# Codes Correcteurs

Julie Badets, Corentin Frade, Quentin Rouland, Émeric Tosi

12 mars 2014

**Résumé**

Un code correcteur d'erreur est utilisé pour transmettre un message dans un canal bruité ; il permet de reconstituer le message émis même si des erreurs (en nombre limité), dues au bruit, ont altéré le message. L'alphabet source, comme l'alphabet du code, est $\{0, 1\}$ . On s'intéresse au codage par blocs : chaque mot de longueur m est codé par un mot de longueur n avec $n \geq m$. Le codage est donc une application de $\{0, 1\}^m$ vers $\{0, 1\}^n$. Parmi les $n$ bits du mot-code que nous allons décrire, $m$ reproduisent le mot-source, les $n - m$ autres sont les bits de correction : le taux de transmission est de $\frac{n}{m}$.

On considère les erreurs comme indépendantes les unes des autres et tous les bits ont la même probabilité d'erreur. Nous nous intéressons donc aux codes correcteurs d'une façon plutôt théorique. En pratique, si on prend un exemple dans les communications sans fil, des problèmes de parasites se posent et l'indépendance des erreurs est compromise.

Pour la suite nous prendrons comme exemple un message qui est un simple caractère encodé en UTF-7. Ce message a donc une taille de 7 bits. Cela facilite ainsi les calculs et l'implémentation pour les tests.

# Sommaire

# Chapitre 1

# Code de répétition

## 1.1 Introduction

On transmet simplement plusieurs fois le même message.

## 1.2 Fiabilité

Une seule erreur peut être détectée à coup sur.

## 1.3 Probabilité de détection

## 1.4 Rendement

Le rendement de ce code est très mauvais, on double au minimum la taille du message :

$$Rendement = \text{Taille du message} * \text{Nombre de répétitions}$$

Pour un message sur 7 bits (un simple caractère encodé en UTF-7 par exemple) et avec 1 répétition seulement :

$$Rendement = \frac{7}{7 * 2} = 50\%$$

Pour le même message mais avec 2 répétitions :

$$Rendement = \frac{7}{7 * 3} \approx 33.3\%$$

# Chapitre 2

# VRC : Bit de parité

## 2.1 Introduction

le VRC (Vertical Redundancy Check), plus connu sous le nom de bit de parité, est simplement le rajout d'un bit en fin de message pour assurer la parité du message. Ce dernier bit la valeur nécessaire pour assurer un nombre pair de bit à 1 dans le message final. Il est donc à 0 pour un nombre pair de bit à 1 dans le message de départ, ou est à 1.

## 2.2 Fiabilité

Une seule erreur peut être détectée à coup sur. Toutes les erreurs où un nombre pair de bits sont modifiés ne sont pas détectées, les erreurs détectées sont donc celles où un nombre impair de bits ont changé d'état. Si une seule erreur intervient mais porte sur le bit de parité, le message est considéré comme invalide. Ce code ne permet pas la correction d'erreur, il est nécessaire de redemander l'envoi du message détecté invalide.

## 2.3 Probabilité de détection

$$P(\text{Transmission Parfaite}) = P(X = 0) = \binom{8}{0} p^0 (1-p)^{8-0} = (1-p)^8$$

$$P(\text{Message Erroné}) = 1 - P(\text{Transmission Parfaite}) = 1 - (1-p)^8$$

$$P(\text{Détection}) = P(1 \text{ erreur}) + P(3 \text{ erreurs}) + P(5 \text{ erreurs}) + P(7 \text{ erreurs})$$
$$= \binom{8}{1} p^1 (1-p)^{8-1} + \binom{8}{3} p^3 (1-p)^{8-3} + \binom{8}{5} p^5 (1-p)^{8-5} + \binom{8}{7} p^7 (1-p)^{8-7}$$

$$= 8p(1-p)^7 + \binom{8}{3}p^3(1-p)^5 + \binom{8}{5}p^5(1-p)^3 + 8p^7(1-p)$$

$$P(\text{Reconnaissance Erreur}) = \frac{P(\text{Détection})}{P(\text{Message Erroné})}$$

Pour une probabilité de 10% d'erreurs :

$$P(\text{Transmission Parfaite}) = (1-0.1)^8 = 43\%$$

$$P(\text{Message Erroné}) = 1 - 0.43 = 57\%$$

$$P(\text{Détection}) = 8*0.1(1-0.1)^7 + \binom{8}{3}0.1^3(1-0.1)^5 + \binom{8}{5}0.1^5(1-0.1)^3 + 8*0.1^7(1-0.1)$$

$$P(\text{Détection}) = 0.8 * 0.9^7 + 56 * 0.1^3 * 0.9^5 + 56 * 0.1^5 * 0.9^3 + 8 * 0.1^7 * 0.9$$

$$P(\text{Détection}) = 42\%$$

$$P(\text{Reconnaissance Erreur}) = \frac{0.42}{0.57} \approx 74\%$$

## 2.4 Rendement

Le rendement de ce code est très bon :

$$\textit{Rendement} = \frac{\text{Taille du message}}{\text{Taille du message} + 1}$$

Pour notre message d'exemple (un simple caractère encodé en UTF-7) le rendement est déjà excellent :

$$\textit{Rendement} = \frac{7}{7+1} = 87.5\%$$

## 2.5 Exemple pratique

```
>> perl envoiNoise.pl 8001 127.0.0.1:9000

 -> envoi du caractere : o

 -> nombre de caracteres envoyes : 100000
```

```
>> perl receptionNoise.pl 9000 127.0.0.1:8001

 -> nombre de receptions : 100000
 -> nombre de receptions supposees bonnes : 58311
 -> nombre de vrais bons caracteres : 43079
 -> nombre d'erreurs au total : 56921
 -> nombre d'erreurs detectees : 41689
 -> nombre d'erreurs non detectees : 15232
 -> fiabilitee de l'envoi/reception : 43.079%
 -> fiabilitee de detection d'erreur : 73.2401047065231%
```

# Chapitre 3

# LRC :Contrôle parité croisée

## 3.1 Introduction

À partir de plusieurs message encodés grâce au VRC. Pour rester cohérent avec notre message d'exemple nous prendrons le cas d'une matrice carrée composée de 7 messages de taille 7 chacun. Il sera appliqué horizontalement à la matrice (à chaque message) le bit de parité. La matrice passera donc à une taille de 7 sur 8. Le dernier message qui sera généré grâce à l'application du bit de parité verticalement sur la matrice.

## 3.2 Fiabilité

## 3.3 Probabilité de détection

## 3.4 Rendement

Le rendement de ce code est bon :

$$Rendement = \frac{n * \text{Taille du message}}{(n+1) * (\text{Taille du message} + 1)}$$

Dans le cas que nous étudions (7 messages de 7 bits) :

$$Rendement = \frac{7 * 7}{(7+1) * (7+1)} \approx 76.6\%$$

# Chapitre 4

# CRC : Code de redondance cyclique

## 4.1 Introduction

le CRC (Cyclic Redundancy Check), contrôle de redondance cyclique, représente la principale méthode de détection d'erreurs utilisée dans les télécommunications et consiste à protéger des blocs de données, appelés trames. À chaque trame est associé un bloc de données, appelé code de contrôle (parfois CRC par abus de langage).

On choisit un polynôme générateur, fixé et donc connu des deux entités qui se transmettent le message. Grâce à celui ci, l'émetteur peut générer le code de contrôle qui est le reste de la division avec le message à envoyer. Le récepteur divise ce qu'il a reçut, retrouve le message et sait si il y a eu un problème.

Il existe plusieurs variantes du CRC selon le choix du polynôme : CRC 12, CRC 16, CRC CCIT v41, CRC 32, CRC ARPA.

## 4.2 Fiabilité

Deux erreurs peuvent être détectées à coup sur grâce au CRC 16. Les erreurs détectées sont seulement celles où un nombre impairs de bit ont changé d'état ou celles qui sont des suites de bit qui ont tous changés (rafales), de taille inférieur au degré du polynôme.

## 4.3 Probabilité de détection

## 4.4 Rendement

Le rendement de ce code est dépendant de la taille du message :

$$Rendement = \frac{\text{Taille du message}}{\text{Taille du message} + \text{Degré du polynôme}}$$

Pour notre message d'exemple, un message de seulement 7 bit de longueur, un codage avec du CRC 16 donne un rendement très médiocre :

$$Rendement = \frac{7}{7+16} \approx 30\%$$

Cependant avec un message de taille plus importante comme par exemple un message de 128 bit avec du CRC 16 le rendement devient excellent :

$$Rendement = \frac{128}{128+16} \approx 89\%$$

# Chapitre 5

# Hamming

## 5.1 Introduction

On montre que si deux mots distincts du code de hamming diffèrent au moins en d bits, alors le code permet de corriger exactement (d-1)/2 erreurs.

## 5.2 Fiabilité

## 5.3 Probabilité de détection

## 5.4 Rendement

## .1 Exemple(s)

*emphatique* **gras** `machine à écrire` *incliné* PETITES MAJUSCULES

The foundations of the rigorous study of *analysis* were laid in the nineteenth century, notably by the mathematicians Cauchy and Weierstrass. Central to the study of this subject are the formal definitions of *limits* and *continuity*.

Let $D$ be a subset of $\mathbf{R}$ and let $f\colon D \to \mathbf{R}$ be a real-valued function on $D$. The function $f$ is said to be *continuous* on $D$ if, for all $\epsilon > 0$ and for all $x \in D$, there exists some $\delta > 0$ (which may depend on $x$) such that if $y \in D$ satisfies

$$|y - x| < \delta$$

then

$$|f(y) - f(x)| < \epsilon.$$

One may readily verify that if $f$ and $g$ are continuous functions on $D$ then the functions $f + g$, $f - g$ and $f.g$ are continuous. If in addition $g$ is everywhere non-zero then $f/g$ is continuous.

```
1       my $url = 'http://192.168.0.7:8080/POG/getINDI.jsp?zarefno=a1
            .';
2       use LWP::Simple;
3       my $content = get $url;
4       die "Couldn't get $url" unless defined $content;
5       print $content;
6       print "\n";
7       print "Length " + length($content)
```

```
1  #!/usr/bin/perl
2
3  use strict;
4  use Physique::LinkUDP;
5  use Parity;
6
7  sub verification {
8      my $car = ord($_[0]);
9
10 # retour du resultat du test de la parite de ce caractere :
11     return not Parity::parity($car);
12 }
13
14 sub decodage {
15     my $car = ord($_[0]);
16
17 # ou retourner le caractere vide si il y a erreur :
18     return '' unless ( verification(chr($car)) );
19
20 # retourner le decodage du caractere par decalage a droite si la
       paritee est respectee :
21     return chr( $car >> 1);
22 }
23
24
25
26
```

```
27 my $link = P_open(@ARGV);
28 my $nbrRec = 0; # nbr caracteres recus
29 my $nbrErr = 0; # nbr d'erreurs
30 my $nbrErrD = 0; # nbr d'erreurs detectees
31
32 do{
33 my $carRec = P_recoitCar($link);
34
35 # on test la paritee de cette reception (on test si la reception est
       valide) :
36    $nbrErrD+=1 unless (verification($carRec)); # la reception ne
         respecte pas la parite
37
38 # on test la validite de ce caractere recu :
39    $nbrErr+=1 if (decodage($carRec) ne 'o'); # le caractere n'est
         pas celui attendu
40
41    $nbrRec += 1;
42 }while ($nbrRec < 100000);
43
44
45 print "\n";
46 print " -> nombre de receptions : ".$nbrRec."\n";
47 print " -> nombre de receptions supposees bonnes : ".($nbrRec-
       $nbrErrD)."\n";
48 print " -> nombre de vrais bons caracteres : ".($nbrRec-$nbrErr)."\n
       ";
49 print " -> nombre d'erreurs au total : ".$nbrErr."\n";
50 print " -> nombre d'erreurs detectees : ".$nbrErrD."\n";
51 print " -> nombre d'erreurs non detectees : ".($nbrErr-$nbrErrD)."\n
       ";
52 # attention a la division par "0" ! xD
53 print (" -> fiabilitee de l'envoi/reception : ".(100-100*$nbrErr/
       $nbrRec)."%\n") if($nbrRec != 0);
54 print (" -> fiabilitee de detection d'erreur : ".(100*$nbrErrD/
       $nbrErr)."%\n") if($nbrErr != 0);
55 print "\n";
56
57 P_close($link);
```

```
1 #!/usr/bin/perl
2
3 use strict;
4 use Physique::LinkUDP;
5 use Physique::Noise; # pour generer des problemes et erreurs de
       transmission
6 use Parity;
7
8 sub codage {
```

```
9     my ($car) = @_;
10
11  # ajout d'un zero a droite de la valeur binaire du caractere (
        decalage a gauche) :
12    $car = ord($car) << 1;
13
14  # application de la paritee si ce caractere "impair", on inverse le
        dernier bit qui devient donc un "1" :
15    $car = $car ^ 1 if Parity::parity($car);
16
17  # on retourne le caractere et non pas sa valeur binaire :
18    return chr($car);
19  }
20
21
22
23  my $car = 'o';
24  my $link = P_open(@ARGV);
25  my $nbrEnv = 0;
26  my $temp = codage($car); # on ne l'encode qu'une seule fois puisque
        ce caractere ne change pas
27
28  print "\n -> envoi du caractere : ".$car." \n";
29
30  while($nbrEnv<100000)
31  {
32    P_envoiCar($link,$temp);
33    $nbrEnv+=1;
34  }
35
36  print "\n -> nombre de caracteres envoyes : ".$nbrEnv."\n\n";
37
38  P_close($link);
```

# Table des figures