des mots du code, $x + z$ est un mot du code : $x \mathcal{C} z$.

Classes d'équivalence

Démonstration - suite et fin

- On note $[x]$ la classe d'équivalence d'un mot x pour la relation $\mathcal{C}$.
  Si w est un mot du code alors $(x + w) \mathcal{C} x$, c'est-à-dire $x + w \in [x]$: en effet $(x + w) + x = w \in C$.
- On peut alors regarder l'application $\tau_x : C \to [x]$, $w \mapsto x + w$.
  Elle est **injective** : $\tau_x(w) = \tau_x(w') \Rightarrow x + w = x + w' \Rightarrow w = w'$.
  Elle est **surjective** : si $y \in [x]$, alors $x + y \in C$ est un antécédent de y : $\tau_x(x + y) = x + x + y = y$.
  L'application τ_x est bijective. On en déduit que chaque classe d'équivalence a le même nombre d'éléments que C, c'est-à-dire 2^p.
- Les classes d'équivalence forment une partition de $(\mathbb{Z}/2\mathbb{Z})^n$ par des sous-ensembles de cardinal 2^p. Il y a donc $card((\mathbb{Z}/2\mathbb{Z})^n)/2^p = 2^{n-p}$ classes d'équivalence.
- Enfin, il est clair que la classe de 0_n est C : en effet $w \mathcal{C} 0_n \Leftrightarrow w + 0_n \in C \Leftrightarrow w \in C$.

Vecteur d'erreur

On se place dans le cadre d'une transmission utilisant un code linéaire C.

Définition

Soit $w \in C$ le mot du code envoyé et soit $w' \in (\mathbb{Z}/2\mathbb{Z})^n$ le mot reçu.
On appelle **vecteur d'erreur** *de la transmission le mot $e = w + w'$.*

Le vecteur d'erreur vérifie $w = w' + e$: il indique précisément quels sont les erreurs ayant eu lieu pendant la transmission.

Propriété

Le vecteur d'erreur e appartient à la classe d'équivalence du mot reçu w'.

Recherche du vecteur d'erreur

En pratique, on ne connait que le mot reçu w', et on veut trouver le mot du code le plus proche de w'.

Cela revient à rechercher, parmi tous les vecteurs d'erreur possibles, *c'est-à-dire parmi tous les éléments de la classe d'équivalence de w'*, celui de plus petit poids.

Une première méthode de décodage consiste donc à déterminer, préalablement, un élément de plus petit poids dans chacune des classes d'équivalence du code : c'est la méthode du **tableau standard**.

Construction du tableau standard

Un tableau standard pour le code C contient tous les mots de $(\mathbb{Z}/2\mathbb{Z})^n$. Sur chaque ligne sont rangés tous les éléments d'une même classe d'équivalence.

On procède de la façon suivante :

- première ligne : on liste les mots de C, en commençant par $a_1 = 0_n$
- deuxième ligne : on choisit un mot a_2, de poids minimum, qui n'est pas déjà dans le tableau (on obtient a_2 en parcourant tous les mots de poids 1, puis 2, 3...). On remplit alors la ligne en inscrivant $a_2 + x$ dans la colonne ayant au sommet le mot du code x.
- troisième ligne : on choisit un mot a_3, de poids minimum, qui n'est pas déjà dans le tableau. On remplit alors la ligne en inscrivant $a_3 + x$ dans la colonne ayant au sommet le mot du code x.
- On continue de la même façon jusqu'à que tous les mots du code soient inscrits et que les 2^{n-p} lignes soient remplies.

Exemple

Soit C le code linéaire de taille $(4, 2)$, de matrice génératrice $G = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$, c'est-à-dire le code $C = \{0000, 1011, 0101, 1110\}$.

Construction du tableau standard :

0000	1011	0101	1110
1000	0011	1101	0110
0100	1111	0001	1010
0010	1001	0111	1100

Remarque : le tableau standard n'est pas unique. En particulier, si dans une classe d'équivalence le mot de plus petit poids n'est pas unique, le choix de celui qui est en tête de ligne dans le tableau aura une influence sur le décodage.

Utilisation du tableau standard

Méthode

Soit w' le mot reçu. On cherche sa position dans le tableau, puis on le corrige par le mot w situé en haut de la même colonne.

Cela revient à ajouter à w' le vecteur d'erreur a_i situé en tête de sa ligne.

Par construction du tableau, a_i est un élément de poids minimum dans sa classe d'équivalence. Donc w est bien un mot du code le plus proche de w', ce qui justifie cette méthode de décodage.

Exemple

On avait obtenu le tableau standard suivant :

0000	1011	0101	1110
1000	0011	1101	0110
0100	1111	0001	1010
0010	1001	0111	1100

Si on reçoit le mot 0111 : on le corrige en 0101. Le vecteur d'erreur est 0010.

Si on reçoit le mot 0110 : on le corrige en 1110. Le vecteur d'erreur est 1000.

Mesure probabiliste de l'efficacité de la méthode

Préliminaire : Soit w un message envoyé. La probabilité que le message reçu soit un mot donné w' dépend uniquement du vecteur d'erreur $e = w' + w$, qui indique quelles erreurs de transmission doivent se produire.

$$\begin{aligned} P(\text{message reçu}{=}w') &= P(\text{vecteur d'erreur}{=}e) \\ &= p^{p(e)}\,(1-p)^{n-p(e)} \end{aligned}$$

Exemple : Si le mot envoyé est 1101, la probabilité que le mot reçu soit 0111 est

$$\begin{aligned} P(\text{message reçu}{=}0111) &= P(\text{vecteur d'erreur}{=}1010) \\ &= P(\text{1er bit erroné}) \times P(\text{2e bit correct}) \\ &\quad \times P(\text{3e bit erroné}) \times P(\text{4e bit correct}) \\ &= p^2(1-p)^2 \end{aligned}$$

Mesure probabiliste de l'efficacité de la méthode

Soit w le mot du code envoyé. Le mot reçu w' est corrigé en w si w et w' sont dans la même colonne.

Par construction du tableau standard, w et w' sont dans la même colonne si et seulement si leur somme $w + w'$ est dans la première colonne du tableau.
La correction fonctionne donc quand le vecteur d'erreur est dans la première colonne.

Proposition

On note a_i les mots de la $1^{ère}$ colonne du tableau standard, w le mot du code envoyé, w' le message reçu et $e = w + w'$ le vecteur d'erreur. Alors

$$P(w' \text{ corrigé en } w) = \sum_{i=0}^{2^{n-p}} P(e{=}a_i) = \sum_{i=0}^{2^{n-p}} p^{p(a_i)}\,(1-p)^{n-p(a_i)}$$

Mesure probabiliste de l'efficacité de la méthode

Application

On reprend le tableau standard construit précédemment :

0000	1011	0101	1110
1000	0011	1101	0110
0100	1111	0001	1010
0010	1001	0111	1100

La probabilité que le décodage soit correct est

$$P(e{=}0000)+P(e{=}1000)+P(e{=}0100)+P(e{=}0010) = (1-p)^4+3\,p(1-p)^3$$

Par exemple avec $p = 10^{-4}$, la probabilité que le décodage soit correct est environ 0,9999.

Inconvénient de la méthode

La méthode de correction par le tableau standard présente plusieurs inconvénients :

- Le tableau est long à construire.
- Dès que la taille des blocs est relativement importante ($n \geq 30$ environ), le tableau devient beaucoup trop gros pour être utilisable.
- La recherche du message reçu dans le tableau est lente.

On va voir une deuxième méthode, plus algébrique, de correction des messages.

Un exemple de détection d'erreurs

On utilise pour la transmission un codage linéaire systématique de taille $(3,6)$, donné par la matrice $\begin{pmatrix}1&0&0&1&1&0\\0&1&0&1&0&1\\0&0&1&0&1&1\end{pmatrix}$.

On reçoit le message 011111. Comment savoir si ce message est correct ?

Le codage est systématique, donc si ce message est correct, le mot d'origine était 011, et on devrait retrouver le message reçu en codant 011.

On compare donc le message reçu 011111 avec celui obtenu en codant les trois premiers bits $011 \mapsto 011110$

Seule la comparaison des trois bits de contrôle est pertinente. On a trouvé 110 à la place des bits reçus 111, l'erreur entre les deux est $110 + 111 = 001 \neq 000$, donc le message reçu n'est pas correct.

Un exemple de détection d'erreurs

Cette opération

- garder les p premiers bits du message reçu, les recoder,
- prendre les $n-p$ bits de contrôle du résultat et les additionner aux bits de contrôles du message reçu,
- comparer le résultat à 0,

revient à faire le produit du message reçu avec la matrice $H = \begin{pmatrix} P \\ I_{n-p} \end{pmatrix}$,

où P est la matrice de parité du codage.

Dans l'exemple précédent, on a $P = \begin{pmatrix}1&1&0\\1&0&1\\0&1&1\end{pmatrix}$, donc $H = \left(\begin{array}{ccc}1&1&0\\1&0&1\\0&1&1\\\hline 1&0&0\\0&1&0\\0&0&1\end{array}\right)$.

On vérifie qu'on a bien $011111 \times H = 001$.

Matrice de contrôle - syndrômes

Définition

Soit C un code linéaire de taille (n,p) ayant une matrice génératrice standard $G = (I_p\, P)$, P étant la matrice de parité.
La **matrice de contrôle** *du code C est la matrice de taille $n \times (n-p)$* $H = \begin{pmatrix} P \\ I_{n-p} \end{pmatrix}$.

Définition

Soit m un mot binaire de taille n. On appelle **syndrôme** *de m, et on note $\sigma(m)$, le produit mH. C'est un mot de longueur $n-p$.*

Le syndrôme correspond à la somme des bits de contrôles du message reçu et des bits de contrôle recalculés.

Exemples

- Codage $(3,1)$ par répétition : $G = \begin{pmatrix}1&1&1\end{pmatrix}$, donc $H = \begin{pmatrix}1&1\\1&0\\0&1\end{pmatrix}$.
  Exemple de calcul de syndrôme : $\sigma(101) = 10$, $\sigma(111) = 00$.
- Codage $(4,3)$ par parité : $G = \begin{pmatrix}1&0&0&1\\0&1&0&1\\0&0&1&1\end{pmatrix}$, donc $H = \begin{pmatrix}1\\1\\1\\1\end{pmatrix}$.
  Exemple de calcul de syndrôme : $\sigma(1011) = 1$, $\sigma(1001) = 0$.
- Autre code : $G = \begin{pmatrix}1&0&0&1&1\\0&1&0&1&0\\0&0&1&0&1\end{pmatrix}$, donc $H = \begin{pmatrix}1&1\\1&0\\0&1\\1&0\\0&1\end{pmatrix}$. Exemple de calcul de syndrôme : $\sigma(01010) = 00$, $\sigma(11111) = 11$.

Matrice de contrôle et détection d'erreurs

Théorème

Soient C un code linéaire et H sa matrice de contrôle.
Un message m est un mot du code si et seulement si son syndrôme est nul :

$$m \in C \iff mH = 0_{n-p}$$

La matrice de contrôle fournit donc un moyen efficace de détection des erreurs.

Matrice de contrôle et détection d'erreurs

Démonstration

Soit $m \in (\mathbb{Z}/2\mathbb{Z})^n$ un mot de longueur n. On note $m_I \in (\mathbb{Z}/2\mathbb{Z})^p$ le mot formé des p premiers bits de m (bits d'informations), et $m_C \in (\mathbb{Z}/2\mathbb{Z})^{n-p}$ le mot formé des $n-p$ dernier bits de m (bits de contrôle) : $m = m_I.m_C$ (concaténation).

On utilise le fait que H s'écrit $\begin{pmatrix} P \\ I_{n-p} \end{pmatrix}$.

$mH = 0_{n-p} \Leftrightarrow (m_I.m_C) \times \begin{pmatrix} P \\ I_{n-p} \end{pmatrix} = 0_{n-p} \Leftrightarrow m_I \times P + m_C = 0_{n-p}$
$\Leftrightarrow m_C = m_I \times P \Leftrightarrow m_I.m_C = m_I \times (I_p\, P) \Leftrightarrow m = m_I \times G \Leftrightarrow m \in C.$

Applications

Soit C le code linéaire donné par sa matrice génératrice standard

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

Les mots $m_1 = 010101010$, $m_2 = 111111111$, $m_3 = 011010000$ font-ils partie du code ?

Applications

La matrice de contrôle est $H = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$.

On calcule les syndrômes :

$m_1 H = 1100$,
$m_2 H = 0110$,
$m_3 H = 0000$,

donc seul m_3 fait partie du code.

Syndrômes et classes d'équivalence du code

On rappelle la définition de la relation d'équivalence $\mathcal{C}$ associée à un code linéaire C :

$$u \mathcal{C} v \iff u + v \in C$$

Théorème

Deux messages sont équivalents pour la relation $\mathcal{C}$ si et seulement si ils ont le même syndrôme :

$$u \mathcal{C} v \iff \sigma(u) = \sigma(v)$$

Démonstration

$u \mathcal{C} v \Leftrightarrow u + v \in C \Leftrightarrow \sigma(u + v) = 0_{n-p} \Leftrightarrow \sigma(u) + \sigma(v) = 0_{n-p}$
$\Leftrightarrow \sigma(u) = \sigma(v)$

Liste des syndrômes

On a vu que le principe de la correction d'erreurs pour un code linéaire consiste à déterminer, dans chaque classe d'équivalence du code, un mot de plus petit poids (le vecteur d'erreurs).

On va utiliser le fait que les classes d'équivalences du code sont en bijection avec l'ensemble des syndrômes possibles

On va donc construire un tableau, appelé **liste des syndrômes**, associant à chaque syndrôme possible un mot de plus petit poids ayant ce syndrôme.

Construction de la liste des syndrômes

On se donne un code linéaire C, de taille (n, p), dont on connaît une matrice de contrôle H.

Méthode

- On commence par écrire dans une première colonne tous les syndrômes possibles, c'est-à-dire tous les mots de longueur $n - p$.
- On parcourt ensuite l'ensemble des mots de longueur n de poids 0, 1, puis 2, 3 etc. Pour chaque mot on calcule son syndrôme. Si c'est la première fois que ce syndrôme apparaît, on écrit le mot dans la deuxième colonne, en face de son syndrôme.
- On continue jusqu'à ce qu'il y ait un mot en face de chaque syndrôme.

Exemple

Soit C le code linéaire de taille $(4, 2)$, de matrice génératrice standard $G = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$ (même code que dans l'exemple du tableau standard).

La matrice de contrôle est $H = \begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$

Construction de la liste des syndrômes :

syndrômes	vecteurs d'erreurs
00	0000
01	0001
10	0010
11	1000

$\sigma(0000) = 00$
$\sigma(0001) = 01$
$\sigma(0010) = 10$
$\sigma(0100) = 01$, déjà présent
$\sigma(1000) = 11$

Utilisation de la liste des syndrômes

Méthode

Soit w' le mot reçu. On calcule son syndrôme $\sigma(w')$.

On regarde dans la liste le vecteur d'erreurs e correspondant à ce syndrôme, puis on corrige le mot reçu en le remplaçant par $w = w' + e$.

Par construction de la liste, e est un élément de poids minimum dans la classe d'équivalence de w'. Donc w est bien un mot du code le plus proche de w', ce qui justifie cette méthode de correction.

Remarque : la liste des syndrômes n'est pas unique. En particulier, si plusieurs mots de plus petit poids ont le même syndrôme, le choix de celui qui apparaît dans le tableau aura une influence sur la correction.

Exemple

On avait obtenu la liste des syndrômes suivante :

syndrômes	vecteurs d'erreurs
00	0000
01	0001
10	0010
11	1000

avec la matrice de contrôle

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

Si on reçoit le mot 0111 : $\sigma(0111) = 10$,
on le corrige en $0111 + 0010 = 0101$.

Si on reçoit le mot 0110 : $\sigma(0110) = 11$,
on le corrige en $0110 + 1000 = 1110$.

Remarques

- Attention : pour la correction, il faut disposer de la liste des syndrômes *et* de la matrice de contrôle (pour calculer les syndrômes).
- La correction par syndrôme est un peu plus compliquée mais plus rapide et moins encombrante que la correction par tableau standard : elle est donc à privilégier.
- La construction de la liste des syndrômes peut quand même être longue et fastidieuse. Pour certains codes linéaires particuliers, il existe des méthodes de correction plus rapides.

Matrice de contrôle et distance du code

La matrice de contrôle donne directement des informations sur la capacité de correction d'un code linéaire.

Théorème

Soit C un code linéaire, de distance de Hamming $d(C)$, et ayant une matrice de contrôle H.
On a $d(C) \geq 3$ si et seulement si les lignes de H sont toutes distinctes et non nulles.

On se servira de ce résultat pour construire des codes linéaires pouvant corriger une erreur.

Matrice de contrôle et distance du code

Démonstration

- On remarque que le syndrôme d'un mot de poids 1, 0...010...0, ayant exactement un 1 en i-ème position, est la i-ème ligne de H : les lignes de la matrice de contrôle correspondent aux syndrômes des mots de poids 1.
- Donc une ligne de H est nulle si et seulement si il existe un mot de poids 1 de syndrôme nul, c'est-à-dire un mot du code de poids 1, ce qui équivaut à $d(C) = 1$.
- De la même façon, les syndrômes des mots de poids 2 correspondent aux sommes de deux lignes distinctes de H.
- Deux lignes de H sont identiques si et seulement si leur somme est nulle, donc si et seulement si il existe un mot de poids 2 de syndrôme nul, c'est-à-dire un mot du code de poids 2.

Construction de codes linéaires

Pour construire un code linéaire C de taille (n, p) pouvant corriger au moins une erreur (i.e. $d(C) \geq 3$) :

- on construit la matrice de contrôle H de taille $n \times (n-p)$, en commençant par l'identité en bas, puis en ajoutant des lignes non nulles toutes différentes (possible uniquement si $n \leq 2^{n-p} - 1$)
- connaissant $H = \begin{pmatrix} P \\ I_{n-p} \end{pmatrix}$, on connaît la matrice de parité P, et donc la matrice génératrice standard $G = (I_p P)$ du code.

Codes de Hamming

Définition

Un **code de Hamming** *est un code linéaire dont la matrice de contrôle H est constitué de tous les mots binaires non nuls de longueur k.*
C'est un code parfait, de taille $(2^k - 1, 2^k - 1 - k)$*, et de distance* 3.

Exercice : construire un mot du code de poids 3.

Les codes de Hamming sont simples à construire et permettent de corriger exactement une erreur, ce qui justifie leur utilité.

Exemples de codes de Hamming

- Si $k = 2$: alors $H = \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$, donc $G = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$.
  On retrouve le code $(3, 1)$ par répétition : c'est le plus petit code de Hamming.
- Si $k = 3$: on construit $H = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ par exemple, donc $G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$. C'est un code de Hamming de taille $(7, 4)$.