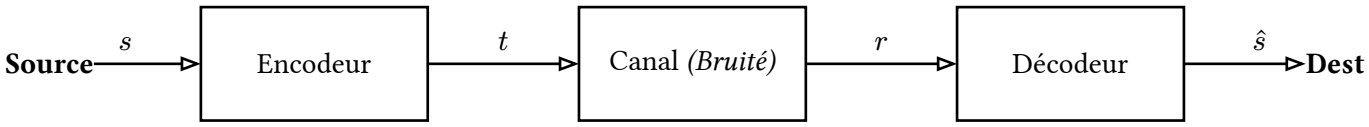
Codes LDPC

Notes pour TIPE

Introduction à la théorie de l'information

Problème principal : il y a du bruit dans les transmissions mais on veut pas d'erreurs.
Au lieu de trouver des modifications physiques on va creer des solutions pour corriger les erreurs.
Il faut un **encodeur** qui ajoute de la redondance et un **décodeur**.



Code correcteur d'erreurs pour un cannal binaire symetrice [1]

Le but est de transformer un cannal bruité en cannal fiable avec un coups de calculs en plus (encodeur / décodeur).
On vas chercher la meilleur performance de correction d'erreurs. Ce sont les limites théoriques que cherchent à trouver la *Théorie de l'information*. Pas de retransmissions.

Codes de répétition

Définitions

Il s'agit ici de répter tous les bits. Un message source $\boldsymbol{s}$, un message transmit $\boldsymbol{t}$, un vecteur de bruit $\boldsymbol{n}$ et un message recu $\boldsymbol{r}$. $\boldsymbol{r = t + n}$.

$$\begin{array}{ll} s = & \overbrace{0}^{} \quad \overbrace{1}^{} \\ t = & \overbrace{000}^{} \quad \overbrace{111}^{} \\ n = & 001 \quad 010 \\ r = & 001 \quad 101 \end{array}$$

On décode en choisissant le bit le plus présent dans un bloc.
C'est ce que représente le *Likelihood ratios* : $\frac{P(r \mid s=1)}{P(r \mid s=0)}$
Ici $001 \to 0$ et $101 \to 1$ donc $\hat{s} = 0\ 1$.
Preuve de l'optimalité (dans le sens la plus faible probabilité d'erreurs) [1] p6.
Problème : trois fois plus de bande passante…

Single parity check code et définitions algébriques

Ajout d'uniquement 1 bit d'information à la fin sur la parité du nombre de 1.
Un message $\boldsymbol{s}$ est de la forme

$$s = [s_1\ s_2\ s_3\ s_4\ s_5\ s_6]$$

où $c_i \in \{0, 1\}$ et le *codeword* vérifie la contraine si

$$s_1 \oplus s_2 \oplus s_3 \oplus s_4 \oplus s_5 \oplus s_6 = 0 \qquad \text{E}$$

parity-check equation.
Inversion d'un nombre bit paire $\Rightarrow$ E = 0 donc aucune erreur détécté.
C'est donc pas assez puissant pour savoir quel bit à changé.

On écrit sous forme matricielle.

$$Hs^T = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

avec H la matrice *parity-check* où chaque ligne de H correspond à l'équation de parité et chaque colonne de H correspond à un bit du *codeword*.

Les contraines sont alors les suivantes

$$s_4 = s_1 \oplus s_2 s_5 = s_2 \oplus s_3 s_6 = s_1 \oplus s_2 \oplus s_3$$

De plus s_4, s_5, s_6 sont les bits de parités et :

$$s = [s_1 s_2 s_3 s_4 s_5 s_6] = [s_1, s_2, s_3] \underbrace{\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}}_{G}$$

où $\boldsymbol{G}$ est la matrice génératrice du code.

On note $u = [u_1, ..., u_k]$ où u contient les k bits du message, ici $u = [u_1, u_2, u_2]$

$$s = uG$$

Pour un message de longueur k et n *codewords*, $G \in M_{k \times n}(\mathbb{Z}/2\mathbb{Z})$.

De plus $\frac{k}{n}$ est le *rate* du code.

Un code de taille k contient 2^k *codewords*. Ces *codewords* sont des sous-ensembles avec 2^n vecteurs de taille n possibles.

On peut obtenir H sout la forme

$$H = [A, I_{n-k}]$$

avec $A \in M_{(n-k) \times k}(\mathbb{Z}/2\mathbb{Z})$ et donc

$$G = [I_k, A^T]$$

De plus si G est la matrice génératrice pour un code avec matrice de parité H alors

$$GH^T = 0$$

G est orthogonal à H.

Un code peut avoir autant de contraites *parity-check* qu'il veut mais seulement $n - k$ d'entre elle seront linéairement independantes. C'est à dire :

$$n - k = \text{rg}(H)$$

Voir [2]

Comment détécter et corriger les erreurs

Supposon qu'on envoie $s = [1\ 0\ 1\ 1\ 1\ 0]$ et qu'on recois $r = [1\ 0\ 1\ 0\ 1\ 0]$ alors

$$Hr^T = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

Le vecteur $s = Hr^T$ est le **syndrome** de r, il indique quel contraine de *parity-check* ne sont pas satisfaites par r.

Ici $s = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ et l'équation de parité associé est $s_4 = s_1 \oplus s_2$.

Un *block code* ne peut détecter des erreurs que si ces dernières ne transforment pas un *codeword* valide en un autre *codeword* valide. (voir Code de *Hamming*)

- Distance de Hamming : Nombre de positions où les bits diffèrent entre deux *codewords*.
  Exemple : [1 0 1 0 0 1 1 0] et [1 0 0 0 0 1 1 1] diffèrent aux positions 3 et 8
  ⇒ Distance de Hamming = 2.
- Distance minimale ($d_{\min}$) : La plus petite distance de Hamming mesurée entre n'importe quelle paire de *codewords* appartenant au code.

Un code avec une distance minimale $d_{\min}$ peut garantir la détection de t erreurs si et seulement si :

$$t < d_{\min}$$

Exemple :
Pour *Hamming* (7,4) vu après, on a $d_{\min} = 3$.
⇒ Il garantit la détection de 1 ou 2 erreurs ($t < 3$).
⇒ Si 3 bits (ou plus) s'inversent, le message peut correspondre à un autre *codeword* valide. (Exemple 1.8 [1]).

Pour corriger l'erreur, le décodeur cherche le *codeword* le plus probable.

Principe (*maximum-likelihood* (ML) Decoder) : Il choisit le *codeword* s valide qui a la plus petite distance de Hamming avec le message reçu r. (Si égalité alors le choix est aléatoire).

$$\hat{s} = \min_{c \in C} d_H(r, s)$$

Avec C l'ensemble des *codewords* valides.

Code de *Hamming*

But : Ajouter de la redondance à des bloques de données.
Block code : règle de conversion d'un sequence de bits $\boldsymbol{s}$ de longueur K dans une séquencce $\boldsymbol{t}$ de N bits. (Redondance ⇒ $N > K$).
Dans un code linéaire les $N - K$ bits réstant sont linéaire en fonction des K bits originaux, ce sont les *parity-check bits*.

- *Hamming* (7,4)

  L'encodage se visualise via 3 cercles sécants (Diagramme de Venn). Les 7 bits sont placés de sorte que la parité de chaque cercle soit paire (somme = 0).

  - Bits de Source (s_1, s_2, s_3, s_4) : Copiés directement dans le message transmis ($t_1..t_4$).
  - Bits de Parité (t_5, t_6, t_7) : Calculés pour valider les cercles.

$$t_5 = s_1 \oplus s_2 \oplus s_3 \text{ (Cercle 1)}$$
$$t_6 = s_2 \oplus s_3 \oplus s_4 \text{ (Cercle 2)}$$
$$t_7 = s_1 \oplus s_3 \oplus s_4 \text{ (Cercle 3)}$$

Le **Syndrome** $\boldsymbol{z}$: On vérifie la parité des cercles à l'arrivée.
- 1 cercle faux → Erreur sur le bit de parité.
- 2 ou 3 cercles faux → Erreur à l'intersection unique des cercles fautifs.

On peut le voir sous forme de matrice.
Message transmit $\boldsymbol{t}$ (*codeword*) :

$$t = G^T s$$

avec $\boldsymbol{G}$ la matrice génératrice du code.

Visualisation de la solution avec diagramme de Venn.
Trouver P tel que

$$G^T = \begin{bmatrix} I_n \\ P \end{bmatrix}$$

avec $z = Hr$ et H la mtrice *parity-check* $H = \begin{bmatrix} -P & I_{n-1} \end{bmatrix} = \begin{bmatrix} P & I_{n-1} \end{bmatrix}$
Et donc tous les *codewords* satisfont $t = G^T s$,

$$Ht = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Mais $r = G^T s + n$ on doit trouver $\boldsymbol{n}$ tel que $Hn = z$. C'est le probleme *maximum-likelihood decoder*.

Voir exemple [1] p9. | 3Blue1Brown Hamming codes

Low-density parity-check codes (LDPC) [2]

Block codes définis par une matrice *parity-check* $\boldsymbol{H}$ très **creuse** (beaucoup de 0).
⇒ Décodage en $\mathcal{O}(n)$

Défintions et Propriétés

Création : On construit d'abord la matrice creuse $\boldsymbol{H}$, puis on en déduit la matrice génératrice $\boldsymbol{G}$ (contrairement aux codes classiques).

Décodage : Décodage itératif basé sur une représentation graphique de $\boldsymbol{H}$ (Graphe de Tanner) (au lieu du décodage ML habituel).

Régularité : Un code LDPC est dit (w_c, w_r)-régulier si :
- Chaque bit de code appartient à w_c équations de parité.
- Chaque équation de parité contient w_r bits de code.

Irréguliers Le nombre de 1 varie selon les lignes et les colonnes de $\boldsymbol{H}$.
On définit la **distribution des degrés** (v, h) :
- $\boldsymbol{v_i}$: la fraction des colonnes (bits) ayant un poids de i.
- $\boldsymbol{h_i}$: la fraction des lignes (équations de parité) ayant un poids de i.

Propriété : nombre total de 1 dans $\boldsymbol{H}$:
Pour une matrice de m lignes et n colonnes :
- Code régulier : $m \cdot w_r = n \cdot w_c$

- Code irrégulier : $m \sum_i (h_i \cdot i) = n \sum_i (v_i \cdot i)$

LDPC constructions

Principe : On part d'une matrice remplie de zéros et on y place un petit nombre de 1 pour respecter la distribution de degrés.

Gallager

Il faut imaginer que la matrice $\boldsymbol{H}$ (de $\boldsymbol{m}$ lignes) est découpée horizontalement en $\boldsymbol{w_c}$ « bandes » de même taille (chacune a donc $\frac{\boldsymbol{m}}{\boldsymbol{w_c}}$ lignes).

- **Création de la 1ère bande** : On place $\boldsymbol{w_r}$ 1 consécutifs sur chaque ligne. À chaque fois qu'on descend d'une ligne, on décale ces 1 vers la droite.
- **Création des autres bandes** : On prend simplement la 1ère bande et on mélange aléatoirement l'ordre de ses colonnes.
- **Résultat** : Comme on a superposé $\boldsymbol{w_c}$ bandes, et que dans chaque bande il y a exactement un 1 par colonne, on est certain que chaque colonne de $\boldsymbol{H}$ aura exactement un poids de $\boldsymbol{w_c}$.
- Exemple :

  Code régulier avec $n = 12$ colonnes, $w_c = 3$ et $w_r = 4$.

  $\boldsymbol{H}$ aura $m = \frac{12 \cdot 3}{4} = 9$ lignes.

  On divise ces 9 lignes en $w_c = 3$ blocs horizontaux de 3 lignes.

  Bloc 1 (Escalier) :

  $$B_1 = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

  Bloc 2 (Permutation aléatoire des colonnes du Bloc 1) :

  $$B_2 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

  Bloc 3 (Permutation aléatoire des colonnes du Bloc 1) :

  $$B_3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

  Et donc

  $$H = \begin{bmatrix} B_1 \\ B_2 \\ B_3 \end{bmatrix}$$

  Finalement dans n'importe quel bloc en regardant une colonne il n'y à qu'un 1.

MacKay et Neal

- La matrice $\boldsymbol{H}$ est remplie une colonne à la fois, de gauche à droite.
- Les 1 sont placés aléatoirement dans les lignes qui ne sont pas encore pleines.

- Si à un moment, il y a plus de lignes incomplètes que de colonnes restantes à ajouter, la distribution des lignes ne sera pas exacte. On peut alors revenir de quelques colonnes en arrière ou recommencer le processus jusqu'à obtenir le bon résultat.
- Exemple :

  Code régulier $(3, 4)$ de longueur 12.
  Quand on ajoute la 11ème colonne, les lignes non remplies étaient les lignes $2, 4, 5, 6$ et 9.
  L'algorithme a choisi d'y placer un 1 sur les lignes $2, 4$ et 6.

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

Codes Repeat-Accumulate (RA)

- Les $\boldsymbol{m}$ dernières colonnes de la matrice $\boldsymbol{H}$ ont toutes un poids de 2 et forment un motif « en escalier ».
- Encodage rapide.
- Exemple :

  Code RA de longueur 12 avec un ratio de $\frac{1}{4}$.

$$H = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

  Les trois premières colonnes de $\boldsymbol{H}$ correspondent aux bits du message initial.
  Les bits de parité (à partir de la 4ème colonne) se calculent ensuite en cascade :
  $c_4 = c_1$
  $c_5 = c_4 \oplus c_1$
  $c_6 = c_5 \oplus c_2$
  ...
  Chaque bit de parité peut être calculé un par un en utilisant seulement les bits du message et le bit de parité juste avant lui.

Graphe de Tanner

Deux ensembles de sommets :

- $G = (X \sqcup Y, A)$ biparti.
- $X = \{\text{bits du codeword}\}$ et $|X| = n$
- $Y = \{\text{équations de parité}\}$ (*check-nodes*) et $|Y| = m$

- Soit $x \in X, y \in Y$, alors il existe $(x, y) \in A$ si le bit est inclu dans l'équation de parité corerspondante.
- $|A| =$ nombre de 1 dans la matrice de parité.

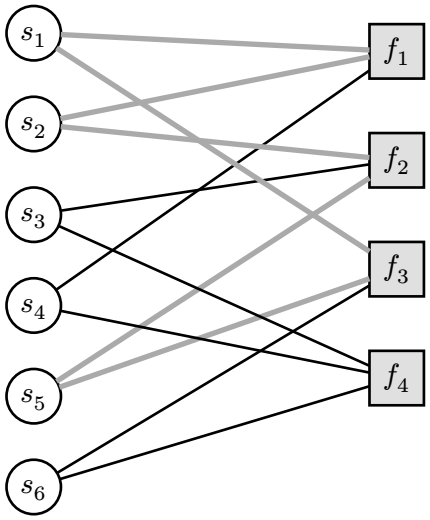
Exemple : Pour la matrice *parity-check* suivante (régulière $w_c = 2$, $w_r = 3$)

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Le graphe de Tanner correspondant

Noeuds de bits
(Variables s_i)

Noeuds de parité
(Contraintes f_j)



- ***Girth*** : La taille du plus petit cycle présent dans tout le graphe. (biparti $\Rightarrow$ cycle de taille pair donc plus petit cycle de taille 4).
- **Exemple :**

Dans notre H le plus petit cycle est de taille 6 en gris. $c = s_1 \to f_1 \to s_2 \to f_2 \to s_5 \to f_3 \to s_1$.

On veut éviter les cycles courts, il existe un algorithme de Mackay Neal qui permet de d'éviter les 4-cycles. algorithme [2] p14. D'autre methode vont être vu (algébriques) pour des court codes.

Encodage

Matrice *parity-check*

$$H = [A, I_{n-k}]$$

avec $A \in M_{(n-k)\times k}(\mathbb{Z}/2\mathbb{Z})$ on trouve alors G par réducation de Gausse-Jordan sur H.

$$G = [I_k, A^T]$$

1. Mettre H sous forme échelonée puis sous forme ligne-cheloné réduite puis sous forme standard (avec des permutation de colonnes) puis construire G voir [2] p15. Cette méthode ne rend pas la matrice creuse donc complexité nul.
2. Transformer H en matrice triangulaire inférieur aproximative. Voir [2] p17.

Décodage

Voir les exemple du papier [2] p21.

QC-LDPC

Code Rust

- Voir la strucutre du projet en amont
- Implémentation des codes ldpc (avec un peu toutes les méthodes possible pour faire des test etc)
  - Encodage Implémentation avec H aléatoire faire Gausse-Jordan pour trouver G et $s = uG$
  - Decodage par bit-flipping
  - —
  - Decodage sum product
  - Hashmap pour ne pas avoir à recalculer G pour toutes les longueurs de message. (à H fixé)
  - Implémenter le « repeat-accumulate » (RA) compliqué
- Et faire un export pour visualiser les matrice et le graphe dans un fichier typst par exemple ou autre
- Implementation en rust https://github.com/daniestevez/ldpc-toolbox (lire le code pour voir)
- Voir SIMD (`std::simd`) pour le calcul de LLR sur plusieur bits en même temps.
- Voir le multi-threadage possible (après l'implementation)

Bibliographie

[1] David J.C. MacKay, *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

[2] Sarah J. Johnson, « Introducing Low-Density Parity-Check Codes ».