Théorie de l'information et turbocodes

Karkar Skander, Picard Bertille, Zamolodtchikov Petr

28 février 2019

L'objectif de ce projet est d'étudier les codes correcteurs d'erreurs, et plus précisément les turbo-codes. Cela nous amènera à nous pencher sur la théorie du codage de canal, aussi appelée théorie des codes, qui s'inscrit dans le cadre plus général de la théorie de l'information. L'idée des codes correcteurs d'erreurs est de retrouver l'information transmise au cours d'une communication lorsque celle-ci comporte des erreurs une fois reçue, dues à des problèmes de transmission. Par exemple, une petite rayure sur un CD peut altérer l'information. Le code correcteur d'erreur va permettre de rétablir l'information initiale. Dans un premier temps, nous passerons brièvement en revue la littérature existante sur le sujet. Nous situerons notre projet dans le cadre de la théorie des codes, ce qui nous amènera à expliciter les notions principales dont on se resservira. Nous présenterons aussi à cette occasion les différents types de codes correcteurs existants. Dans un deuxième temps, nous nous pencherons en détails sur un type de codes, les codes linéaires, dont nous expliquerons les mécanismes assez simples recourant à des notions d'algèbre. Nous proposerons à cette occasion un programme permettant de corriger un message codé comportant des erreurs à la réception. Dans un troisième temps, nous nous intéresserons au codes dits « *Low-Density Parity-Check* ». A cette occasion, nous présenterons l'algorithme à propagation de croyance dont on étudiera le comportement asymptotique. Nous proposerons également un programme pour illustrer le fonctionnement des codes LDPC.

1 De la limite de Shannon aux codes LDPC

La théorie des codes est fondée avec l'article de Shannon, « Mathematical Theory of Communication »(1948) [9]. Non seulement Shannon a mis en place un certain nombre de notions que nous retrouverons dans les travaux de ses successeurs comme celle d'information ou d'entropie, mais il y énonce également deux théorèmes majeurs. Aujourd'hui encore, les codes développés visent la limite de Shannon. Pour comprendre cet objectif au centre des travaux, nous reviendrons sur les bases établies par Shannon dans son article. Nous examinerons ensuite les codes proposés pour approcher la limite de Shannon.

1.1 Shannon et la communication comme problème probabiliste

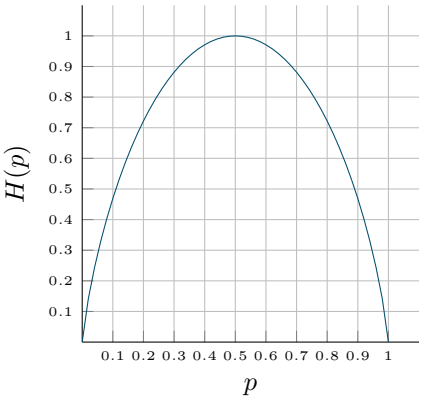
Notions fondamentales : information, entropie et capacité du canal

Comme le souligne Weaver (1998) [10] dans un article présentant les travaux de Shannon, cette théorie se développe avant tout autour des aspects techniques de la communication, bien qu'elle concerne aussi d'autres niveaux de la communication comme la compréhension d'un message ou son effet. Le problème « avec quelle précision les symboles de la communication peuvent être transmis ? » [1] est le problème central et est d'ordre technique. Cela

1. « How accurately can the symbols of communication be transmitted ? », [10]

explique pourquoi la théorie de l'information délimite de manière si spécifique son objet de recherche : **la quantité d'information** est définie par le nombre de messages dans l'ensemble des messages possibles, ou par n'importe quelle fonction monotone de ce nombre. L'utilisation fréquente du logarithme est assez intuitive. En effet, si je transmets trois chiffres prenant chacun les valeurs 1 ou 0, j'ai 2^3 messages possibles : 000, 001, 011, 010, 100, 101, 110, 111. Plutôt que de dire que la quantité d'information vaut 8 (nombre de messages possibles), nous dirons que nous avons trois unités d'information, précisément trois bits, dans ce cas. De manière générale, lorsque l'on peut transmettre une séquence de n symboles, on dit que la quantité d'information vaut n, et ce qu'importe si ces symboles prennent chacun pour valeurs 0 ou 1, un chiffre entre 0 et 9, ou des lettres de l'alphabet.

L'entropie intervient dans notre problème [2] car la source d'information peut émettre plusieurs symboles avec des probabilités données. L'**entropie** est alors une mesure de l'incertitude portant sur l'information émise par la source. Ainsi, si une source n'émet systématiquement qu'un seul symbole, disons 0, alors l'entropie est nulle (et minimale). Cependant, si la source émet avec une probabilité α un 0, et avec une probabilité $1-\alpha$ un 1, il y a une incertitude, ou entropie. Cette entropie a une définition mathématique. C'est la fonction H des probabilités d'occurrence des évènements (et de la base b) définie [3] par : $H(p_1, ...p_k) = -\sum_{i=1}^{k} p_i \log_b p_i$



Prenons un exemple simple pour illustrer cette notion d'entropie. Considérons une source envoyant 0 avec la probabilité p, et 1 avec probabilité $1-p$. Nous avons alors :

$$H(p) = -(p \log_2(p) + (1-p) \log_2(1-p))$$

Nous avons représenté la fonction ci-contre. On retrouve les deux cas où l'entropie est nulle : lorsque 0 est envoyé avec certitude, et lorsque 1 est envoyé avec certitude. L'entropie est maximale lorsque les probabilités associées aux deux symboles sont égales, c'est-à-dire, assez intuitivement, lorsqu'on est le moins sûr du symbole qui va être envoyé.

Enfin, une dernière notion qu'il nous faudra introduire pour comprendre le théorème suivant est celle de capacité du canal. La capacité C d'un canal bruité correspond au taux de transmission possible maximum.

Théorème fondamental du codage

Dans le cas d'une source d'information discrète [4], Shannon (1948) [9] énonce le théorème suivant. Soit un canal discret dont on note la capacité C. Soit une source d'information discrète dont on note l'entropie par seconde H. Si $H \leq C$, alors il existe des codes correcteurs tels que le message envoyé peut être transmis avec une probabilité d'erreur aussi petite qu'on le souhaite [5]. Le problème est que ce théorème nous donne l'existence des codes correcteurs d'erreurs approchant une limite, mais ne nous dit pas comment les construire. Nous allons

2. L'entropie intervient y compris avant même d'introduire du bruit dans la transmission du message. Ainsi on pourra parler d'entropie à la source au moment de l'envoi du message, et d'entropie à la sortie, c'est-à-dire à la réception du message.

3. Notons que l'on pourrait prendre également $H'(p_1, ..., p_k) = CH(p_1, ..., p_k)$ où C est une constante positive. La fonction H' vérifierait toujours les propriétés qui nous intéressent.

4. Nous renvoyons le lecteur à la suite de l'article pour le cas continu.

5. « *Let a discrete channel have the capacity C and a discrete source the entropy per second H. If $H \leq C$, there exists a coding system such that the output of the source can be transmitted over the channel with an arbitrarily small frequency of erros (or an arbitrarily small equivocation). [...]* », Shannon (1948) [9]

donc passer en revue les différents types de codes correcteurs proposés à la suite de la publication de l'article de Shannon.

1.2 Approcher la limite de Shannon : diversité des stratégies

La littérature qui s'est développée après 1948 vise avant tout à proposer des solutions pratiques pour approcher la limite de Shannon. Les codes linéaires constituent une première catégorie de codes correcteurs dont on détaillera les mécanismes dans la section suivante et qui reposent sur une redondance de l'information grâce à des combinaisons linéaires des symboles du message initial envoyé. On peut citer parmi ceux-ci le code de Hamming (1950) [5] permettant de corriger une erreur sur une lettre d'un message par exemple. Reed et Solomon (1960) [7] construisent plus tard des codes reposant sur des polynômes et permettant de corriger davantage d'erreurs. Ces codes s'approchent tous assez peu finalement de la limite de Shannon et les turbocodes marquent une avancée sur ce point. Nous nous pencherons pour ce projet sur les codes LDPC, les turbocodes pouvant être considérés comme des codes LDPC (MacKay, 1998 [2]). Ces codes reposent sur des calculs de logarithmes des rapports de vraisemblance et feront donc appel à des notions de statistiques étudiées en cours.

2 Les codes linéaires

La partie qui suit est davantage formalisée. Nous ne reprenons que les étapes théoriques essentielles dans la mise en place des codes linéaires, et reportons le lecteur aux manuels pour davantage de détails (voir par exemple Papini (1998) [6]). Nous proposons par la suite un programme pour mettre en pratique les codes linéaires.

2.1 Codes correcteurs d'erreurs : définition générale du cadre d'étude

Un message est constitué d'une suite de caractères étant susceptibes d'être altérés par des phénomènes physiques ; le décodage d'un message a pour ambition de restituer l'original. Nous présenterons ici le cadre général du problème du décodage d'un message.

Soit $n \in \mathbb{N}$ et $\mathcal{A}$ un alphabet, c'est à dire un ensemble de caractères quelconque.

Définition 0.1. Distance de Hamming :
Soit $x = (x_1, ..., x_n), y = (y_1, ...y_n) \in \mathcal{A}^n$ on définit la distance de Hamming comme suit :

$$d(x, y) = card(\{i \in \{1, ..., n\} | x_i \neq y_i\})$$

On vérifie immédiatement qu'il s'agit bien d'une distance sur $\mathcal{A}^n$. $(\mathcal{A}^n, d)$ est alors un espace métrique et avec cette métrique nous allons pouvoir définir des conditions nécessaires de décodage. Un message est donc transmis, ce message est au départ $x = (x_1, ..., x_n)$ et le message reçu est $x' = (x'_1, ..., x'_n)$, le nombre d'erreurs dans x' est $d(x, x')$, ainsi en admettant un nombre raisonnable maximal d'erreurs e nous pouvons isoler l'ensemble des messages qui auraient pû produire x', il s'agit de la boule centrée en x' et de rayon e : $\mathcal{B}(x', e)$. L'idée est d'isoler un sous-ensemble $\mathcal{C}$ de $\mathcal{A}^n$ qui permette de réduire $\mathcal{B}(x', e) \cap \mathcal{C}$ à un singleton. De cette manière, tous les messages avec au plus e erreurs auraient un et un seul mot de $\mathcal{C}$ dont ils pourraient provenir.

Définition 0.2. Un code de longueur n sur $\mathcal{A}$ est un sous-ensemble $\mathcal{C}$ de $\mathcal{A}^n$

Supposons dans la suite que le nombre d'erreurs est majoré par $e \in \mathbb{N}^*$.

Définition 0.3. Le code $\mathcal{C}$ de longueur n vérifie la condition de décodage d'ordre e si

$$\forall\ x' \in \mathcal{A}^n, \exists \text{ au plus un } x \in \mathcal{C} \text{ tel que } x' \in \mathcal{B}(x,e)$$
$$\Leftrightarrow$$
$$\forall x, y \in \mathcal{C}, x \neq y \Rightarrow \mathcal{B}(x,e) \cap \mathcal{B}(y,e) = \emptyset$$

Si la condition de décodage d'ordre e est vérifiée, tous les messages ayant subi au plus e erreurs seront décodés de façon exacte.

Si on définit d_{min} la distance minimale du code par $d_{min} = \min\{d(x,y) \mid x \neq y \in \mathcal{C}\}$ alors il est évident que le code vérifie la condition de décodage d'ordre e dès que $d_{min} \geq 2e+1$. On appelle donc la **capacité de correction** du code la partie entière de $\frac{d_{min}-1}{2}$.

2.2 Codes linéaires

Ici l'alphabet $\mathcal{A}$ est un corps fini $\mathbb{K}$. Par exemple $\mathbb{F}_2$ (i.e $\mathbb{Z}/2\mathbb{Z}$). $(\mathcal{A}^n, +, \times)$ est donc un $\mathbb{K}$-espace vectoriel

Définition 1.1. Pour tout $x \in \mathcal{A}^n$ on appellera poids de x l'application définie par :

$$w(x) = card(\{i \in \{1,...,n\} | x_i \neq 0\})$$

On a alors pour tout $x, y \in \mathcal{A}^n$:

1. $d(x,y) = w(x-y)$
2. $w(x) = d(x,0)$
3. $w(x) = 0 \Leftrightarrow x = 0$
4. $\forall \lambda \in \mathbb{K}^*, w(\lambda x) = w(x)$
5. $w(x+y) \leq w(x) + w(y)$

Définition 1.2. On appellera code linéaire de longueur n sur $\mathbb{K}$ et de dimension k, tout sous-espace vectoriel $\mathcal{C}$ de $\mathbb{K}^n$ de dimension k. On a alors $card(\mathcal{C}) = card(\mathbb{K})^k$

Définition 1.3. On appellera matrice génératrice d'un code linéaire $\mathcal{C}$ toute matrice de $\mathcal{M}_{k,n}(\mathbb{K})$ dont les lignes sont une base de $\mathcal{C}$. Autrement dit, toute matrice associée à une application linéaire surjective de $\mathbb{K}^k \to \mathcal{C}$.

Les codes et les matrices génératrices associées sont définies à une permutation près, on parle de codes équivalents quand les coordonnées des vecteurs de $\mathcal{C}$ sont simplement permutés. Ceci nous permet de parler de matrice normalisée : une matrice génératrice G normalisée est alors de la forme $[I_k][R]$ où R est appelée matrice de redondance. Un code possédant une telle matrice génératrice sera appelé un code systématique.

Nous allons à présent voir comment nous pouvons coder puis décoder des messages à l'aide d'un code linéaire. Nous supposerons construit $\mathcal{C}$ un code linéaire e-correcteur.

Nous avons un message $(u_1, u_2, ..., u_k)$, on obtient le message codé en 'plongeant' ce vecteur dans $\mathbb{K}^n$: le message codé qu'on envoye est $m_u = (u_1, ...u_k)G$ où G est génératrice de $\mathcal{C}$. Si G est normalisée, le message obtenu est de la forme $m_u = (u_1, ..., u_k, x_{k+1}, ...x_n)$ où $(x_{k+1}, ..., x_n)$ est la composante de redondance du message, elle permettra de distinguer deux messages même erronés.

Définition 1.4. On appelle matrice de contrôle de $\mathcal{C}$ toute matrice génératrice de l'orthogonal de $\mathcal{C}$

Proposition 1.1. Soit $\mathcal{C}$ un code linéaire systématique et G une matrice normalisée de $\mathcal{C}$, $G = [I_k][R]$ alors $H = [R^t][-I_{n-k}]$ est une matrice de contrôle de $\mathcal{C}$.

La matrice de contrôle sert à décoder un message. Si on note h l'application linéaire associée à la matrice de contrôle H, par définition de H, $x \in \mathcal{C} \Leftrightarrow h(x) = 0$ soit $\mathcal{C} = ker(H)$ ainsi si x est le mot envoyé et $y = x + \epsilon$ est le mot reçu, où $\epsilon \in \mathbb{K}^n$ est l'erreur ($w(\epsilon)$ est donc le nombre d'erreurs), on aura, par linéarité, $h(y) = h(\epsilon)$. Le message reçu appartient donc à la même classe d'équivalence que l'erreur dans la relation d'équivalence latérale par rapport à h (les classes d'équivalence sont définies comme suit : $x \sim y \Leftrightarrow h(x - y) = 0$).

Définition 1.5. On appellera syndrome de $x \in \mathbb{K}^n$ par rapport à H le vecteur $h(x) \in \mathbb{K}^{n-k}$.

Ainsi pour connaître le message envoyé grâce à un code de capacité de correction e (sous l'hypothèse qu'il n'y a effectivement pas eu plus de e erreurs), il suffit de regarder le syndrome du message reçu, i.e $h(y)$ puis de chercher dans la classe d'équivalence correspondante le seul mot de poids au plus e, soit ϵ ce mot (il est unique car le code est e-correcteur, et $d(x, y) = w(y - x) = w(\epsilon)$). Finalement on retrouve $x = y - \epsilon$. Le message initial décodé est donc les k premiers caractères de x.

Nous proposons en annexe une implémentation d'un code correcteur et son application à un message erronné. Le programme implémente un code linéaire à partir de sa matrice génératrice normalisée. Le programme calcule la matrice de contrôle du code puis sa capacité de correction e (à partir de sa distance minimale qui dans le cas d'un code linéaire est simplement le poids minimal des mots non nuls du code). Il génère ensuite la liste des erreurs corrigeables (de poids inférieur ou égal à la capacité de correction) et leurs syndromes ce qui permet de décoder. Un mot (en binaire) est codé au moyen de la matrice génératrice. Une erreur aléatoire est ensuite commise et le programme trouve l'erreur et donc retrouve le mot initial car le syndrome du message reçu est égal au syndrome de l'erreur commise.

Nous allons maintenant introduire les codes Low-Density Parity-Check (LDPC) en nous basant sur le chapitre 9 de [1]. Ces codes font appel cette fois à des outils statistiques comme le maximum de vraisemblance. Ils doivent leur succès à leurs performances.

3 Les codes Low-Density Parity-Check (LDPC)

3.1 Préliminaires

On considère un problème de transmission de $(c_1, ...c_n)$ où $\forall i$, $c_i \in \{0, 1\}$. On suppose que la variable aléatoire c_i suit une loi de Bernoulli avec paramètre $\frac{1}{2}$. Notons que cette hypothèse est raisonnable dans le cas où l'on s'intéresse à la transmission d'un texte avec beaucoup de caractères. Celui-ci comportera en effet des nombres très grands et équivalents de 0 et de 1. On va définir une quantité centrale dans notre problème qui est le log-likelihood ratio. C'est un rapport défini ainsi :

Définition 2.1. le LLR de c (variable aléatoire dans $\{0, 1\}$) est $L(c) = \log \frac{P(c=0)}{P(c=1)}$

Remarquons que cette quantité s'étend de $-\infty$ à $+\infty$. On a de plus $P(c = 1) = e^{-L(c)}P(c = 0) = e^{-L(c)}(1 - P(c = 1))$, donc :

$$P(c = 1) = \frac{1}{1 + e^{L(c)}} \text{ et } P(c = 0) = \frac{e^{L(c)}}{1 + e^{L(c)}} \quad (1)$$

Définition 2.2. Une équation de parité est une équation reliant n données binaires par l'opérateur $\oplus$ (ou exclusif) : $c_1 \oplus c_2 \oplus ... \oplus c_n = 0$

Exemple. On dispose de variables binaires c_1, c_2 et c_3 liées par la contrainte de parité $c_1 \oplus c_2 \oplus c_3 = 0$. On connaît $L(c_1)$ et $L(c_2)$. On a que $c_3 = 0$ si et seulement si $(c_1, c_2, c_3) \in \{(0,0,0),(1,1,0)\}$. De même $c_3 = 1$ si et seulement si $(c_1, c_2, c_3) \in \{(1,0,1),(0,1,1)\}$. Donc :

$$P(c_3 = 1) = P(c_1 = 1)P(c_2 = 0) + P(c_1 = 0)P(c_2 = 1)$$
$$P(c_3 = 0) = P(c_1 = 0)P(c_2 = 0) + P(c_1 = 1)P(c_2 = 1)$$

D'après les équations (1) :

$$P(c_3 = 1) = \frac{1}{1+e^{L(c_1)}}\frac{e^{L(c_2)}}{1+e^{L(c_2)}} + \frac{e^{L(c_1)}}{1+e^{L(c_1)}}\frac{1}{1+e^{L(c_2)}} = \frac{e^{L(c_1)}+e^{L(c_2)}}{(1+e^{L(c_1)})(1+e^{L(c_2)})}$$
$$P(c_3 = 0) = \frac{e^{L(c_1)}}{1+e^{L(c_1)}}\frac{e^{L(c_2)}}{1+e^{L(c_2)}} + \frac{1}{1+e^{L(c_1)}}\frac{1}{1+e^{L(c_2)}} = \frac{e^{L(c_1)+L(c_2)}+1}{(1+e^{L(c_1)})(1+e^{L(c_2)})}$$

Donc

$$\frac{P(c_3 = 0)}{P(c_3 = 1)} = \frac{e^{L(c_1)+L(c_2)}+1}{e^{L(c_1)}+e^{L(c_2)}}$$

$$L(c_3) = \log\frac{e^{L(c_1)+L(c_2)}+1}{e^{L(c_1)}+e^{L(c_2)}} =: L(c_1) \oplus L(c_2)$$

Si on applique la fonction $\tanh(\frac{x}{2}) = \frac{e^x-1}{e^x+1}$ à la dernière équation on obtient :

$$\tanh(\frac{L(c_3)}{2}) = \frac{e^{L(c_1)+L(c_2)}+1-e^{L(c_1)}-e^{L(c_2)}}{e^{L(c_1)}+e^{L(c_2)}}\frac{e^{L(c_1)}+e^{L(c_2)}}{e^{L(c_1)+L(c_2)}+1+e^{L(c_1)}+e^{L(c_2)}}$$
$$\tanh(\frac{L(c_3)}{2}) = \frac{e^{L(c_1)}-1}{e^{L(c_1)}+1}\frac{e^{L(c_2)}-1}{e^{L(c_2)}+1} = \prod_{j=1}^{2}\tanh(\frac{L(c_j)}{2})$$

Dans le cas d'une équation de parité à n termes, le résultat précédent se généralise ainsi :

Proposition 2.1. On dispose de variables binaires $c_1, ..., c_n$ telles que $c_1 \oplus c_2 \oplus ... \oplus c_n = 0$ et on connaît les LLR des variables c_j $\forall j \neq i$ alors

$$L(c_i) = \bigoplus_{j\neq i} L(c_j)$$
$$\tanh(\frac{L(c_i)}{2}) = \prod_{j\neq i}\tanh(\frac{L(c_j)}{2})$$

Remarque. Souvent pour des raisons pratiques ou pour simplifier les calculs on considère des variables binaires à valeurs dans $\{\pm 1\}$ en appliquant la transformation $c \mapsto (-1)^c$. L'opération du groupe à deux éléments devient la multiplication avec 1 pour élément neutre.

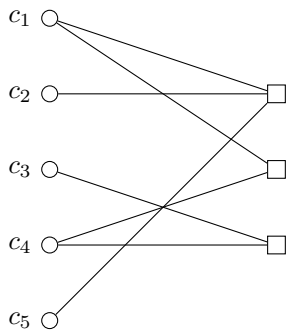
3.2 Cadre du problème

Le mot du code $(c_1, ..., c_n)$ à transmettre est obtenu comme on l'a vu par l'application d'une matrice génératrice G à un message initial de longueur $k < n$. On a vu aussi que le code peut être représenté par une matrice de contrôle H de dimension $m \times n$ où $m = n-k$. Le mot du code $(c_1, ..., c_n)$ vérifie alors $Hc^t = 0$. Ainsi la matrice H correspond à m équations de parité (une par ligne) que doivent vérifier $c_1, ..., c_n$. Pour illustrer ceci, on peut utiliser les

graphes de Tanner : les points représentent les nœuds de variables, les carrés représentent les noeuds de parité. En représentant chaque équation de parité (donc chaque ligne de H) par un nœuds de parité reliant les termes de l'équation, on représente le système d'équations détermminé par H. Un code LDPC consiste en une matrice de contrôle H creuse : on a beaucoup de 0 et seulement quelques bits sont reliés entre eux par des équations de parité. Ainsi, pour la matrice H :

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

On a le graphe de Tanner suivant :



Nœuds de variables Nœuds de parité

Le mot du code $(c_1, ..., c_n)$ est envoyé à travers un canal et est perturbé. Ce qui est reçu de l'autre côté est $x_i = c_i + b_i$ où b_i est un bruit (variable aléatoire continue). x_i n'est pas arrondi à 1 ou 0 et la valeur de b_i va donc servir au décodage (décodage à entrée souple).

On cherche des informations sur c_i sachant x_i qui est reçu. Ces informations sont résumées dans la quantité $L(c|x) = \log \frac{P(c=0|x)}{P(c=1|x)}$. Cette valeur nous informe sur la valeur de c ainsi : si $L(c|x) > 0$ alors $P(c = 0|x) > P(c = 1|x)$ etplus $L(c|x)$ tend vers ∞, plus $c = 0$ est probable. Inversement, plus $L(c|x)$ tend vers $-\infty$, plus $c = 1$ est probable. On peut écrire $L(c|x)$ ainsi grâce à la règle de Bayes :

$$\begin{aligned} L(c|x) = \log \frac{P(c=0|x)}{P(c=1|x)} = \log \frac{p(x|c=0)P(c=0)}{p(x|c=1)P(c=1)} &= \log \frac{p(x|c=0)}{p(x|c=1)} + \log \frac{P(c=0)}{P(c=1)} \\ &= L(x|c) + L(c) \end{aligned}$$

où p est la densité de x. Sous l'hypothèse qu'on a faite au début que c suit une Bernoulli de paramètre $\frac{1}{2}$, $L(c) = \log \frac{P(c=0)}{P(c=1)} = \log(1) = 0$ et la quantité d'intérêt devient $L(c|x) = L(x|c) = \log \frac{p(x|c=0)}{p(x|c=1)}$. De plus, si le bruit b suit une loi $\mathcal{N}(0, \sigma^2)$ alors :

$$\begin{aligned} c = 1 \Rightarrow x = 1 + b \sim \mathcal{N}(1, \sigma^2) \Rightarrow p(x|c=1) &= \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-1)^2}{2\sigma^2}} \\ c = 0 \Rightarrow x = b \sim \mathcal{N}(0, \sigma^2) \Rightarrow p(x|c=0) &= \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}} \end{aligned}$$

Donc on peut calculer

$$\frac{p(x|c=0)}{p(x|c=1)} = \exp(-\frac{x^2}{2\sigma^2} + \frac{(x-1)^2}{2\sigma^2}) = \exp(\frac{-2x+1}{2\sigma^2})$$
$$L(c|x) = L(x|c) = \frac{-2x+1}{2\sigma^2}$$

Définition 2.2. Si le bruit b suit une loi $\mathcal{N}(0,\sigma^2)$ et que la sortie du canal à partir d'une entrée binaire c est $x = c+b$ alors le canal est dit BIAWGNC (*Binary Input Additive White Gaussian Noise Channel*) de paramètre σ.

3.3 Algorithme à propagation de croyance

On présente l'algorithme à propagation de croyance (algorithme BP pour *belief propagation*) pour décoder un mot du code envoyé à travers un canal BIAWGNC. D'après ce qui précède, la valeur $I_i := \frac{-2x_i+1}{2\sigma^2}$ est utilisée pour initialiser notre estimation de la quantité d'intérêt $L(c_i|x_i)$. On a bien que plus x_i est grand (positif), plus $L(c_i|x_i)$ est petit (négatif) et plus $P(c_i = 1)$ est grand, ce qui est logique car $x_i = c_i + b_i$. Cette valeur initiale ne prend pas en compte les équations de parité que les c_i doivent vérifier.

Pour tout nœud de variable indexé par $i \in \{1,...,n\}$ on note $P(i) = \{$indices des nœuds de parité liés au nœud de variable $c_i\}$ = $\{$indices des équations de parité où c_i apparaît$\}$. Et pour tout nœud de parité indexé par $p \in \{1,...,m\}$, $C(p) = \{$indices des nœuds de variables liés au nœud de parité numéro $p\}$ = $\{i$ tels que c_i apparaît dans la p-ième équation de parité$\}$

Le principe de l'algorithme est d'avoir un échange d'informations entre les nœuds de variables et les nœuds de parité où les nœuds de variables apportent leurs informations initiales et les nœuds de parité s'assurent que les équations de parité sont vérifiées. Chaque nœud de variable i envoie une valeur actuelle LV_{ip} au nœud de parité p. Chaque nœud de parité p applique la formule de la proposition 2.1 à chaque nœud de variable i qui lui est liée et lui renvoie une nouvelle valeur LP_{pi} (i.e. lui renvoie la prédiction faite par l'équation de parité sur la valeur de $L(c_i)$ en se basant sur les dernières estimations des autres $L(c_j)$ qu'il a reçues). Une grande valeur de LP_{pi} indique une forte probabilité que $c_i = 0$. Chaque nœud de variable i met alors à jour sa valeur LV_{ip} qu'elle renvoie au nœud de parité p en faisant la somme des valeurs qu'elle vient de recevoir à partir des autres nœuds de parité et ainsi de suite. L'algorithme est :

0. initialisation. $\forall\ i \in \{1,...,n\}$, $\forall\ p \in \{1,...,m\}$, $LV_{ip} = I_i$.
1. étape horizontale - traitement des parités. $\forall\ p \in \{1,...,m\}$, $\forall\ i \in C(p)$:

$$LP_{pi} = 2\tanh^{-1}(\prod_{j\in C(p)\setminus\{i\}} \tanh(\frac{LV_{jp}}{2})\ )$$

2. étape verticale - traitement des variables. $\forall\ i \in \{1,...,n\}$, $\forall\ p \in P(i)$:

$$LV_{ip} = I_i + \sum_{k\in P(i)\setminus\{p\}} LP_{ki}$$

3. estimation du bit. répéter 1 et 2 jusqu'à convergence et retourner $\forall\ i \in \{1,...,n\}$:

$$L_i = I_i + \sum_{k\in P(i)} LP_{ki}$$

Enfin, notre décodage est le suivant : $c_i = \mathbb{1}_{L_i<0}$

3.4 Evolution de densité

On veut étudier le comportement asymptotique de l'algorithme BP en se basant sur l'article [8]. En particulier on s'intéresse à l'évolution de la proportion de mauvaises décisions de décodage quand le nombre d'itérations de l'algorithme tend vers l'infini.

On considère des codes LDPC réguliers : un code LDPC est (d_v, d_p)-régulier si chaque noeud de variable est lié à d_v noeuds de parité et chaque noeud de parité est lié à d_p noeuds de variable. Si d_v, d_p et n (longueur du code) sont fixés, alors le nombre d'équations de parité est déterminé $m = \frac{nd_v}{d_p}$. On note alors $\mathcal{C}^n(d_v, d_p)$ l'ensemble des codes LDPC (d_v, d_p)-réguliers de longueur n.

On peut tirer un code de $\mathcal{C}^n(d_v, d_p)$ uniformément au hasard ainsi : un graphe d'un code de $\mathcal{C}^n(d_v, d_p)$ contient nd_v arrêtes. On numérote de 1 à nd_v les points de départs de ces arrêtes du côté des noeuds de variable et de même du côté des noeuds de parité et on tire uniformément une permutation ϕ de l'ensemble $\mathcal{S}_{nd_v}$ des permutations de $\{1, .., nd_v\}$. Les arrêtes du graphe sont alors $\{(i, \phi(i)),\ i = 1, .., nd_v\}$. On peut alors avoir des arrêtes parallèles liant les mêmes noeuds, mais on définit $\mathcal{C}^n(d_v, d_p)$ comme incluant de tels codes dégénérés. $\mathcal{C}^n(d_v, d_p)$ est en effet alors l'ensemble des graphes bipartites$(n, \frac{nd_v}{d_p})$ biréguliers(d_v, d_p) à arrêtes labellisées, qui est en bijection avec $\mathcal{S}_{nd_v}$. Ceci ne gênera pas notre analyse car on va en premier lieu exclure les graphes contenant des cycles courts (donc des arrêtes parallèles) et en deuxième lieu voir qu'une proportion de plus en plus grande de codes de $\mathcal{C}^n(d_v, d_p)$ (y compris ces codes dégénérés) se comportent comme dans le cas acyclique quand n explose.

On considère dans cette partie $c_i \in \{\pm 1\}$ avec la transformation évoquée plus haut qui envoie 0 vers 1 et 1 vers -1. L'initialisation I_i de l'algorithme BP devient $I_i = \frac{2x_i}{\sigma^2}$ mais tous les autres résultats sont identiques.

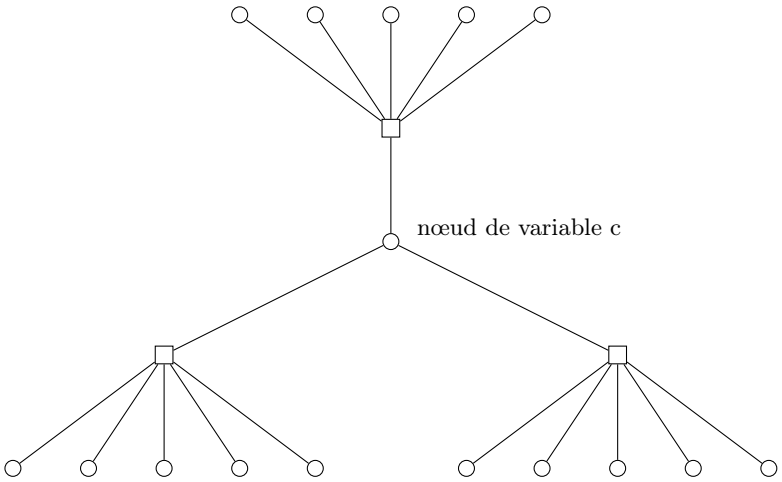
On s'intéresse à $P^{(n)}_{erreur}(l)$ l'espérance de la proportion de messages envoyés par les noeuds de variables (notés LV_{ij} plus haut) à l'itération l de l'algorithme qui sont erronés. Un message m est erroné si la décision $\hat{c_i} = 2\mathbb{1}_{m>0} - 1$ qu'il implique est fausse (différente de c_i). L'espérance est prise sur tous les codes de $\mathcal{C}^n(d_v, d_p)$ avec tirage uniforme, tous les mots du code et le bruit du canal. On a alors les résultats suivants :

1. Concentration : $\forall\ \delta > 0$, la probabilité que la proportion de messages erronés envoyés à l'itération l pour une instance quelconque (du code dans $\mathcal{C}^n(d_v, d_p)$, du mot du code et du bruit) soit en dehors de $]P^{(n)}_{erreur}(l) - \delta, P^{(n)}_{erreur}(l) + \delta[$ tend vers 0 exponentiellement en n.
2. Convergence vers code acyclique : $\lim\limits_{n\to\infty} P^{(n)}_{erreur}(l) = P^{\infty}_{erreur}(l)$, où $P^{\infty}_{erreur}(l)$ est l'espérance de la proportion de messages erronés envoyés à l'itération l de l'algorithme sur les codes dont le graphe ne contient pas de cycles de longueurs inférieures ou égales à $4l$.
3. Evolution de densité : $P^{\infty}_{erreur}(l)$ est calculable par un algorithme déterministe.
4. Seuil : Dans le cas d'un BIAWGNC, il existe une valeur du paramètre appelée seuil (*threshold*) σ^* telle que si le canal utilisé a pour paramètre $\sigma < \sigma^*$ alors $\lim\limits_{l\to\infty} P^{\infty}_{erreur}(l) = 0$ et si le canal utilisé a pour paramètre $\sigma > \sigma^*$ alors $\exists\ \gamma(\sigma) > 0$ telle que $\forall\ l \geq 1, P^{\infty}_{erreur}(l) \geq \gamma(\sigma)$.

Le premier résultat indique que presque tous les codes se comportent de la même manière asymptotiquement. Déterminer le comportement moyen $P^{(n)}_{erreur}(l)$ est donc intéressant. Le deuxième résultat indique que pour de longs codes, ce comportement moyen tend vers celui de codes sans cycles plus faciles à étudier. Le troisième résultat affirme qu'on peut en effet étudier ces codes. On va expliciter et prouver ce résultat.

On suppose donc désormais qu'on a un code LDPC (d_v, d_p)-régulier sans cycles de longueur inférieure ou égale à $4l$, avec l fixé. L'analyse qu'on va faire s'applique à d'autres canaux que BIAWGNC et à une famille d'algorithmes de décodage contenant l'algorithme BP, d'où l'introduction de nouvelles notations.

On considère les informations initiales I_i comme des variables aléatoires ($I_i = \frac{2x_i}{\sigma^2}$ et x_i est une variable aléatoire). Alors les I_i sont iid (car les x_i iid par hypothèse, dans le cas BIAWGNC : $x_i = c_i + b_i$ où les c_i sont iid et les b_i sont iid normaux centrés et $c_i \perp\!\!\!\perp b_j \ \forall\ i, j$). Tous les messages échangés sont des fonctions des initialisations I_i et sont donc aussi des variables aléatoires. L'absence de cycle de longueur inférieure ou égale à $4l$ signifie que pour tout noeud du graphe du code, son voisinage de profondeur $2l$ (ensemble des noeuds à distance $\leq 2l$ arrêtes de lui, appelé aussi voisinage de décodage à l'étape l) est un arbre.



Voisinage de profondeur 2 de c dans un code $(3, 6)$-régulier sans cycle de longueur ≤ 4

Soit v_i un noeud de variable. On indexe par j ses noeuds de parité voisins p_j. Le noeud v_i forme ses messages LV_{ij} à une itération $h \leq l$ à partir de messages LP_{ji} lui provenant des p_j. La structure d'arbre du voisinage de profondeur $2l$ de v_i assure que chaque message entrant LP_{ji} est une fonction (la même à chaque fois) d'un vecteur aléatoire $\mathbf{I}_j$ (de dimension déterminée par d_v et d_p) formé de variables initiales distinctes I_k. Aucun I_k n'apparaît dans deux vecteurs $\mathbf{I}_j$ (i.e. chacque $\mathbf{I}_j$ est formé d'informations initiales propres à lui). Les messages LP_{ji} entrant vers v_i sont donc iid (On a utilisé aussi le fait que dans l'algorithme BP, un noeud n'utilise pas dans son message à un noeud voisin l'information que ce noeud voisin vient de lui envoyer). Exactement le même raisonnement montre que les messages $LV_{.k}$ entrant vers un noeud de parité p_k sont iid. Aussi, tous les messages sortant à une itération $h \leq l$ dans la même direction (i.e. tous les messages LV et tous les messages LP) ont la même loi car ils sont formés par l'application de la même fonction à des vecteurs aléatoires de même loi (vecteurs aléatoires de même dimension de compostantes I_k). Par contre ils ne sont pas forcément indépendants car les vecteurs aléatoires ont des composantes en commun. Même les messages sortant d'un même noeud ne sont pas indépendants.

Pour résumer, à toute itération $h \leq l$ tous les messages allant vers le même noeud sont iid et tous les messages sortant dans la même direction ont la même loi. Comme les messages

sortant des noeuds de variables à l'étape l ont la même loi, l'espérance de la proportion de messages erronés est simplement la probabilité qu'un de ces messages soit erroné. L'algorithme *density evolution* permet de déterminer la loi des messages sortant des noeuds de variable à chaque itération jusqu'à l'itération l et donc de déterminer cette probabilité.

Pour formaliser ceci, on note que l'algorithme BP est un cas particulier d'une classe d'algorithmes de décodage de codes LDPC réguliers appelés *message passing algorithms*. Un algorithme *message passing* a exactement la même structure que l'algorithme BP, et est déterminé par ses fonctions de calcul des nouveaux messages à partir des messages reçus :
initialisation : on note $\Psi_v^{(o)} : \mathbb{R} \to \mathbb{R}$ l'application d'initialisation à partir de la sortie x_i du canal, i.e avec les notations précédentes : $I_i = \Psi_v^{(o)}(x_i)$. Dans l'algorithme BP, $\Psi_v^{(o)} = \frac{2x_i}{\sigma^2}$.
traitement des parités : on note $\Psi_p : \mathbb{R}^{d_p-1} \to \mathbb{R}$ l'application de calcul des messages d'un noeud de parité à partir des informations reçues des noeuds de variable qui lui sont incidents (à part le noeud auquel le message est adressé, d'où le $d_p - 1$). Dans le cas de l'algorithme BP, $\Psi_p(m_1, .., m_{d_p-1}) = 2\tanh^{-1}(\prod_{i=1}^{d_p-1} \tanh \frac{m_i}{2})$.
traitement des variables : on note $\Psi_v : \mathbb{R} \times \mathbb{R}^{d_v-1} \to \mathbb{R}$ l'application de calcul des messages d'un noeud de variable à partir des informations reçues des noeuds de parité qui lui sont incidents (à part le noeud auquel le message est adressé, d'où le $d_v - 1$) et de l'information initial (en premier argument). Dans le cas de l'algorithme BP, $\Psi_v(m_0, m_1, ..., m_{d_v-1}) = \sum_{i=0}^{d_v-1} m_i$.
Remarque : en toute généralité, un algorithme de type *message passing* autorise les fonctions Ψ_v et Ψ_p à dépendre de l'itération l de l'algorithme.

On note $\Pi(A)$ l'ensemble des lois de probabilité sur l'espace A. Soit alors l'application ${}^\star\Psi_v : \Pi(\mathbb{R}) \times \Pi(\mathbb{R}^{d_v-1}) \to \Pi(\mathbb{R})$ telle que $\Psi_v(m_0, m_1, ..., m_{d_v-1})$ a pour loi ${}^\star\Psi_v(P_0, P_1, .., P_{d_v-1})$ si $m_i \sim P_i$ et les m_i sont indépendants. De même soit ${}^\star\Psi_p : \Pi(\mathbb{R}^{d_p-1}) \to \Pi(\mathbb{R})$ telle que $\Psi_p(m_1, .., m_{d_p-1})$ a pour loi ${}^\star\Psi_p(P_1, ..., P_{d_p-1})$ si $m_i \sim P_i$ et les m_i sont indépendants.

On va maintenant montrer que la probabilité qu'un message soit erroné à une itération fixée ne dépend pas du mot du code envoyé. On a besoin d'abord des conditions suivantes :

Définition 4.1. Conditions de symétrie (s'appliquent au canal et à l'algorithme utilisés) :

— Symétrie du canal : $p(x_i = x|c_i = 1) = p(x_i = -x|c_i = -1)$ où p est la densité de x_i.

— Symétrie des noeuds de parité : $\forall\ (b_1, ..., b_{d_p-1}) \in \{\pm 1\}$

$$\Psi_p(b_1 m_1, ..., b_{d_p-1} m_{d_p-1}) = (\prod_{i=1}^{d_p-1} b_i)\ \Psi_p(m_1, ..., m_{d_p-1})$$

— Symétrie des noeuds de variable : $\Psi_v^{(0)}(-m_o) = -\Psi_v^{(0)}(m_o)$ et

$$\Psi_v(-m_0, -m_1, ..., -m_{d_v-1}) = -\Psi_v(m_0, m_1, ..., m_{d_v-1})$$

On vérifie facilement que le canal BIAWGNC ($x_i|c_i \sim \mathcal{N}(c_i, \sigma^2)$) et l'algorithme BP vérifient ces conditions (notamment car $\tanh(-x) = -\tanh(x)$ et $\tanh^{-1}(x) = \frac{1}{2}\log\frac{1+x}{1-x}$ donc $\tanh^{-1}(-x) = -\tanh^{-1}(x)$ aussi). On a alors le résultat suivant :

Proposition 4.1. Si le canal et l'algorithme *message passing* de décodage vérifient les conditions de symétrie alors la probabilité qu'un message soit erroné à l'itération l de l'algorithme ne dépend pas du mot du code envoyé.

Preuve : on note $p(q) = p(x_i = q|c_i = 1)$ (probabilité de transition du canal). Un canal à entrée $c_i \in \{\pm 1\}$ et à sortie symétrique (selon définition 4.1) peut être remodélisé ainsi : $x_i = c_i z_i$ avec z_i iid de densité $p(q)$ et indépendants de c_i. Soit $c = (c_1, ..., c_n)$ l'entrée du canal, $z = (z_1, ..., z_n)$ la réalisation du canal et $x = (x_1, ..., x_n) = (c_1 z_1, ..., c_n, z_n)$ la sortie du canal. Soit v_i un noeud de variable quelconque et p_j un noeud de parité qui lui est lié. Soit $m_{ij}^{(h)}(x)$ le message envoyé par v_i à p_j et $m_{ji}^{(h)}(x)$ le message envoyé par p_j à v_i (notés LV_{ij} et LP_{ji} plus haut) à l'itération h de l'algorithme sachant que x a été reçu.
D'après la symétrie des noeuds de variable on a $m_{ij}^{(0)}(x) = \Psi_v^{(0)}(x_i) = c_i \Psi_v^{(0)}(z_i) = c_i m_{ij}^{(0)}(z)$. Supposons qu'à l'itération h on ait $m_{ij}^{(h)}(x) = c_i m_{ij}^{(h)}(z)$. Comme c est un mot du code alors $\prod_{k \,:\, c_k \text{ lié à } p_j} c_k = 1$. D'après la symétrie des noeuds de parité, si on indexe de 1 à $d_p - 1$ tous les noeuds de variable autres que v_i liés à p_j alors

$$\begin{aligned} m_{ji}^{(h+1)}(x) &= \Psi_p(m_{1j}^{(h)}(x), ..., m_{d_p-1,j}^{(h)}(x)) = \Psi_p(c_1 m_{1j}^{(h)}(z), ..., c_{d_p-1} m_{d_p-1,j}^{(h)}(z)) \\ &= \Big(\prod_{\substack{k \neq i \\ c_k \text{ lié à } p_j}} c_k \Big)\ \Psi_p(m_{1j}^{(h)}(z), ..., m_{d_p-1,j}^{(h)}(z)) = c_i \Psi_p(m_{1j}^{(h)}(z), ..., m_{d_p-1,j}^{(h)}(z)) \\ &= c_i m_{ji}^{(h+1)}(z) \end{aligned}$$

Donc v_i reçoit à l'itération $h+1$ les messages $c_i m_{ki}^{(h+1)}(z)$ des noeuds de parité p_k qui lui sont liés, et d'après la symétrie des noeuds de variable $m_{ij}^{(h+1)}(x) = c_i m_{ij}^{(h+1)}(z)$. On a donc prouvé par récurrence que à toute itération $h \leq l$, $m_{ij}^{(h)}(x) = c_i m_{ij}^{(h)}(z)$. Donc en particulier $m_{ij}^{(l)}(x) = c_i m_{ij}^{(l)}(z)$. Le message est errroné si son signe est différent du signe du bit c_i envoyé associé, i.e. si $c_i m_{ij}^{(l)}(x) < 0 \Leftrightarrow c_i^2 m_{ij}^{(l)}(z) < 0 \Leftrightarrow m_{ij}^{(l)}(z) < 0$, d'où le résultat $\square$

La loi des messages des noeuds de variable à une étape l dépend de la loi des x_i, donc des bits envoyés c_i et du canal (bruit b_i), mais ne dépend pas du code LDPC (d_v, d_p)-régulier utilisé tant qu'il est sans cycles de longueur inférieure ou égale à $4l$ car l'arbre du voisinage de profondeur $2l$ d'un noeud reste alors identique. Mais on vient de voir que la probabilité qu'un message de noeud de variable à l'étape l soit erroné ne dépend pas du mot de code envoyé c. Elle ne dépend donc que de σ. On peut donc supposer dans la suite que le mot envoyé est $c = (1, ..., 1)$.

On peut maintenant présenter l'algorithme d'évolution de densité pour l'algorithme BP. Plaçons nous à un noeud de variable donné. A une itération donnée, il dispose pour former un nouveau message de son information initiale $m_0 = \Psi_v^0(x_i) = \frac{2x_i}{\sigma^2} \sim P_0$ et de messages $m_1, ..., m_{d_v-1}$ provenant des noeuds de parité qui lui sont liés. Ces messages $m_1, ..., m_{d_v-1}$ sont indépendants de m_0 car fonction d'autres informations initiales que celle du noeud en question (toujours par l'absence de cycle). Donc le noeud renvoie un message $\Psi_v(m_0, m_1, ..., m_{d_v-1}) = \sum_{i=0}^{d_v-1} m_i$ qui a pour loi ${}^\star\Psi_v(P_0, P_1, .., P_{d_v-1}) = P_0 * P_1 * ...P_{d_v-1}$ (produit de convolution où $m_i \sim P_i$). De plus par l'analyse précédente les messages $m_1, ..., m_{d_v-1}$ sont iid. On appelle leur loi P. On a donc que la loi d'un message envoyé par un noeud de variable en fonction de la loi P des messages des noeuds de parité qu'il vient de reçevoir est ${}^\star\Psi_v(P_0, P, .., P) = P_0 * P * ... * P$.

La loi P_0 de $\frac{2xi}{\sigma^2}$ est continue et les fonctions Ψ_v et Ψ_p sont continues donc toutes les lois considérées des messages seront continues et on peut les identifier à leurs densités. Le produit de convolution de deux fonctions f et g est $(f * g)(x) = \int f(y) g(x - y)\, \mathrm{d}y$. La transformée de Fourier d'une fonction réelle f est $\hat{f}(s) = \mathcal{F}(f)(s) = \int_{-\infty}^{\infty} e^{-2\pi \mathrm{i} x s} f(x)\, \mathrm{d}x$. On rappelle le théorème de convolution : $\mathcal{F}(f * g) = \mathcal{F}(f)\mathcal{F}(g)$. Le théorème d'inversion de

Fourier assure que $f(x) = \int_{-\infty}^{\infty} e^{2\pi \mathrm{i} x s} \hat{f}(s)\,\mathrm{d}s$. On a donc $\mathcal{F}^{-1}(f)(x) = \int_{-\infty}^{\infty} e^{2\pi \mathrm{i} x s} f(x)\,\mathrm{d}x$. En appliquant ceci on obtient :

$$^{\star}\Psi_v(P_0, P, .., P) = \mathcal{F}^{-1}(\mathcal{F}(P_0)\mathcal{F}(P)^{d_v - 1}) \tag{2}$$

On se place à un noeud de parité pour chercher la loi des messages qu'il envoie. On sait que ce noeud utilise des messages $m_1, ..., m_{d_p-1}$ iid et que :

$$\Psi_p(m_1, ..., m_{d_p-1}) = 2\tanh^{-1}(\prod_{i=1}^{d_p-1} \tanh\frac{m_i}{2}) = \log\frac{1 + \prod_{i=1}^{d_p-1}\tanh\frac{m_i}{2}}{1 - \prod_{i=1}^{d_p-1}\tanh\frac{m_i}{2}}$$

On considère l'application $\varphi : m \in \mathbb{R} \longmapsto (\mathrm{lgsgn}(m), -\log|\tanh\frac{m}{2}|) \in \mathbb{Z}/2\mathbb{Z} \times [0, \infty[$ où $\mathrm{lgsgn}(m) = \mathbb{1}_{m<0}$. L'avantage de cette représentation $\varphi(m)$ du message m est que la transformation Ψ_p devient simplement une addition sur $\mathbb{Z}/2\mathbb{Z} \times [0, \infty[$, c'est-à-dire :

Proposition 4.2. $\varphi(\Psi_p(m_1, ..., m_{d_p-1})) = \sum_{i=1}^{d_p-1} \varphi(m_i)$.

Preuve : On écrit

$$\begin{aligned}
\varphi(\Psi_p(m_1, ..., m_{d_p-1})) &= (\ \mathrm{lgsgn}(\Psi_p(m_1, ..., m_{d_p-1}))\ , -\log|\tanh\frac{\Psi_p(m_1, ..., m_{d_p-1})}{2}|\ ) \\
&= (\ \mathrm{lgsgn}(2\tanh^{-1}(\prod_{i=1}^{d_p-1}\tanh\frac{m_i}{2}))\ , -\log|\tanh\frac{2\tanh^{-1}(\prod_{i=1}^{d_p-1}\tanh\frac{m_i}{2})}{2}|\ ) \\
&= (\ \mathrm{lgsgn}(\log\frac{1+\prod_{i=1}^{d_p-1}\tanh\frac{m_i}{2}}{1-\prod_{i=1}^{d_p-1}\tanh\frac{m_i}{2}})\ , -\log|\prod_{i=1}^{d_p-1}\tanh\frac{m_i}{2}|\ ) \\
&= (\ \mathbb{1}_{\{\prod_{i=1}^{d_p-1}\tanh\frac{m_i}{2}<0\}}\ , -\sum_{i=1}^{d_p-1}\log|\tanh\frac{m_i}{2}|\ ) \\
&= (\ \bigoplus_{i=1}^{d_p-1}\mathrm{lgsgn}(m_i)\ , -\sum_{i=1}^{d_p-1}\log|\tanh\frac{m_i}{2}|\ ) = \sum_{i=1}^{d_p-1}\varphi(m_i)
\end{aligned}$$

où on utilise $\mathrm{sgn}(\tanh\frac{m}{2}) = \mathrm{sgn}(m)$ (donc pour lgsgn aussi) pour passer à la dernière ligne $\square$

Avec des calculs simples, φ est inversible et on trouve pour tout $(s, y) \in \mathbb{Z}/2\mathbb{Z} \times [0, \infty[$ $\varphi^{-1}(s, y) = (2s-1)\log\tanh\frac{y}{2}$. On a donc $\Psi_p(m_1, ..., m_{d_p-1}) = \varphi^{-1}(\sum_{i=1}^{d_p-1}\varphi(m_i))$. On veut donc calculer la loi de $\varphi(m_i)$ à partir de la loi de m_i, ensuite utiliser le produit de convolution et la transformée de Fourier pour calculer la loi de la somme et enfin déduire la loi de $\Psi_p(m_1, ..., m_{d_p-1})$.

Si M est une v.a réelle à densité P, on cherche la loi de $\varphi(M)$. Soit $f : \mathbb{Z}/2\mathbb{Z} \times [0, \infty[\longmapsto \mathbb{R}$ intégrable quelconque, alors :

$$\begin{aligned}
\mathbb{E}[f(\varphi(M))] &= \int_{-\infty}^{\infty} f(\mathrm{lgsgn}(m), -\log|\tanh\frac{m}{2}|)P(m)\,\mathrm{d}m \\
&= \int_0^{\infty} f(0, -\log\tanh\frac{m}{2})P(m)\,\mathrm{d}m + \int_{-\infty}^{0} f(1, -\log\tanh\frac{-m}{2})P(m)\,\mathrm{d}m \\
&= \int_0^{\infty} f(0, y)\frac{1}{\sinh y}P(-\log\tanh\frac{y}{2})\,\mathrm{d}y + \int_0^{\infty} f(1, y)\frac{1}{\sinh y}P(\log\tanh\frac{y}{2})\,\mathrm{d}y
\end{aligned}$$

$$= \sum_{s=0}^{1} \int_0^\infty f(s,y) \frac{P((2s-1)\log\tanh\frac{y}{2})}{\sinh y}\,\mathrm{d}y$$
$$= \int\int_{\{0,1\}\times[0,\infty[} f(s,y) \frac{P((2s-1)\log\tanh\frac{y}{2})}{\sinh y}\,\mathrm{d}(\nu\otimes\lambda)(s,y)$$

où ν est la mesure de comptage sur $\{0,1\}$ et où on utilise les changements de variable $y = -\log\tanh\frac{m}{2}$ et $y = -\log\tanh\frac{-m}{2}$ avec le fait que $(\log\tanh\frac{y}{2})' = \frac{1}{\sinh y}$. Ainsi $\varphi(M)$ est de densité $\tilde{P}$ par rapport à $\nu\otimes\lambda$ avec

$$\tilde{P}(s,y) = \frac{1}{\sinh y} P((2s-1)\log\tanh\frac{y}{2}) \tag{3}$$

Par des calculs similaires, si le vecteur aléatoire (S,Y) à valeurs dans $\mathbb{Z}/2\mathbb{Z}\times[0,\infty[$ a la densité $\tilde{P}$ par rapport à $\nu\otimes\lambda$, alors $M = \varphi^{-1}(S,Y)$ est une v.a réelle de densité

$$P(m) = \frac{1}{\sinh|m|}\tilde{P}(\mathrm{lgsgn}(m), -\log\tanh\frac{|m|}{2}) \tag{4}$$

On veut calculer la transformée de Fourier de $f : \mathbb{Z}/2\mathbb{Z}\times[0,\infty[\longmapsto \mathbb{R}$. On la définit en faisant une transformée de Fourier discrète sur la première coordonnée (transformée de Fourier discrète : pour une suite finie $(s_0,...,s_{N-1})$, $\mathcal{F}(s_k) = \sum_{n=0}^{N-1} e^{-\frac{2\pi\mathrm{i}}{N}kn}s_n$, vérifie le théorème de convolution et donc pour une fonction g sur $\mathbb{Z}/2\mathbb{Z}$, $\mathcal{F}(g)(0) = g(0)+g(1)$ et $\mathcal{F}(g)(1) = g(0)-g(1)$) et une transformée de Fourier sur les réels ou de Laplace (de Fourier en pratique) sur la deuxième coordonnée : pour $(t,z)\in\mathbb{Z}/2\mathbb{Z}\times[0,\infty[$ on définit

$$\begin{aligned}\mathcal{F}(f)(s,y) &= \int_{\{0,1\}}\int_0^\infty e^{-\frac{2\pi\mathrm{i}}{2}st}e^{-2\pi\mathrm{i}yz}f(t,z)\,\mathrm{d}z\,\mathrm{d}\nu(t)\\ &= \int_0^\infty e^{-2\pi\mathrm{i}yz}f(o,z)\,\mathrm{d}z + e^{-\pi\mathrm{i}s}\int_0^\infty e^{-2\pi\mathrm{i}yz}f(1,z)\,\mathrm{d}z\\ &= \hat{f^0}(y) + e^{-\pi\mathrm{i}s}\hat{f^1}(y)\end{aligned}$$

où $f^0 = f(0,.)$ et $f^1 = f(0,.)$ et $\hat{f^0}$ et $\hat{f^0}$ leurs transformées de Fourier. Cette définition vérifie le théorème de convolution pour fonctions de $\mathbb{Z}/2\mathbb{Z}\times[0,\infty[$ et l'inverse est $\mathcal{F}^{-1}(f)(s,y) = \int_{\{0,1\}}\int_0^\infty \frac{1}{2}e^{\frac{2\pi\mathrm{i}}{2}st}e^{2\pi\mathrm{i}yz}f(t,z)\,\mathrm{d}z\,\mathrm{d}\nu(t)$.

Si $m_i\sim P_i$ alors $\varphi(m_i)\sim\tilde{P_i}$ densité sur $\mathbb{Z}/2\mathbb{Z}\times[0,\infty[$ calculée avec la formule (3). Si les m_i sont indépendants alors la densité de $\sum_{i=1}^{d_p-1}\varphi(m_i)$ est $\tilde{Q} = \tilde{P_1}*...*\tilde{P}_{d_p-1}$ et on a $\mathcal{F}(\tilde{Q}) = \prod_{i=1}^{d_p-1}\mathcal{F}(\tilde{P_i}) = \mathcal{F}(\tilde{P})^{d_p-1}$ car ici les m_i sont iid $\sim P$ donc les $\varphi(m_i)$ sont iid $\sim\tilde{P}$. Avec la formule et les notations plus haut ceci s'écrit pour $(s,y)\in\mathbb{Z}/2\mathbb{Z}\times[0,\infty[$

$$\mathcal{F}(\tilde{Q})(s,y) = (\hat{\tilde{P}}^0(y) + e^{-\pi\mathrm{i}s}\hat{\tilde{P}}^1(y))^{d_p-1} \tag{5}$$

On obtient $\tilde{Q}$ par inversion de Fourier. $\Psi_p(m_1,...,m_{d_p-1}) = \varphi^{-1}(\sum_{i=1}^{d_p-1}\varphi(m_i))$ alors la densité Q de $\Psi_p(m_1,...,m_{d_p-1})$ est obtenu par l'application de la formule (4) à $\tilde{Q}$.

En mettant ensemble tous ces résultats on a l'**algorithme d'évolution de densité** : On note $P^{(l)}$ la densité des messages envoyés par les noeuds de variable à la fin de l'itération $l\geq 0$ et $Q^{(l)}$ la densité des messages envoyés par les noeuds de parité au milieu de l'itération $l\geq 1$. On initialise $P^{(l)}$ à $P^{(0)} = P_0$ densité de $\frac{2X}{\sigma^2}$ où $X = c+b$ avec $\mathbb{P}(c=\pm1) = \frac{1}{2}$, $b\sim\mathcal{N}(0,\sigma^2)$ et $b\perp\!\!\!\perp c$ (densité calculable, dépend uniquement de σ). A partir de $P^{(l)}$ connue :

1. On calcule $\tilde{P}^{(l)}$ par l'équation trouvé en (3) : $\tilde{P}^{(l)}(s,y) = \frac{1}{\sinh y} P^{(l)}((2s-1)\log\tanh\frac{y}{2})$
2. On calcule $\hat{\tilde{P}}^{(l),0}$ et $\hat{\tilde{P}}^{(l),1}$ transformées de Fourier de $\tilde{P}^{(l),0} = \tilde{P}^{(l)}(0,.)$ et $\tilde{P}^{(l),1} = \tilde{P}^{(l)}(1,.)$
3. On calcule la transformée de Fourier de $\tilde{Q}^{(l+1)}$ par la formule trouvé en (5) :

$$\begin{cases} \mathcal{F}(\tilde{Q}^{(l+1)})(0,y) = (\hat{\tilde{P}}^{(l),0}(y) + \hat{\tilde{P}}^{(l),1}(y))^{d_p-1} \\ \mathcal{F}(\tilde{Q}^{(l+1)})(1,y) = (\hat{\tilde{P}}^{(l),0}(y) - \hat{\tilde{P}}^{(l),1}(y))^{d_p-1} \end{cases} \tag{6}$$

4. On calcule $\tilde{Q}^{(l+1)}$ par inversion de Fourier (via un algorithme FFT)
5. On calcule $Q^{(l+1)}$ par (4) : $Q^{(l+1)}(m) = \frac{1}{\sinh|m|}\tilde{Q}^{(l+1)}(\mathrm{lgsgm}(m), -\log\tanh\frac{|m|}{2})$
6. On a $\mathcal{F}(P^{(l+1)}) = \mathcal{F}(P_0)(\mathcal{F}(Q^{(l+1)}))^{d_v-1}$ par (2) et on trouve $P^{(l+1)}$ par une inversion de Fourier via FFT.

Comme on a vu qu'on peut supposer que le mot envoyé est $c = (1,...,1)$, la probabilité d'erreur par bit à l'étape l est $P^{(l)}(]-\infty,0[)$. On peut alors utiliser l'algorithme d'évolution de densité pour approcher numériquement la valeur du seuil σ^* seulement en deçà duquel on a un bon comporetement asymptotique (résultat 4). En effet, on peut montrer assez facilement que pour les canaux BIAWGNC, si pour un certain $\sigma > 0$, $\lim_{l\to\infty} P^{\infty}_{erreur}(l) = 0$ alors pour tout $\sigma' < \sigma$, on a aussi $\lim_{l\to\infty} P^{\infty}_{erreur}(l) = 0$. On fait donc tourner l'algorithme de densité d'évolution sur un canal de paramètre σ donné et on regarde si $P^{\infty}_{erreur}(l) = P^{(l)}(]-\infty,0[)$ semble tendre vers 0. Si oui, on sait alors que $\sigma^* \geq \sigma$. Sinon, on déduit que $\sigma^* \leq \sigma$. De plus, le seuil est commun à tous les codes (d_v,d_p)-réguliers et on cherche donc d_v et d_p avec le seuil le plus grand. Aussi, les résultats 1 et 2 (concentration et convergence) montrent que quand la longueur n tend vers l'infini, le comportement des codes (d_v,d_p)-réguliers se concentre autour du comportement qu'on vient de calculer via évolution de densité.

Ces quatres résultats mis ensemble donnent l'affirmation suivante : si on considère le problème de transmission à travers un BIAWGNC(σ) de messages codés par des codes LDPC (d_v,d_p)-réguliers (avec d_v et d_p fixés) et décodés avec l'algorithme BP, alors il existe (et on peut approcher avec l'évolution de densité) une valeur seuil σ^* du paramètre du canal (dépend uniquement de d_v et d_p) telle que si $\sigma < \sigma^*$ alors $\forall\ \varepsilon > 0$, $\exists\ l_{\varepsilon,\sigma}$ et $n_{\varepsilon,\sigma}$ tels que si on choisit aléatoirement de manière uniforme un code de $\mathcal{C}^n(d_v,d_p)$ avec $n \geq n_{\varepsilon,\sigma}$ alors avec probabilité tendant exponentiellement vers 1 en n, ce code a une probabilité d'erreur par bit après $l_{\varepsilon,\sigma}$ itérations de l'algorithme BP inférieure à ε. C'est-à-dire si $\sigma < \sigma^*$ et quand n et l tendent vers l'infini, presque tous les codes LDPC (d_v,d_p)-réguliers (exponentiellement en n) sont aussi fiable que l'ont veut. Mais si $\sigma > \sigma^*$, presque tous ne sont pas fiables.

Ces résultats se généralisent aux codes LDPC non réguliers qui ont de meilleurs performances. En particulier, si on note λ_i la proportion d'arrêtes incidentes à des noeuds de variable de degré i, ρ_i la proportion d'arrêtes incidentes à des noeuds de parité de degré i, $d_v^{\max}$ le degré maximal d'un noeud de variable et $d_p^{\max}$ le degré maximal d'un noeud de parité, alors on peut définir les polynomes λ (appelé profil d'irrégularité des variables) et ρ (appelé profil d'irrégularité des parités) :

$$\lambda(x) = \sum_{i=2}^{d_v^{\max}} \lambda_i x^{i-1} \text{ et } \rho(x) = \sum_{i=2}^{d_p^{\max}} \rho_i x^{i-1}$$

On a alors que les seuls changements dans l'algorithme d'évolution de densité sont que les

équations (6) deviennent :

$$\begin{cases} \mathcal{F}(\tilde{Q}^{(l+1)})(0,y) = \rho(\hat{\tilde{P}}^{(l),0}(y) + \hat{\tilde{P}}^{(l),1}(y)) \\ \mathcal{F}(\tilde{Q}^{(l+1)})(1,y) = \rho(\hat{\tilde{P}}^{(l),0}(y) - \hat{\tilde{P}}^{(l),1}(y)) \end{cases}$$

et que la formule finale pour $\mathcal{F}(P^{(l+1)})$ devient $\mathcal{F}(P^{(l+1)}) = \mathcal{F}(P_0)\lambda(\mathcal{F}(Q^{(l+1)}))$. On peut alors ici aussi approcher par évolution de densité le seuil σ^* qui dépend des profils d'irrégularité (donc de la distribution des degrés dans le graphe) et chercher la distribution de degrés qui a le seuil le plus élevé.

Nous verrons dans la partie suivante des exemples de mise en oeuvre de codes LDPC et de débruitage à l'aide de l'algorithme de propagation de croyance.

4 Codes LDPC et algorithme BP en pratique

Nous nous intéreserons ici à la mise en pratique de toute la théorie qui précède.

4.1 Sommaire

Pour commencer, il est important de préciser quelques appellations que nous utiliserons par la suite, ici exposées par ordre d'intervention dans le déroulement du processus de simulation d'échange d'information dans le canal.
Ainsi nous appellerons :

1. « message initial » le message original, décidé par l'expéditeur.
2. « message codé » le message initial une fois codé par la matrice de codage
3. « message bruité » le message codé et bruité par le canal, c'est-à-dire le message brut reçu par le destinataire.
4. « message débruité », ou « message estimé », le message que le destinataire obtient après estimation de ses bits par algorithme de propagation de croyance.
5. « message débruité naïf » le message débruité simplement par arrondi sur chacun de ses bits.
6. « message décodé » le message estimé décodé par la matrice de codage.

La mise en œuvre de certaines parties du traitement des messages nécessite l'applications de méthodes « gloutonnes » comme le pivot de Gauss, pour éviter les calculs trop longs, nous découpons les messages en messages plus petits et appliquons l'algorithme de propagation de croyace à chacun d'entre eux, cela revient à l'appliquer au message entier avec une matrice de contrôle correspondant à un graphe de Tanner constitué de sous-graphes indépendants.
Il est aussi important de remarquer l'utilisation du package pyldpc dont nous nous servons pour trouver les matrices de codage à partir des matrices de contrôle ainsi que pour décoder des messages codés par une matrice génératrice.

4.2 Simulation de décodage d'images

Ici nous nous chercherons à visualiser le décodage et débruitage d'un message. Nous prendrons donc tout naturellement une image et nous verrons l'évolution de l'image bruitée sur plusieurs itérations de l'algorithme BP.
Nous utiliserons des images de nombres écrits à la main issues de la base de données du MNIST.

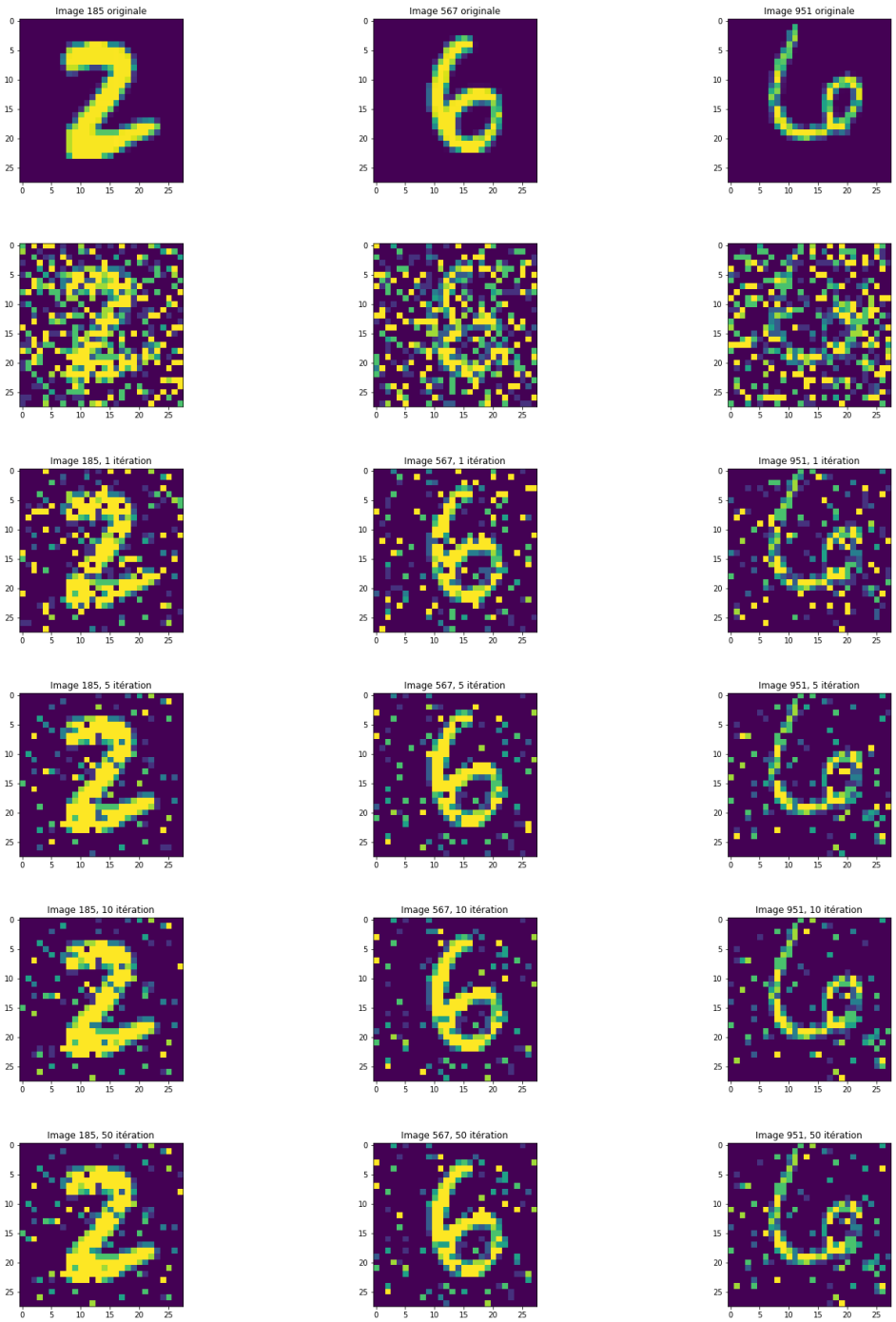


TABLE 1 – Images originales en première ligne, Images débruitées naïvement en seconde ligne, puis de haut en bas résultat après 1, 5, 10 et 50 itérations de l'algorithme de propagation de croyance

On constate que l'algorithme de propagation de croyance permet de conserver beaucoup d'information le débruitage converge très vite vers un rendu imparfait mais néanmoins relativement fidèle à l'original. À noter que les performances du débruitage dépendant grandement de la structure de la matrice de contrôle correspondant au code, et donc du graphe de Tanner sous-jacent.
Dans les exemples çi dessus, chaque pixel est codé avec 3 bits pour un total de $28 \times 28 = 764$ pixels ; on considère qu'un pixel est un message en soi. Le code est déterminé par la matrice de contrôle suivante, choisie pour l'absence de cycles d'ordre 4 dans son graphe de Tanner :

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

Par la suite nous allons analyser le taux d'erreur d'un code LDPC régulier en fonction de la taille de sa matrice de contrôle.

4.3 Analyse empirique du taux d'erreur

Ici nous construisons des matrices de Gallagher aléatoires, ce sont des matrices de contrôle de codes réguliers, la construction de ces matrices est explicitée dans Gallager (1962) [3].
On pour une taille de message donné nous générons une multitude me matrices de contrôle et un message par matrice. Ces messages une fois codés, bruités, estimés et décodés sont comparés aux originaux pour estimer le taux d'erreur moyen et son évolution en fonction de la taille du mot.
Le graphe suivant représente le taux d'erreur en fonction de la taille du mot avec des codes de Gallager réguliers $(3, 6)$.

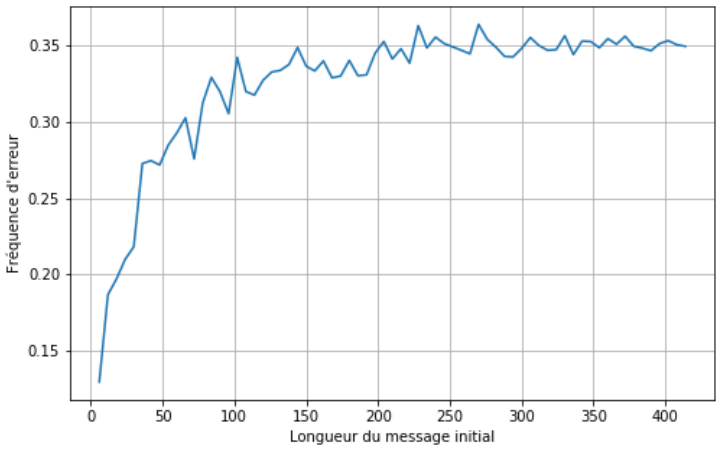


FIGURE 1 – Évolution du taux d'erreur en fonction de la longueur des messages

La simulation est réalisée en tirant au hasard une matrice de Gallager de taille $(n*(d_c*d_v))$, en construisant un mot au hasard, en le codant et en lui appliquant un bruit gaussien de variance $\sigma = 1$ puis en comparant la sortie de l'algorithme de propagation de croyance et le mot original. Ceci 100 fois pour tout n. La courbe croit, et cela est concordant avec les résultats faits en 3.4 car le nombre d'itération est fixe et σ est supérieur à la valeur de seuil pour les codes réguliers $(3, 6)$ qui est 0.88 selon [8].

Le taux d'erreur converge vers une valeur proche de $\frac{d_v}{d_c}$ ce qui est cohérent avec les résultats de Berrou [1].

Ainsi nous avons mis en évidence la cohérence entre aspect théorique et aspect pratique des codes LDPC réguliers ainsi que de l'algorithme de propagation de croyance.

Conclusion

Les codes LDPC sont aujourd'hui omniprésents dans les communications, leurs performances accrues dépassent de loin celles des codes linéaires ou cycliques. Après avoir passé en revue les rudiments théoriques de la théorie de l'information de Shannon ainsi que des codes correcteurs d'erreurs, nous avons pu exhiber un code linéaire et en démontrer le fonctionnement. Les codes LDPC ont ensuite étés introduits, nous avons porté un intérêt tout particulier à la construction de l'algorithme de propagation de croyance ou algorithme BP. L'algorithme BP permettant d'estimer la valeur des bits d'un message bruité en exploitant l'interdépendance de ces derniers, interdépendance induite par le codage lui même.
Ainsi nous avons présenté et démontré les formules régissant le fonctionnement de l'algorithme BP avant d'explorer la convergence même de ce dernier par l'analyse asymptotique de la probabilité d'erreur bit par bit en fonction du nombre d'itérations opérées.
Pour terminer nous avons implémenté l'algorithme BP ainsi qu'une multitude de codes LDPC afin d'illustrer les propos précédents et d'analyser l'évolution des performances en terme de taux d'erreur de l'algorithme BP en fonction de la taille des messages.
Pour conclure, il semble qu'après les recherches menées, la grande difficulté serait de voir comment construire un code LDPC aussi efficace que possible. Une autre question serait de savoir comment croiser les codes LDPC pour réduire l'erreur tout en gardant un temps de calcul raisonnable.

Annexe

Implémentation en python d'un code linéraire

```
import numpy as np
import math, random

class LinearCode :
def __init__(self,generator):
self.G=generator
self.k = np.shape(generator)[0]
self.n = np.shape(generator)[1]
self.H=self.control_matrix()
self.dmin=self.dmin() # distance minimale du code
self.e=math.floor((self.dmin - 1) / 2) # capacité de correction du code
self.table,self.errors=self.tab() # erreurs corrigeables et leurs syndromes
def control_matrix(self): # matrice de controle
M = self.G[:,self.k:]
MT = np.transpose(M)
I = -1*np.eye(self.n - self.k) % 2
H = np.concatenate((MT,I), axis=1)
return H.astype(int)
def code(self,m): # code un mot
c = m.dot(self.G) % 2
return c.astype(int)
def syndrome(self,r) : # syndrome du message reçu
rT = np.transpose(r)
s = self.H.dot(rT) % 2
return s.astype(int)
def dmin(self): # distance minimale du code (poids minimal des mots non nuls du code)
M = np.empty([2 ** self.k, self.k]) # matrice de tous les messages
for i in range(0, 2 ** self.k):
M[i]=np.flipud( np.fromstring(np.binary_repr(i, width=self.k), 'u1')-ord('0') )
M=M.astype(int)
dmin = self.n;
for m in M:
c = self.code(m)
w=np.count_nonzero(c)
if w != 0 and w < dmin : dmin = w
return dmin
def tab(self): # erreurs corrigeables et leurs syndromes
table = {}
E = np.empty([2 ** np.shape(self.H)[0], self.n])
count = 0
for i in range(0, 2 ** self.n):
e = np.flipud( np.fromstring(np.binary_repr(i, width=self.n), 'u1')-ord('0') )
if np.count_nonzero(e) <= self.e and count < E.shape[0]:
E[count] = e
syndrome = self.syndrome(e)
s = ''
for j in range(np.shape(self.H)[0]): s += str(int(syndrome[0,j]))
```

```
table[s] = e
count += 1
if count >= E.shape[0]: break
return table, E.astype(int)
def print_table(self) :
print('syndromes : ')
for k,v in self.table.items() :
print(k,' : ',v)
def decode(self, r): # decode le message
syndrome = self.syndrome(r)
s = ''
for j in range(np.shape(self.H)[0]): s += str(int(syndrome[j,0]))
error = self.table[s]
c = (r - error) % 2
return c

def exemple(G,m) :
lin_code=LinearCode(G)
print('G : \n',lin_code.G)
print('H : \n',lin_code.H)
print('dmin : ',lin_code.dmin)
print('e : ',lin_code.e)
print('erreurs : \n', lin_code.errors)
lin_code.print_table()
print('m : ', m)
c=lin_code.code(m)
print('c : ', c)
r=c.copy()
r[0,random.randint(0,lin_code.n-1)]+=1
r = r % 2 # message bruité
print('r : ', r)
e= (r - c) % 2 # erreur
print('e : ', e)
s=lin_code.syndrome(r)
print('s : ', np.transpose(s))
d=lin_code.decode(r)
print('d : ', d)
l=d[0,:lin_code.k].copy() # partie du message décodé correspondant au mot initial
print('l : ', l)
print(l==m)

G=np.matrix('1 0 0 0 0 1 1; 0 1 0 0 1 0 1; 0 0 1 0 1 1 0; 0 0 0 1 1 1 1 ')
m=np.matrix('0 1 0 1')
exemple(G,m)
```

Ce code affiche :

```
G :
[[1 0 0 0 0 1 1]
[0 1 0 0 1 0 1]
[0 0 1 0 1 1 0]
[0 0 0 1 1 1 1]]
```

```
H :
[[0 1 1 1 1 0 0]
[1 0 1 1 0 1 0]
[1 1 0 1 0 0 1]]
dmin :  3
e :  1
erreurs :
[[0 0 0 0 0 0 0]
[1 0 0 0 0 0 0]
[0 1 0 0 0 0 0]
[0 0 1 0 0 0 0]
[0 0 0 1 0 0 0]
[0 0 0 0 1 0 0]
[0 0 0 0 0 1 0]
[0 0 0 0 0 0 1]]
syndromes :
110  :  [0 0 1 0 0 0 0]
101  :  [0 1 0 0 0 0 0]
001  :  [0 0 0 0 0 0 1]
100  :  [0 0 0 0 1 0 0]
010  :  [0 0 0 0 0 1 0]
000  :  [0 0 0 0 0 0 0]
111  :  [0 0 0 1 0 0 0]
011  :  [1 0 0 0 0 0 0]
m :  [[0 1 0 1]]
c :  [[0 1 0 1 0 1 0]]
r :  [[0 1 1 1 0 1 0]]
e :  [[0 0 1 0 0 0 0]]
s :  [[1 1 0]]
d :  [[0 1 0 1 0 1 0]]
l :  [[0 1 0 1]]
[[ True  True  True  True]]
```

Implémentation Python d'un algorithme BP et ensemble de fonctions permettant de traiter des messages

```
import numpy as np
import math
import pyldpc
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import pandas as pd

def random_message(bits):
    "Renvoie un message aléatoire binaire de longueur bits"
    return(np.random.randint(2, size = bits))

def code_message(tG, message):
    "Code un message avec une matrice de codage G, tG sa transposée"
    return(np.dot(message, np.transpose(tG)))
```

```
def add_to_mult(message):
    "Transforme un message dont les bits sont dans

{0, 1} en un message où les bits sont dans {-1, 1}"
    return(np.power(-1, message))

def mult_to_add(message):
    "Inverse de add_to_mult"
    return(np.array((message < 0)).astype(int))

def add_noise(message):
    "Ajoute un bruit gaussien à un message"
    return(message + np.random.normal(0, 1, size = message.shape))

def random_control_matrix(n, d_c, d_v):
    "Renvoie une matrice de contrôle d'un code régulier par la méthode de Gallagher"
    if n%d_c:
        print('d_c doit diviser n')
    else:
        alpha = n//d_c
        sub_H = np.zeros((alpha, n), dtype = int)
        for i in range(alpha):
            sub_H[i] = np.array(i*(d_c*[0]) + d_c*[1] + [0 for j in range(n-(i+1)*d_c)])
        k = 1
        H = sub_H
        rows = []
        while k<d_v:
            new_row = np.transpose(np.transpose(sub_H)[[np.random.permutation(n)]])
            H = np.vstack((H,np.transpose(np.transpose(sub_H)[[np.random.permutation(n)]])))
            k+=1
    return(H, pyldpc.CodingMatrix(H))

def show_img(image, title = 0):
    "En argument: une image sous forme de vecteur"
    "Permet de représenter graphiquement l'image IM"
    chunks = [image[28*i:28*(i+1)] for i in range(28)]
    chunks = np.array(chunks)
    imgplot = plt.imshow(chunks)
    if title:
        plt.title(title)
    plt.show()

def represent_image(image, pxbit):
    "Prend en argument un vecteur représentant une image, chaque pixel étant codé en n bits."
    "Réduit éventuellement le nombre de bits nécéssaires à coder chaque pixel à pxbits"
    "Renvoie une matrice dont les lignes sont les représentations binaires de chaque pixel"
    "image must be an array of shape (n_pixls,) or (1, npixls)"
    img = np.reshape(image, (max(image.shape),))
    M = max(img)
    imgbit = np.round(np.log(M+1)/np.log(2))
    if pxbit > imgbit:
```

```
        print("Must code in less bits than the original image")
        return
    img = np.floor(img/(2**int(imgbit-pxbit))).astype(int)
    pxmat = np.zeros((len(img), pxbit)).astype(int)
    for i in range(len(img)):
        pxmat[i] = [0]*(2+pxbit-len(bin(img[i]))) + [int(i) for i in bin(img[i])[2:]]

    return(pxmat)

def convert_to_decimal(matrix):
    "Prend en argument une matrice dont chaque ligne binaire d'un pixel et"
    "renvoie le vecteur correspondant converti en décimal"
    vect = []
    d = matrix.shape[1]
    for i in range(matrix.shape[0]):
        vect.append(sum(matrix[i,j]*(2**(d-j-1)) for j in range(d)))
    return(vect)

def naive_estimation(image):
    dec = np.zeros(image.shape)
    for i in range(image.shape[0]):
        dec[i] = np.sign(image[i])
    return(dec)

def belief_propagation(message, control_matrix, sigma_error = 1, iter_max = 100):
    "1er algorithme de débruitage, décrit dans le rapport"
    parity_card, length_code = control_matrix.shape

    ### INITIALISATION ###
    Z = np.zeros((length_code, parity_card))
    Z_tot = np.zeros(length_code)
    L_tot = np.zeros(length_code)
    I = 2*message/sigma_error**2
    L = np.transpose(np.array([I for i in range(parity_card)]))
    N = 0
    while N < iter_max:

        N += 1
        ### message variable vers parité ###
        for i in range(parity_card):
            for j in range(length_code):
                if control_matrix[i,j]:
                    product = []
                    for k in range(length_code):
                        if control_matrix[i,k] and k!=j:
                            product.append(np.tanh(0.5*L[k,i]))
                    prod = np.prod(product)
                    if (abs(prod) == 1):
                        Z[j,i] = np.sign(prod)
                    else:
                        Z[j,i] = np.log((1+prod)/(1-prod))
```

```
        ### message parité vers variable ###
        for j in range(length_code):
            S = 0
            for i in range(parity_card):
                S += Z[j,i]
            Z_tot[j] = S
            for i in range(parity_card):
                L[j,i] = I[j] + Z_tot[j] - Z[j,i]

    L_tot = I + Z_tot
    return(np.sign(L_tot))



def decode_BP(message, control_matrix, sigma_error = 1, iter_max = 100):
    "Deuxième algorithme de débruitage, décrit dans le rapport"
    m,n = control_matrix.shape


    ### INITIALISATION ###
    I = 2*message/sigma_error**2
    L = np.zeros(shape = (m,n))
    Lr = np.zeros(shape = (m,n))
    Iter = 0
    while(True):
        Iter += 1
        ### message noeuds de parité -> variables ###
        for i in range(m):
            for j in range(n):
                if control_matrix[i,j]:
                    product = []
                    for k in range(n):
                        if control_matrix[i,k] and k!= j:
                            if Iter == 1:
                                product.append(np.tanh(0.5*I[k]))
                            else:
                                product.append(np.tanh(0.5*L[i,k]))
                    product = np.prod(product)
                    if abs(product) == 1:
                        X = np.sign(product)
                        Lr[i,j] = X
                    else:
                        X = np.log((1+product)/(1-product))
                        Lr[i,j] = X


        ### message variable -> noeuds de parité ###
        for j in range(n):
            for i in range(m):
                if control_matrix[i,j]:
                    Sum = 0
```

```
                    for k in range(m):
                        if control_matrix[k,j] and k!= i:
                            Sum += Lr[k,j]

                L[i,j] = I[j] + Sum


        if Iter >= iter_max:
            break
    L_fin = np.zeros(n)
    for j in range(n):
        S = 0
        for i in range(m):
            if control_matrix[i,j]:
                S += Lr[i,j]
        L_fin[j] = S + I[j]
    return(np.sign(L_fin))


def estimate(message, control_matrix, iter_max = 10, alg = 2):
    "Fonction prenant en argument une matrice dont les lignes représentent un message

chacune, bruité"
    "Permet de débruiter tous les messages un par un et de concaténer le résultat"
    if alg not in [1, 2]:
        return('alg must be in {1,2}')
    result = np.zeros(message.shape)
    if alg == 2:
        for i in range(message.shape[0]):
            result[i] = decode_BP(message[i], control_matrix, iter_max= iter_max)
    else:
        for i in range(message.shape[0]):
            result[i] = belief_propagation(message[i], control_matrix, iter_max= iter_max)
    return(result)

def decode_msg(message, tG):
    "Renvoie le message décodé par la matrice tG"
    decod = np.zeros((message.shape[0], tG.shape[1]))
    for i in range(message.shape[0]):
        decod[i] = pyldpc.DecodedMessage(tG=tG, x= message[i])
    return(decod)
```

Le process pour simuler le transfert d'une image peut être le suivant (par exemple pour l'image 567 du dataset MNIST de test :

```
H = np.array([[1,1,0,1,0,0,0,0], [1,0,1,0,1,0,0,0],
[0,1,1,0,0,1,0,0], [1,0,0,0,1,0,1,0], [0,1,0,0,1,0,0,1]])
tG = tG = pyldpc.CodingMatrix(H)

data = pd.read_csv('./mnist_test.csv')
```

```
data.drop('7', axis = 1, inplace = True)
data = data.as_matrix()
IMAGE = data[567]
img = represent_image(IMAGE, pxbit=3)
img_coded = code_message(message=img, tG=tG)
img_modulated = add_to_mult(img_coded)
img_noisy = add_noise(img_modulated)
img_estimated = estimate(img_noisy, H, iter_max=5)
img_unmodulated = mult_to_add(img_estimated)
img_decoded = decode_msg(img_unmodulated, tG = tG)
img_decoded = convert_to_decimal(img_decoded)
img = convert_to_decimal(img)
show_img(img)
show_img(img_decoded)
```

Références

1. Claude Berrou. *Codes et turbocodes*. Springer Science & Business Media, 2007.

2. D.J.C. Mackay. Turbo codes are low-density parity-check codes. `http://www.inference.org.uk/mackay/turbo-ldpc.pdf`, 1998. First draft in 1998, accessed : 28-01-2018.

3. Robert Gallager. Low-density parity-check codes. *IRE Transactions on information theory*, 8(1) :21–28, 1962.

4. H. Janati. Simulation of ldpc codes and applications. `https://github.com/hichamjanati/pyldpc/`. accessed : 02-05-2018.

5. R. W. Hamming. Error detecting and error correcting codes. *Bell Labs Technical Journal*, 29(2) :147–160, 1950.

6. O. Papini and J. Wolfmann. Algèbre discrète et codes correcteurs. 1995.

7. I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2) :300–304, 1960.

8. Thomas J Richardson and Rüdiger L Urbanke. The capacity of low-density parity-check codes under message-passing decoding. *IEEE Transactions on information theory*, 47(2) :599–618, 2001.

9. C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(1) :379–423, 1948.

10. W. Weaver. Recent contributions to the mathematical theory of communication. *The Mathematical Theory of Communication*, pages 1–28, 1949.