LES CODES CORRECTEURS

MODELISATIONS MATHEMATIQUES

Yoann Gathignol – Titouan Bouete-Giraud – Guillaume Robert
Groupe 1A
2016-2017

TABLE DES MATIÈRES

Contenu

Introduction

L'origine du codage correcteur d'erreurs remonte à la fin des années 40. Ce-dernier permet de transmettre de façon fiable de l'information, qui est codée grâce à des mots binaires d'une longueur donnée, envoyés sur des lignes plus ou moins bruitées. Cette transmission présente un risque d'erreur variable selon les cas, le principe est donc de trouver un moyen de les corriger à la réception. Cependant on observe plusieurs contraintes : la redondance et le temps d'occupation de la ligne. En effet afin de procéder à la correction, une redondance de l'information est quasi-obligatoire. C'est pourquoi il faut "optimiser" cette redondance pour qu'elle n'occupe pas trop longtemps la ligne de transmission. Les messages sont transmis en binaire {0,1}.

Les premières personnes à avoir travaillés sur ce sujet sont principalement Golay et Hamming (qui donneront leur nom à des codes).

De nos jours on retrouve les codes correcteurs dans tous les réseaux, à des niveaux techniques plus ou moins complexes. Aussi, ces codes sont retrouvés à un niveau sophistiqué dans les sondes spatiales, les systèmes de guidage, les lecteurs de disques numériques et de disques compacts.

I. Probabilité des erreurs

La probabilité p qu'un bit soit mal transmis (signe de bit différent à la réception) doit être mesurée. Pour cela, il suffit de transmettre un certain nombre de données, puis de comparer les résultats afin d'obtenir un ratio nombre d'erreur/quantité de données transmise. Cette probabilité peut fortement varier en fonction du mode de transmission et d'éventuels éléments perturbateurs (bruits de fond pour transmission téléphonique, rayures sur un disque…). On estime cependant que de manière générale, cette probabilité relève de l'ordre de 10^{-6} (1 erreurs pour 1 000 000 de bits transmis).

Cette probabilité est au cœur de calculs plus avancés. Par exemple, on peut déterminer la probabilité de n'avoir aucune erreur dans un message grâce à la loi binomiale :

$$\mathbb{P}(X = k) = \binom{n}{k} p^k (1-p)^{n-k}.$$

k = nombre de bits transmis correctement

n = nombre total de bits transmis

p = probabilité d'erreur de transmission sur 1 bit

Afin de déterminer l'efficacité d'un code correcteur, il est nécessaire de quantifier 2 éléments :

- la capacité du code à détecter les erreurs éventuelles
- la capacité du code à corriger ces erreurs sans renvois de l'information

Cependant, dans le second cas, il peut s'avérer préférable de renvoyer l'information plutôt que d'allouer de nouvelles ressources à sa correction. Bien qu'il existe tout de même des cas où la transmission de l'information peut s'avérer très coûteuse (ex : satellites) et pour lesquels on cherchera à un implémenter un code correcteur très efficace, dans la majorité des cas, on utilisera un code plus léger.

Les codes les plus simples et les plus connus et utilisés aujourd'hui sont : le code de parité, le CRC, le code de répétition.

II. Le code de parité

1. PRESENTATION

Le *code de parité* ou VRC (*Vertical Redundacy Check*) est un code simple qui consiste à ajouter un bit à la fin du mot transmis, ce bit indique la "parité" du mot. La parité d'un message varie si le nombre de bits à 1 dans le mot est pair ou non. Il est donc à 0 pour un nombre pair de "1" et à 0 pour un nombre impair de "1" dans le mot envoyé. Le récepteur vérifie donc la parité du message et la compare à la valeur du bit de parité, il permet donc de détecter un nombre **impair** d'erreur dans le mot.

Ce code a cependant ses limites : il ne permet de détecter qu'un nombre impair d'erreurs, car en changeant un nombre pair de bits dans le mot on retrouverait la parité attendue. Ce code ne permet pas de localiser l'erreur mais seulement de détecter son existence, il faudra donc que le récepteur du message demande le renvoi du message de l'auteur.

Ce code est principalement utilisé dans les mémoires RAM des ordinateurs, ajoutant un bit additionnel à chaque mot de la RAM. Les constructeurs d'ordinateurs choisissent ce qu'ils préfèrent : contrôle de parité ou d'imparité, portant sur les 0 ou les 1, l'efficacité est la même.

2. PROBABILITE DE DETECTION

Comme nous avons pu le voir dans l'introduction, il y a en moyenne 1 erreur pour 1 000 000 de bits transmis, ce qui nous donne une probabilité très faible d'avoir une erreur, soit environ 10^{-6} pour la transmission moyenne en conditions "parfaites", c'est à dire une transmission sans aucun bruit.

Ce code étant basé sur une logique primaire des mathématiques qui est celle de la parité, la probabilité de détection lorsque qu'il y aurait un nombre impair de bits faussés dans le mot est de 100%.

Cependant pour exactement la même raison, il est totalement impossible de détecter une erreur dans le cas où il y aurait un nombre pair d'erreur, car cela ne changera pas la parité du nombre de bits à 1 du mot.

Dans le cas d'une probabilité d'erreur de 10^{-6} par bit :

$$P(\textit{Transmission "parfaite"}) = (1 - 10^{-6})^{\wedge 8} \simeq 0.999$$

$$P(\textit{Mot erroné}) = 1 - P(\textit{Transmission "parfaite"})$$

La probabilité d'avoir un mot de 8 bit contenant une erreur est faible à l'extrême, ce qui rend difficile le calcul des probabilités.

Les mots contenant un nombre impair d'erreur étant indétectables par le code, nous en déduisons la probabilité suivante :

Pour un mot de 8 bits :

$$P(\textit{Mot erroné détecté}) = P(\textit{1 erreur}) + P(\textit{3 erreurs}) + P(\textit{5 erreurs}) + P(\textit{7 erreurs})$$

Et par conséquent :

$$P(\textit{Envoi et détection d'un mot erroné}) = \frac{\boldsymbol{P(Mot\ erroné\ détecté)}}{\boldsymbol{P(Mot\ erroné)}}$$

3. EXEMPLE

Nous allons illustrer ici les 2 cas abordés dans la partie ci-dessus.

- **1er cas, nombre impair d'erreurs :**

Admettons que nous voulons transmettre le nombre "**231**", après conversion en binaire celui-ci est : **11100111**. L'émetteur applique le code de parité et rajoute donc un "**0**" à la fin du message (car le nombre de bits à 1 est pair).

Message envoyé : **111001110** ← en rouge : bit de parité

Message reçu : **101111110** ← en orange : bits faussés

Le récepteur reçoit et analyse le mot : 7 bits sont à 1, or le bit de parité est à 0, le récepteur détecte donc directement l'erreur et procède au processus de correction (application d'un autre code correcteur, demande de renvoi du mot…)

- **2ème cas, nombre pair d'erreurs :**

Admettons que nous voulons transmettre le nombre "**231**", après conversion en binaire celui-ci est : **11100111**. L'émetteur applique le code de parité et rajoute donc un "**0**" à la fin du message (car le nombre de bits à 1 est pair).

Message envoyé : **111001110** ← en rouge : bit de parité

Message reçu : **100111110** ← en orange : bits faussés

Le récepteur reçoit et analyse le mot : 6 bits sont à 1, et le bit de parité est à 0, le récepteur ne détecte pas que le mot reçu contient des erreurs.

4. RENDEMENT

Comme nous l'avons vu précédemment, ce code ne permet de détecter qu'un nombre impair d'erreurs, cependant la probabilité d'avoir qu'une erreur étant déjà très faible, celle d'en avoir deux est d'autant plus faible. Nous pouvons donc estimer que le rendement de ce code est très bon voire excellent en effet le rendement est le suivant :

Rendement = k/ (k-1)

La variable k étant la taille du message. Par exemple pour un message codé sur 7 bits, le rendement est de 87,5%. D'autant plus c'est un code extrêmement léger car il ne rajoute qu'un seul bit à chaque mot codé.

III. Le code de répétition

1. PRESENTATION

Le code de répétition consiste, comme son nom l'indique, à répéter l'information en la dupliquant. En informatique, cela consiste à dupliquer tous les bits transmis. Le nombre de répétition de chaque bit peut varier en fonction de plusieurs paramètres :

- Le poids final de l'information : on multiplie chaque bit, on multiplie donc le poids total de l'information
- la probabilité de détection : plus il y a de répétition, plus on est sûr de détecter l'information
- correction des erreurs : à partir de 3 répétitions, il est possible de corriger les erreurs, mais il est probable que dans certains cas (probabilité minime), on valide celle-ci.

2. PROBABILITE DE DETECTION

Pour détecter les erreurs avec ce code, on va découper l'information reçue en blocs de bits de taille équivalentes au nombre de répétition (si chaque bit est répété 4 fois, alors on utilise des blocs de 4 bits). On vérifie ensuite que tous les blocs contiennent bien des bits de signe identique. Si un bloc contient des bits de signe différents, alors il y a une erreur au sein de ce bloc. Ce code détecte donc les erreurs tant que l'intégralité des bits au sein d'un bloc ne change pas de signe. En effet, dans le cas 3 répétitions, si les 3 bits au sein d'un bloc changent, alors l'erreur n'est pas détecter. Cependant, comme vue précédemment, la probabilité d'avoir une erreur est déjà très faible (de l'ordre de 10^{-6}), la probabilité d'avoir deux erreurs à la suite est donc tellement insignifiante que 2 répétitions seront généralement acceptables pour obtenir un taux de détection des erreurs proche de 100%.

3. CORRECTION DES ERREURS

Comme nous venons de le voir, 2 répétitions permettent de détecter quasiment toutes les erreurs. Mais ce code permet aussi de corriger les erreurs dans certains cas. En effet, s'il y a une erreur lors de la transmission, celle-ci est non seulement détectée mais aussi localisée. Si l'on détecte que l'un des paquets comporte une erreur, on peut alors chercher à la rectifier en conservant le signe majoritaire au sein de ce paquet. C'est là que le nombre de répétition entre en jeu. Dans le cas de 2 répétitions, le code détecte qu'il y a une erreur au sein du paquet, mais pas le bit sur lequel elle se trouve. On peut donc aléatoirement inverser le signe de l'un des deux bits, mais cela nous laisse déjà avec une probabilité de récupérer le message initial de 50%. Supposons qu'il y ait plusieurs erreurs, et cette probabilité se réduit encore plus. Dans ce cas, la probabilité de récupérer le message initial est de :

½^(n-k)

n le nombre de bit transmis,

k le nombre de bit transmis correctement.

Cependant, si l'on augmente le nombre de répétitions, on augmente fortement l'efficacité de la correction. Encore une fois, la probabilité d'avoir plusieurs erreurs à la suite est très faible. Dans la quasi majorité des cas, une erreur sera donc seule au sein d'un paquet. Une méthode efficace pour rectifier celle-ci consiste donc à repérer le signe dominant (majoritaire) au sein d'un paquet, puis changer le signe des bits qui ne respectent pas cette uniformité. C'est pour cette raison que 3 répétitions sont largement suffisantes pour une détection et une correction de quasiment 100%.

4. RENDEMENT

Bien que reposant sur un concept très simple, le code de répétition s'avère être un code très efficace, non seulement pour détecter, mais aussi pour rectifier les erreurs. Malheureusement, cette efficacité à un coût très lourd, puisque l'on multiplie l'intégralité des données par le nombre de répétitions désiré. Cela veut dire que les messages transmis peuvent être très lourds. En revanche, nous pouvons être quasiment sûrs que ces messages une fois reçus soit entièrement justes. Cela veut dire que dans le cas où un message prendrait longtemps à parvenir à sa destination, ce code s'avère très intéressant. Pour des usages plus conventionnels en revanche, on préférera des codes plus légers.

Evolution de la quantité d'information à transmettre :

n*k

k le nombre de bits à transmettre,

n le nombre de répétition souhaité.

Le rendement de ce code se mesure donc :

k/nk = 1/n

5. EXEMPLE

Pour un nombre de répétition de 3, le message suivant devient :

0 1 1 1 0 0 1 0

0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 0 0 0

Comme on peut le voir, tous les bits sont dupliqués 3 fois. On constate la présence de paquet de 3 bits identiques à la suite.

Supposons qu'une erreur soit apparue lors de la transmission :

0 0 0 1 1 1 1 1 0 1 1 1 0 0 0 0 0 0 1 1 1 0 0 0

Nous avons ici un paquet de 3 bits avec le 3e bit différent, l'erreur est donc détectée. Pour que cela ne soit pas le cas, il aurait fallu que les 3 bits du paquet soient inversé.

De plus, on regardant le paquet, on constate que deux bits sont de signe 1, alors que le troisième est de signe 0. On peut donc corriger l'erreur en appliquant le signe majoritaire des bits du paquet au dernier bit. Dans le cas où deux des trois bits auraient changé de signe, l'erreur n'aurait pas était rectifiée, mais la détection se serait quand même faîte.

IV. Le CRC

1. PRESENTATION

Le code CRC (Cyclic Redundancy Code), contrôle de redondance cyclique, consiste à protéger des blocs de données, appelés *trames.* A chaque trame est associé un bloc de données, appelé *code de contrôle* Le code CRC contient des éléments redondants vis-à-vis de la trame, permettant de détecter les erreurs. Le CRC est donc la méthode principale de détection d'erreur dans la télécommunication.

Afin d'utiliser cette méthode il nous faut choisir un polynôme générateur, qui sera connu de l'émetteur et du récepteur.

L'émetteur :

- Il va diviser le message à transmettre par ce code (en bit).
- Le reste de cette division est le CRC.
- Il suffit maintenant de le concaténer au message à transmettre et de l'envoyer.

Le récepteur :

- Divise le message reçu par le polynôme (toujours en bit)
- Si le reste est 0 alors il n'y a pas d'erreur
- Sinon il y a des erreurs dans le message reçu

Il existe "plusieurs" CRC qui varient en fonction du polynôme choisi : CRC CCIT v41, CRC ARPA, CRC 12, CRC16, CRC 32. Par exemple, "Ethernet" utilise un champ CRC à 32 bits, la compression ZIP utilise un CRC à 16 ou 32 bits. Voici quelques exemples de polynômes générateurs :

- Polynôme générateur CRC-16 : $x^{16} + x^{15} + x^2 + 1$

- Polynôme générateur CRC-12 : $x^{12} + x^{11} + x^3 + x^2 + x + 1$

- Polynôme générateur CRC ARPA : $x^{24} + x^{23} + x^{17} + x^{16} + x^{15} + x^{13} + x^{11} + x^{10} + x^9 + x^8 + x^5 + x^3 + 1$

2. PROBABILITE DE DETECTION

On peut avec n=16 détecter toutes les erreurs simples et doubles, toutes les erreurs comportant un nombre impair de bits et tous les paquets d'erreur de longueur inférieure ou égale à 16 et, avec une très bonne probabilité, les paquets d'erreurs de longueur supérieure. Ce code permet donc de détecter toutes les erreurs d'ordre inférieur au degré du polynôme générateur. De ce fait, la probabilité de ne pas détecter les erreurs est très faible.

3. RENDEMENT

La taille du message influe sur le rendement de ce code :

$$Rendement = \frac{Tailledumessage}{Tailledumessage + Degrdupolynme}$$

Lorsque la taille d'un message est conséquente le rendement du CRC 16 devient très intéressant. Alors que si le message est petit, le CRC 16 est moins intéressant.

Illustration:

- CRC 16 pour un code de 9 bit :

$$Rendement = \frac{9}{9+16} \approx 0.36$$

Rendement = 36%

Le rendement n'est pas très important.

- CRC 16 pour un code à 170 bits :

$$Rendement = \frac{170}{170+16} \approx 0.91$$

Rendement = 91%

Ce rendement est donc très efficace pour les longs messages. Cependant le meilleur moyen de corriger l'information quand on est certain qu'il y a des erreurs serait de le renvoyer.

4. EXEMPLE

Emission et codage :

Admettons que nous voulons transmettre le nombre “231”, après conversion en binaire celui-ci est : 11100111. Avec un polynôme générateur : $x^4 + x^2 + x$ qui nous donne 10110.

On ajoute *m* 0 (le degré du polynôme) au mot binaire à transmettre : 11100111 00000**.** Il faut maintenant procéder à une division polynomiale entre le mot à transmettre et le polynôme : $\frac{11100111}{10110}$.

Le reste de cette division correspond au CRC qui est, ici, 1110. On l’ajoute donc à la fin du message : 11100111 1110**.**

Réception et décodage :

Le message étant reçu il faut donc procéder au décodage afin de vérifier si une erreur s’est introduite. Comme à la première étape, nous allons diviser le message reçu (c’est à dire le contenu + le CRC) par le polynôme : $\frac{111001111110}{10110}$. Le reste de la division est nul, le message a donc été transmis sans erreur.

V. Hamming

1. PRESENTATION

Les codes de Hamming représentent une famille de code, si un code fait partie de cette famille alors il jouit de plusieurs caractéristiques. Par exemple la forte probabilité de détection des erreurs. Un code de Hamming est parfait, c'est à dire qu'il atteint le rendement maximum pour des codes avec une taille donnée et avec une distance minimum de 3. Le code de Hamming est un code linéaire qui a la particularité de permettre la détection et la correction des erreurs de transmissions. La distance de Hamming est une notion mathématique qui est utilisée en information, en traitement du signal et dans les télécommunications. Cette distance entre deux séquences binaires de même taille est égale au nombre de bits de rang identique par lesquels elles diffèrent. Par exemple : d(1**01**0110, 1**10**0110) = 2. C'est donc la distance minimale qui sépare deux mots valides du code. On a donc : d = min {d(x,y): x,y appartenant à C, x ≠ y}. Un code de Hamming est donc structuré de cette manière :

- Les *m* bits du message à transmettre et les *n* bits de contrôle de parité,
- Longueur totale : 2^n-1,
- Longueur du message : m = (2^n-1)-n.

Le poids de Hamming est le nombre de différences dans le code, c'est le poids de l'erreur. Nous avons un mot de *n* bits à transmettre. On ajoute un *n* + 1 ème bit afin qu'en tout il y ai un nombre pair de 1. Ce bit ajouté est le bit de parité que nous avons expliqué auparavant.

Rappel bit de parité :

0 1 0 1 1 1 1 0 1 ⇒ 0 1 0 1 1 1 1 0 1 | **0**

0 1 0 1 1 1 0 0 1 ⇒ 0 1 0 1 1 1 0 0 1 | **1**

Certains de ces codes nous permettent de corriger une seule erreur. Il existe d'autres codes permettant de corriger plusieurs erreurs :

- Reed et Muller
- Codes de Résidus Quadratiques
- Codes BCH
- Codes de Reed-Solomon
- Codes de Goppa

Cependant ces codes sont beaucoup plus complexes.

2. RENDEMENT

Il y a plusieurs rendements selon le code choisi :

- Un mot de code 7 - 4 a un coefficient d'efficacité de 4/7 = 57%
- Un mot de code 15-11 a un coefficient d'efficacité de 11/15 = 73%
- Un mot de code 31-26 a un coefficient d'efficacité de 23/31 = 83%

Conclusion

Au final, lorsque l'on parle de code correcteur, il faut garder à l'esprit que les erreurs restent très rares, et en conséquent, une grande partie de l'information ajoutée par le code détecteur/correcteur choisi sera inutilisée. Par conséquent, le choix du code dépendra énormément des conditions dans lesquels se passe la transmission de cette information, tel que le débit, la facilité à renvoyer cette information, ou même l'impact global des erreurs sur le message final.

Résumé

Nous avons abordés différents codes détecteurs, correcteurs pour certains. Le code de répétition, malgré un principe très simple (duplication de l'information), offre une excellente efficacité, mais son rendement est très faible. Le code de parité en revanche, offre un rendement très attrayant, mais son efficacité est limitée. En effet, en ajoutant à chaque octet un bit représentant la somme des autres bits de cet octet, on peut certes détecter les erreurs pour un coût très faible, mais on ne peut pas les corriger. Nous avons aussi vu qu'une bonne utilisation du code CRC permet une efficacité de plus de 90% en ce qui concerne la détection d'erreur, ce qui rend ce code très efficace.

Abstract

We have addressed various detector codes, correctors for some. The repetition code, despite a very simple principle (duplication of information), is extremely reliable but its efficiency is very low. The parity code, on the other hand, offers a very attractive cost in data, but its effectiveness is limited. By adding to each byte a bit representing the sum of the other bits of this byte, it is possible to detect the errors at a very low cost, but they cannot be corrected. We also saw that a good use of the CRC code allows an efficiency of more than 90% in terms of error detection, which makes this code very effective.