Rapport de TP - SY26

TP06 - Codes correcteurs d'erreurs

Rémi Burtin

Cyril Fougeray

20 juin 2014

Université de Technologie de Compiègne

1 Introduction

L'objectif de ce TP est la mise en œuvre du codage de canal pour simuler leur capacité de correction et en cerner les limites. Nous allons étudier les codes en bloc (linéaires ou cycliques), pour lesquels des blocs de k symboles d'information sont protégés indépendamment les uns des autres par m symboles de contrôle pour former des mots codes à n symboles.

2 Codage de Hamming

2.1 Question 1 - Codage

Pour calculer un mot code à partir de la matrice de contrôle de parité $\mathbf{H} = [\mathbf{P}^T|\mathbf{I}_{n-k}]$ et d'un bloc d'information $\mathbf{m}$, il nous suffit de calculer la matrice génératrice $\mathbf{G}$ qui est définie par $[\mathbf{I}_k|\mathbf{P}]$. Puis on obtient le mot code $\mathbf{c}$ en multipliant $\mathbf{m}$ et $\mathbf{G}$ (multiplication modulo 2). Voir fonction *hamcode* en annexe A.1. La matrice de contrôle de parité $\mathbf{H}$ est quant à elle calculer via la fonction MATLAB *hammgen*. Cette fonction prend un paramètre M permettant le calcul de la longueur du mot code en suivant la relation $N = 2^M - 1$.

2.2 Question 2 - Décodage

Afin de décoder le mot code précédemment créé, éventuellement entaché d'erreurs, nous avons mis en place une fonction *hamdecode* (*cf.* A.2) prenant en paramètre le mot code et la matrice de contrôle de parité. Pour cela, nous commençons par calculer le syndrome $s = r \times H^T$ (r étant le mot encodé), permettant de retrouver les éventuelles erreurs glissées dans le mot encodé. En effet, si s est différent de 0, alors il existe une erreur dans le mot encodé. Cette erreur se retrouve facilement une fois la table des syndromes calculée. Ici, nous ajoutons trois bits de redondance au mot à encoder, il y aura donc 2^3 syndromes possibles. Pour chacun de ces syndromes, on cherche si il existe une configuration e de 1 erreur, puis de 2 erreurs, puis de 3... telle que $s = e \times H^t$. Pour $s = 0$, $e = 0$: il n'existe pas d'erreur. Ainsi, nous pouvons détecter $2^3 - 1 = 7$ erreurs différentes, soit une erreur sur une seule bit du message encodé. Une fonction MATLAB permet de calculer directement la table des syndromes en lui passant la matrice de contrôle de parité : *syndtable*. Sur chaque ligne de la matrice retournée par *syndtable* apparait e, correspondant à la position du ou des bits d'erreur. Une opération logique XOR permet donc de corriger l'erreur dans le mot à décoder et de retourner le bon mot code. Nous avons fait le test en insérant une erreur :

```
>> H = hammgen(3); % creation de la matrice de controle de parite
>> c = hamcode([1 0 1 0], H) % encodage du mot 0101
c =
     0     0     1   |   1     0     1     0

% decodage avec inclusion d'une erreur
>> cdecode = hamdecode( xor([0 1 0 0 0 0 0], c), H)
cdecode =
     0     0     1   |   1     0     1     0

% decodage avec inclusion de deux erreurs (limites)
>> cdecode = hamdecode( xor([0 1 0 0 0 0 1], c), H)
cdecode =
     0     1     1     1     0     0     1
```

Les bits de redondance sont ajoutés au début (trois premiers bits). Nous observons que si un bit contient une erreur, l'erreur est corrigée car le message retourné par le décodage est retrouvé. Par contre, si deux bits du message sont erronés, la table des syndromes ne permet pas de retrouver les erreurs, le message est alors altéré.

3 Codage BCH

Pour récupérer le polynôme générateur BCH permettant de corriger 3 erreurs sur 15 bits, on appelle la fonction *bchgenpoly* en lui passant les valeurs de n et k, qui sont en l'occurrence 15 et 5. Nous générons ensuite un message aléatoire de longueur k avec la fonction *randi*, que nous mettons sous forme de tableau d'éléments du corps de Galois grâce à la fonction *gf*.
Pour coder le message généré, on le passe avec n et k à la fonction *bchenc*. Puis on ajoute 3 erreurs aléatoires :

```
code_bruite = code + randerr(1,n,3)
```

Une fois le message bruité, on peut effectuer le décodage avec la fonction *bchdec* qui nous renvoie le message décodé ainsi que le nombre d'erreurs corrigées. Voir code en annexe A.3.

4 Conclusion

Ce TP nous a permis de mieux comprendre comment fonctionnent les codes correcteurs d'erreurs avec l'application deux deux techniques de détection et de correction d'erreur par code en bloc. Nous avons ainsi pu encoder différents message et leur ajouter des bits de redondance permettant la détection et la correction des erreurs dans certaines limites.

A Codes source MATLAB

A.1 Code de Hamming : Codage

```
function [ c ] = hamcode( m, H )
M = size(H, 1);
n = size(H, 2);

% H = (H(1:3,4:7) H(1:3,1:3));

G = [H(1:M,(n-M):n)' eye(n-M)];

c = m*G;

c = mod(c,2);

end
```

A.2 Code de Hamming : Décodage

```
function decode = hamdecode(r,H)

% Calcul du syndrome
s = mod(r*H', 2);
% Conversion de s (syndrome),
i = bi2de(s,2,'left-msb');

if i~= 0
    % Calcul du vecteur d'erreur
    i = i+1;
    table = syndtable(H)

    e = table(i,:);

    % Mot decode suppose emis
    decode = xor(r,e);
else
    decode = r;
end;

end
```

A.3 Test codage BCH

```
m = 4;

%Longueur du mot code (15)
n = 2^m-1;

%Longueur du message
k = 5;

%Message aleatoire
msg = gf(randi([0 1],1,k));

%Calcul polynome generateur
[gen, t] = bchgenpoly(n,k);

%Codage du message
code = bchenc(msg,n,k)

%Ajout de t erreurs aleatoires
code_bruite = code + randerr(1,n,t)

%Decodage
[msg_decode,nb_err] = bchdec(code_bruite,n,k)
```