

Codage, décodage et optimisation d'une famille de codes correcteurs d'erreurs

Sujet proposé par Jean-Pierre Tillich

Distribué le 11 janvier 2010

1 Quelques mots sur le sujet

Le but de ce projet est de réaliser et de comprendre comment fonctionne un code correcteur d'erreurs très proche des codes correcteurs qui sont standardisés actuellement. La construction, le codage, le décodage et l'optimisation des performances de la famille de codes dont il est question ici font appel à diverses questions d'algorithmique sur les graphes. Il comporte à la fois des questions de programmation et des questions d'algorithmique sur les graphes.

La réponse aux questions d'algorithmique est très courte. Elle peut cependant demander une certaine réflexion (notamment aux questions 4 et 5). L'implémentation du codage et du décodage est également courte et ne demande pas d'effort particulier. En revanche, la question 9 est une question pour laquelle plusieurs approches sont possibles. Elle peut conduire à un approfondissement assez conséquent.

On pourra trouver sur

<http://www.inference.phy.cam.ac.uk/mackay/codes/gifs/>

un exemple de correction d'erreurs appliquée à une image binaire. Le code utilisé dans cet exemple est très proche de la famille étudiée ici.

Les mises à jour du projet pourront être trouvées à partir du mardi 12 janvier 10h sur

<http://www-rocq.inria.fr/secret/Jean-Pierre.Tillich/enseignement/X2009/INF431/projet.html>

2 La correction d'erreurs

Un code correcteur d'erreurs répond au problème suivant. On veut transmettre sur un canal bruité un message binaire $x_1 \dots x_k$ d'une certaine longueur k . Malheureusement, chacun des bits transmis peut être transformé en son opposé. Plus précisément, on considère dans ce projet le modèle d'erreur suivant que l'on appelle le *canal binaire symétrique de probabilité d'erreur p* . Pour ce modèle, chaque bit transmis est changé en son opposé avec une probabilité p , et ce indépendamment des autres bits transmis. En pratique, p de l'ordre de quelques pour cent.

L'idée des codes correcteurs d'erreurs pour résoudre ce problème est d'abord de transformer le message en question en un message plus long (qui contiendra donc de fait de la redondance) par une application injective f qui va de $\{0, 1\}^k$ dans $\{0, 1\}^n$ avec $n > k$. Cette opération est appelée *l'opération de codage*. Au lieu de transmettre x_1, \dots, x_k , on émet $f(x_1, x_2, \dots, x_k)$. Soit y_1, \dots, y_n le mot binaire reçu. Avec le modèle d'erreur donné plus haut, on vérifie facilement que le mot le plus probable est le mot binaire $z_1 \dots z_n$ dans l'image de f qui est le plus proche au sens de la distance de Hamming du mot reçu $y_1 \dots y_n$. Ceci est vrai tant que $p \leq \frac{1}{2}$, hypothèse que l'on fera toujours dans la suite. Rappelons que la distance de

et \mathbf{M} est une matrice de taille $r \times r$ qui est telle que chaque ligne et chaque colonne contienne exactement 4 "1".

4 Réalisation du codage

Nous nous intéressons dans cette section à comment réaliser le codage pour la famille de codes donnée dans la section précédente.

On montre aisément que le rang (sur \mathbb{F}_2) d'une matrice binaire \mathbf{H} de la forme (1) est égal à $r - 1$ et que la sous-matrice de \mathbf{H} formée par les $r - 1$ premières colonnes est de rang plein. Soit C le code binaire linéaire de longueur $2r$ de matrice de parité \mathbf{H} . En utilisant la remarque sur le rang susmentionnée, on peut montrer qu'il existe une matrice binaire \mathbf{G} de taille $(r + 1) \times 2r$ qui est telle que

$$C = \{\mathbf{x} = (x_1, \dots, x_{2r}) \mid \mathbf{x} = (x_r, \dots, x_{2r})\mathbf{G}\}.$$

En d'autres termes \mathbf{G} est de la forme

$$\mathbf{G} = [\mathbf{P} \mid \mathbf{I}]$$

où \mathbf{I} est la matrice identité de taille $r + 1$ et \mathbf{P} une certaine matrice binaire de taille $(r + 1) \times (r - 1)$.

Cette matrice peut être mise à profit pour réaliser le codage des codes étudiés ici. Ainsi, l'application f définie par

$$\begin{aligned} f : \mathbb{F}_2^{r+1} &\rightarrow \mathbb{F}_2^{2r} \\ (x_1, \dots, x_{r+1}) &\mapsto (x_1, \dots, x_{r+1})\mathbf{G} \end{aligned}$$

sera l'application injective choisie pour l'opération de codage. Il est à noter que le mot binaire (x_1, \dots, x_{r+1}) de longueur $r + 1$ auquel on applique l'opération de codage f apparaît en clair dans les $r + 1$ dernières positions de $f(x_1, \dots, x_{r+1})$.

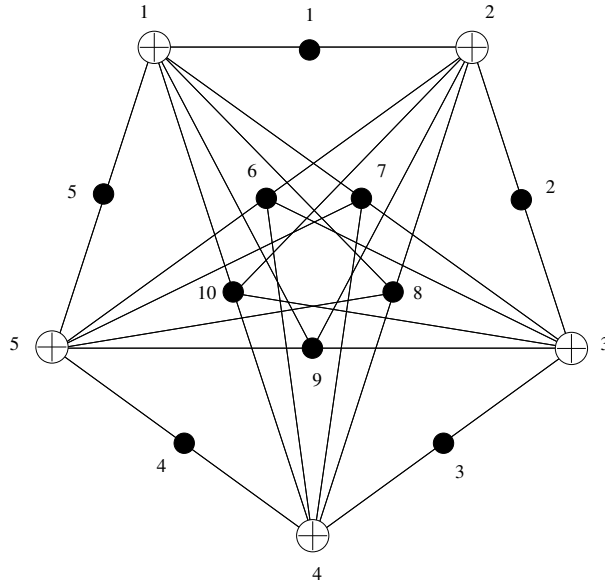
Pour réaliser efficacement les opérations de codage et de décodage, un certain graphe associé à la matrice de parité \mathbf{H} va se révéler particulièrement pertinent. Il s'agit du graphe de Tanner associé à \mathbf{H} et qui est défini par

Définition 1 (graphe de Tanner associé à une matrice de parité) Soit une matrice de parité $\mathbf{H} = (h_{ij})_{\substack{1 \leq i \leq r \\ 1 \leq j \leq n}}$. Le graphe de Tanner associé à cette matrice de parité a pour ensemble de sommets $\{1, 2, \dots, n\} \cup \{\oplus_1, \oplus_2, \dots, \oplus_r\}$. Les sommets de $\{\oplus_1, \dots, \oplus_r\}$ sont appelés les **noeuds de contrôle**. Les sommets de $\{1, 2, \dots, n\}$ sont associés à chaque position du mot de code. On les désigne de manière un peu impropre par les **bits du code**. Les seules arêtes existant dans le graphe relient un bit du code à un noeud de contrôle et il y a une arête entre j et \oplus_i si et seulement si $h_{ij} = 1$.

Exemple : Soit le code de matrice de parité H donné par :

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

et le graphe de Tanner qui lui est associé :



On peut voir ce graphe comme un circuit de vérification pour un code binaire linéaire. En effet, un mot $(x_1, \dots, x_n) \in \mathbb{F}_2^n$ appartient au code de matrice de parité $\mathbf{H} = (h_{ij})_{\substack{1 \leq i \leq r, \\ 1 \leq j \leq n}}$, si et seulement si pour tout noeud de contrôle \oplus_i du graphe de Tanner T associé à \mathbf{H} , la somme (calculée dans \mathbb{F}_2) des x_j où j parcourt l'ensemble des sommets j adjacents à \oplus_i est égale à 0 :

$$\sum_{j \sim \oplus_i} x_j = 0.$$

On note ici par \sim la relation d'adjacence. Ainsi, vérifier qu'un mot (x_1, \dots, x_n) appartient bien au code peut être réalisé en

1. assignant aux noeuds j correspondant aux bits du code j la valeur binaire x_j ;
2. en calculant pour tout noeud de parité \oplus_i la somme des bits du code qui lui sont adjacents et en vérifiant que toutes ces sommes calculées sur \mathbb{F}_2 sont nulles.

Il apparaît que le graphe de Tanner associé à une matrice de parité \mathbf{H} de la forme (1) est donné par un cycle de longueur $2r$ constitué par une alternance de noeuds de parité et bits de code auquel on rajoute r bits de code qui sont adjacents chacun à 4 noeuds de parité du cycle (et chaque noeud de parité est adjacent à 6 bits de code au total). On peut mettre à profit cette structure du graphe (et l'interprétation du graphe de Tanner comme un circuit de vérification) pour répondre à la question suivante.

Question 1 *Montrer que l'opération de codage peut être réalisée avec une complexité en $O(r)$ en n'effectuant que $O(r)$ additions dans \mathbb{F}_2 .*

Question 2 *Implémenter cette opération de codage. Utiliser une représentation du code précédent qui soit de taille $O(r)$.*

5 Un algorithme de décodage

L'opération de décodage consiste à trouver le mot de code le plus proche du mot binaire reçu au sens de la distance de Hamming. Pour un code C de longueur n , un indice i dans $\{1, 2, \dots, n\}$ et un nombre binaire ϵ dans $\{0, 1\}$, on définit le sous-ensemble $C_i(\epsilon)$ comme l'ensemble des mots de code de C valant ϵ en position i :

$$C_i(\epsilon) \stackrel{\text{def}}{=} \{(x_1, \dots, x_n) \in C \mid x_i = \epsilon\}.$$

Enfin pour un mot reçu $\mathbf{y} = (y_1, \dots, y_n)$ et un indice i dans $\{1, 2, \dots, n\}$, on définit les quantités suivantes

$$\Delta_i \stackrel{\text{def}}{=} d(\mathbf{y}, C_i(1)) - d(\mathbf{y}, C_i(0))$$

où l'on définit pour un sous-ensemble A de \mathbb{F}_2^n :

$$d(\mathbf{y}, A) \stackrel{\text{def}}{=} \min_{\mathbf{x} \in A} d(\mathbf{y}, \mathbf{x}).$$

Question 3 Expliquer comment la connaissance des Δ_i peut être mise à profit pour trouver le(s) mots de code \mathbf{x} le(s) plus proche(s) de \mathbf{y} . Quelle est la complexité de l'algorithme retrouvant cet ensemble de mots de code à distance minimale de \mathbf{y} prenant en entrée cet ensemble de Δ_i ?

On va s'intéresser maintenant à comment calculer efficacement Δ_i quand le graphe de Tanner T du code est un arbre. Soit un sommet i_0 de T , soient j_1, \dots, j_k les noeuds de parité adjacents à i_0 et soient i_1, \dots, i_l les bits de code adjacents aux noeuds de parité j_1, \dots, j_k autres que i_0 (voir figure 1). Pour

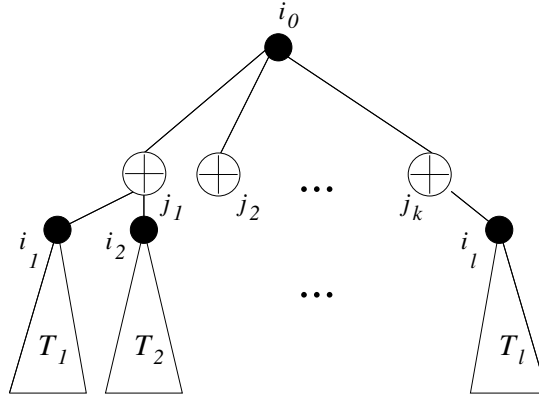


FIG. 1 –

$m \in \{1, \dots, l\}$, on appelle T_m le sous-arbre de T engendré par les noeuds que l'on peut relier à i_m par un chemin ne passant par aucun des j_1, \dots, j_k . En d'autres termes, si l'on enracine T en i_0 , c'est le sous-arbre de T contenant i_m (voir figure). Ces arbres T_m engendrent chacun un code linéaire binaire C^m : c'est tout simplement le code de matrice de parité qui est associée à T_m . On définit Δ_{i_m} de manière analogue aux Δ_i précédents, mais cette fois par rapport au code C^m et au sous-mot \mathbf{y}^m de \mathbf{y} constitué des positions apparaissant dans le sous-arbre T_m :

$$\Delta_{i_m} \stackrel{\text{def}}{=} d(\mathbf{y}^m, C_{i_m}^m(1)) - d(\mathbf{y}^m, C_{i_m}^m(0)),$$

En revanche, Δ_{i_0} reste défini par rapport au code C d'origine et par rapport à \mathbf{y} :

$$\Delta_{i_0} \stackrel{\text{def}}{=} d(\mathbf{y}, C_{i_0}(1)) - d(\mathbf{y}, C_{i_0}(0)),$$

Question 4 Donner une expression de Δ_{i_0} en fonction de $\Delta_{i_1}, \dots, \Delta_{i_l}$ et de la coordonnée i_0 du mot reçu, c'est à dire y_{i_0} . En déduire un algorithme de complexité polynomiale calculant tous les Δ_i quand T est un arbre.

La complexité de l'algorithme que vous venez de donner est probablement en $O(n^2)$, où n est la longueur du code. On peut se convaincre que les calculs effectués pour obtenir tous les Δ_i calculent de nombreuses fois les mêmes quantités. Il est possible de factoriser ces calculs et d'obtenir un algorithme qui est capable de calculer les Δ_i avec une complexité en $O(nD)$ où D est le diamètre du graphe de Tanner. Ceci représente une amélioration substantielle dans de nombreux cas en pratique et présentera l'avantage de pouvoir être généralisé au cas où le graphe de Tanner n'est plus un arbre. Cet algorithme consiste à envoyer une série de messages sur les arêtes du graphe de Tanner. Il prend la forme suivante

Algorithme 1 MIN-SUM

INPUT: $\mathbf{y} = (y_1, \dots, y_n) \in \mathbb{F}_2^n$ **OUTPUT:** $(\Delta_i)_{1 \leq i \leq n}$

{Initialisation :}

for all arêtes $i - \oplus_k$ **do**

$$e_{i \rightarrow \oplus_k}^{(0)} = (-1)^{y_i}$$

{Itérations :}

for $t = 1$ to r **do** {ici $r = \lceil D/2 \rceil$, où D est le diamètre du graphe de Tanner}**for all** arêtes $\oplus_k - i$ **do** $e_{\oplus_k \rightarrow i}^{(t)}$ = fonction des $e_{l \rightarrow \oplus_k}^{(t-1)}$ où l parcourt l'ensemble des bits de code adjacents à \oplus_k à l'exception de i .**for all** arêtes $i - \oplus_k$ **do** $e_{i \rightarrow \oplus_k}^{(t)}$ = fonction de y_i et des $e_{\oplus_j \rightarrow i}^{(t)}$ où \oplus_j parcourt l'ensemble des noeuds de parité adjacents à i à l'exception de \oplus_k .

{Fin des itérations}

for $i = 1$ to n **do**

$$\Delta_i = (-1)^{y_i} + \sum_{\oplus_k \sim i} e_{\oplus_k \rightarrow i}^{(r)}$$

Question 5 Compléter les deux fonctions inconnues dans l'algorithme précédent.

L'algorithme précédent peut bien évidemment être effectué que le graphe de Tanner soit un arbre ou non. C'est l'algorithme que vous allez utiliser avec une valeur de r que l'on va fixer à 50 (et non plus fixé à la moitié du diamètre du graphe) pour la famille de code étudiée.

Question 6 Rajouter à l'étape finale une opération consistant à partir du calcul de Δ_i à donner la valeur qui vous semble la plus raisonnable pour x_i où x_i est la i -ème coordonnée du mot de code le plus proche de \mathbf{y} . Implémenter cet algorithme de décodage.

6 Une première simulation

Question 7 Réaliser une première implémentation de la famille de codes correcteurs en choisissant aléatoirement la matrice \mathbf{M} de (1) pour les longueurs $n = 1000$ et $n = 8000$. Effectuer une simulation de ces codes pour différentes valeurs du bruit p (c'est la probabilité mentionnée dans la section 2) en faisant varier p de 0,01 à 0,08. Il s'agit ici de

1. de produire N mots de code ;
2. de les bruiteur en modifiant chacun des bits avec probabilité p ;
3. de tester si l'algorithme de décodage réalisé dans la section 5 réussit à décoder. Soit N_{err} le nombre de mots de codes pour lequel le décodage échoue.
4. Le rapport $\frac{N_{err}}{N}$ est une estimation de la probabilité d'erreur après décodage.

Estimer pour quelle probabilité d'erreur p la probabilité d'erreur après décodage vaut 10^{-3} et 10^{-4} pour les longueurs $n = 1000$ et $n = 8000$.

7 Optimisation de la structure du code

Il est probable que le choix aléatoire des codes précédents ne conduise pas à des codes ayant des performances exceptionnelles. Il y a deux raisons à cela :

- le graphe de Tanner du code construit peut avoir des petits cycles (il peut par exemple y avoir des cycles de taille 4). L'algorithme de décodage utilisé est construit pour un code avec un graphe de Tanner sans cycle. On peut montrer que les petits cycles détériorent les performances du décodage.
- il peut y avoir des mots de code de poids de Hamming faible. Là aussi, on montre que cela détériore les performances.

Question 8 *Proposer et implémenter un algorithme qui à partir d'un graphe de Tanner associé à une matrice de la forme (1) produit un nouveau graphe de Tanner d'une matrice ayant la même forme et les mêmes dimensions mais sans cycles d'une taille L donnée. On s'attachera notamment aux valeurs $L = 4$ et $L = 6$.*

Indication : On pourra par exemple réaliser un algorithme trouvant tous les cycles d'une longueur L donnée et "casser" ces cycles par une petite transformation du graphe qui conserve la forme (1) de la matrice de parité associée.

Question 9 *Proposer et implémenter un algorithme qui produit tous les mots de code de poids de Hamming t donné pour un code ayant une matrice de parité de la forme (1).*

Il est essentiel dans cette question de proposer un algorithme dans cette question qui soit beaucoup plus efficace (pour des valeurs de t petites) que de tester tous les mots de poids de Hamming t et de longueur n (n étant la longueur du code) appartiennent bien au code. De nombreuses approches peuvent être essayées ici...

Question 10 *Proposer et réaliser un algorithme qui cherche à produire un code appartenant à la famille de codes étudiée dans ce projet n'ayant pas de mot de code non nul de poids faible et dont le graphe de Tanner ne contienne pas de cycles de longueur faible (4 étant à éviter à tout prix). Par exemple, arriverez-vous à produire un code en longueur $n = 1000$ n'ayant pas de mot de code non nul de poids de Hamming inférieur à 20 ? Recommencer les simulations de la section précédente avec un tel code.*

8 Extensions possibles

De nombreuses extensions sont possibles à ce projet. On peut par exemple chercher à améliorer encore les performances des codes de cette famille

- en modifiant la structure de la matrice \mathbf{M} . Il est notamment possible de changer le poids des lignes et des colonnes et de le rendre irrégulier et d'améliorer encore les performances au décodage.
- en modifiant l'algorithme de décodage lui-même. Cela peut se faire par exemple en modifiant l'initialisation dans l'algorithme 1, en choisissant les deux fonctions non spécifiées durant les itérations et l'étape finale de manière à calculer pour chaque position les probabilités que ce bit vaille 0 connaissant le mot reçu. Cette probabilité peut ensuite être mise à profit pour décider si le bit vaut 0 ou non.