Université de Nantes

IUT de Nantes

Codes Correcteurs

Auteurs :
Brewal Henaff
Cédric Berland
Nathan Maraval

Cours :
Modélisation
Mathémathique

16 janvier 2016

UNIVERSITÉ DE NANTES

Sommaire

1 Introduction

Ce projet à été réalisé dans le cadre de la formation en Modélisation Mathematique, en DUT Informatique à l'IUT de Nantes.
Il consiste a concevoir un programme permettant l'encodage d'un message binaire, puis par différentes methode que nous expliciterons plus tard, de le décoder et de corriger les possibles erreurs de transmission.

2 La théorie

Cette partie couvrira la théorie, les opérations effectués lors de ce projet, que ce soit lors de la transmition, ou bien de la récéption, comme du brouillage qui sera effectué pour pouvoir tester les données.

2.1 La détection d'erreur

Il existe plusieurs méthodes pour détecter les erreurs, par exemple associer un mot à une lettre comme utilisé dans l'armée :

exemple : erreur $\rightarrow$ Echo Romeo Romeo Echo Uniforme Romeo

En informatique on utilise ce qu'on appel des "bits de parité" qui indique indique si le nombre de 1 dans l'octet est pair ou non

exemple : A $\rightarrow$ 1000001 et ajoutera donc un 0 puisse que il y a deux 1 on obtiendra donc pour A le code suivant "01000001".
Le bits de parité nous permet de savoir si lors de la transmition le message à été modifié, simplement en regardant si le bit de parité correspond toujours au reste du message (si le nombre de 1 est toujours pair).

2.2 La correction d'erreur

Les différents exemple ci-dessus ne permettent que de détecter les erreurs. Mais ce qui nous intéresse vraiment c'est la correction des erreurs.
Il existe des moyens simple, par exemple la duplication du message :

"erreur" $\rightarrow$ eee rrr rrr eee uuu rrr

On répète chaque lettre 3 fois. Pourquoi 3 fois ? Et pas 2 ?
Imaginons que l'on multiplie seulement 2 fois, on reçois le message suivant : "ââgmee", le message de base peut être "âge" mais aussi "âme".

En revanche si l'on multiplie 3 fois on recevra donc un message tel que "âââgmmeee", en supposant qu'il n'y ai eu qu'une erreur, le message envoyer est donc "âme".

Le problème de ce genre de code de correction c'est qu'ils sont très lourd, et coûteux, on transmet trois fois les données, et lorsque l'on voit le poids des données généralement transmissent (images, musiques, vidéos), on s'en rend vite compte.

2.3 Le code de Hamming

Le code de Hamming est un code correcteur moderne.
Généralement on utilise un code de Hamming C(7,4). C'est à dire que l'on envoie 7 bits et que les 4 premiers contiennent les données à transmettre. Les 3 derniers servent à la détection des erreurs.
Ces 3 derniers bits sont en fait l'addition, bit à bit, des 4 autres. En pratique cela donne ça :

Soit $v_1, v_2, \ldots, v_7$ les bits transmit et u_1, u_2, u_3 et u_4 les bits contenant le message.

$$\begin{aligned} v_1 &= u_1, \\ v_2 &= u_2, \\ v_3 &= u_3, \\ v_4 &= u_4, \\ v_5 &= u_1 + u_2 + u_4, \\ v_6 &= u_1 + u_3 + u_4, \\ v_7 &= u_2 + u_3 + u_4, \end{aligned}$$

Lorsque l'on reçoit le message (bits $w_1, w_2, \ldots, w_7$), on est face à 8 possibilités :
(0) il n'y a pas d'erreur ;
(1) w_1 est erronée ;
(2) w_2 est erronée ;
(3) w_3 est erronée ;
(4) w_4 est erronée ;
(5) w_5 est erronée ;
(6) w_6 est erronée ;
(7) w_7 est erronée.

Grâce aux 3 derniers bits on peut savoir d'où provient l'erreur. Il suffit de les recalculer (bits W_5, W_6, W_7) et de les comparer, pour obtenir un des huit cas précédent :
(0) si $w_5 = W_5$ et $w_6 = W_6$ et $w_7 = W_7$;
(1) si $w_5 \neq W_5$ et $w_6 \neq W_6$;
(2) si $w_5 \neq W_5$ et $w_7 \neq W_7$;
(3) si $w_6 \neq W_6$ et $w_7 \neq W_7$;
(4) si $w_5 \neq W_5$ et $w_6 \neq W_6$ et $w_7 \neq W_7$;
(5) si $w_5 \neq W_5$;
(6) si $w_6 \neq W_6$;
(7) si $w_7 \neq W_7$.

Cependant ce code ne permet de détecter et corriger qu'une seule erreur.

3 Notre code

3.1 L'envoi du message

Test de code :

```
int GMatrix[4][8] = {{1,1,0,1,1,0,0,0},
                     {0,1,1,1,0,1,0,0},
                     {1,0,1,1,0,0,1,0},
                     {1,1,1,0,0,0,0,1}} ;
char G[4] ;
void init_generators() {
 for (int i=0; i < 4; ++i) {
   G[i] = 0 ;
   for (int j=0; j < 8; ++j)
     if (GMatrix[i][j]) G[i] |= (1 << (7-j)) ;
 }
}

 void hamming(char c, char out[2]) {
   out[0] = out[1] = 0;
   for (int i=0; i < 4; ++i) {
     if (c & (1 << i))
       out[1] ^= G[i] ;
```

```
    if (c & (1 << (i+4)))
      out[0] ^= G[i] ;
  }
}
```

3.2 Le brouillage

Afin de pouvoir tester notre code dans des conditions d'erreurs réelles, nous utilisons un programme permettant de brouiller certains bits, aléatoirements. Ce brouillage nous permet de tester notre code décodeur, étant donné que la probabilité d'un brouillage dû au code d'encodage est extremement faible. Pour se faire, nous utilisons le code transmit.c, qui simule une transmition de message (comme par example l'envois d'un message à un satelite en orbite, ce qui peut engendrer de nombreuses erreurs).

3.3 La réception et le décodage du message

4 d'assault

Comments can be added to the margins of the document using the todo command, as shown in the example on the right. You can also add inline comments too :

This is an inline comment.

Here's a comment in the margin!

4.1 Tables and Figures

Use the table and tabular commands for basic tables — see Table **??**, for example. You can upload a figure (JPEG, PNG or PDF) using the files menu. To include it in your document, use the includegraphics command as in the code for Figure **??** below.

4.2 Mathematics

LaTeX is great at typesetting mathematics. Let $X_1, X_2, \ldots, X_n$ be a sequence of independent and identically distributed random variables with $\mathrm{E}[X_i] = \mu$

and $\mathrm{Var}[X_i] = \sigma^2 < \infty$, and let

$$S_n = \frac{X_1 + X_2 + \cdots + X_n}{n} = \frac{1}{n}\sum_{i}^{n} X_i$$

denote their mean. Then as n approaches infinity, the random variables $\sqrt{n}(S_n - \mu)$ converge in distribution to a normal $\mathcal{N}(0, \sigma^2)$.

4.3 Lists

You can make lists with automatic numbering ...

1. Like this,
2. and like this.

...or bullet points ...

— Like this,
— and like this.

We hope you find writeLaTeX useful, and please let us know if you have any feedback using the help menu above.

A Annexe 1