Table des matières

Chapitre 1

Les canaux de communication

1.1 Introduction

Ces 70 dernières années, les machines ont bouleversé nos façons de traiter l'information : que ce soit pour la stocker ou bien la transmettre. Smartphones, télévisions, radios, ordinateurs, internet, satellites, objets connectés... Aujourd'hui, les technologies de l'information et de la communication (TIC) occupent une place majeure dans notre quotidien.

La figure 1.1 représente les différentes étapes pour transmettre une information. Notons que le codage source et le codage canal ainsi que leurs opérations inverses concernent aussi bien la transmission que le stockage d'informations.

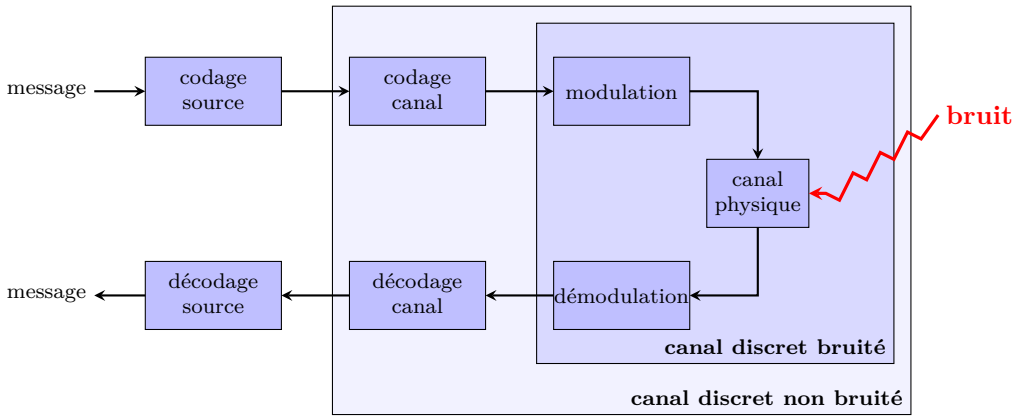


Figure 1.1 – Schéma de transmission numérique.

Le codage source. Le message que nous souhaitons transmettre peut prendre diverses formes ; par exemple, il peut être un texte, un son, une image, une vidéo ou encore une interface graphique interactive. La première étape du schéma de transmission numérique est le codage source qui consiste à transformer ce message en une suite de symboles d'un alphabet fini (généralement une suite de symboles binaires, ou bits). Le codage source peut éventuellement comprendre une opération de compression ainsi que des opérations cryptographiques.

Le codage canal. Le but principal de cette étape est de gérer les erreurs engendrées par le canal physique utilisé pour transporter l'information. Les supports physiques que nous utilisons sont très variés : l'air, des conducteurs électriques, de la fibre optique, des

bandes magnétiques, des disques en aluminium, verre ou céramique... Cependant, tous ces media sont imparfaits et peuvent altérer l'information. Le destinataire doit alors être en mesure de détecter ou même corriger les erreurs présentes dans le message qu'il reçoit. Pour cela, l'émetteur applique un code correcteur d'erreurs qui ajoute de la redondance au message. Un code correcteur de longueur n sur un alphabet $\mathscr{A}$ est simplement un ensemble de mots autorisés parmi l'ensemble $\mathscr{A}^n$ des mots possibles. Corriger les erreurs d'un mot reçu consiste à trouver le mot de code qui lui est le plus proche ; par exemple, celui qui diffère sur le minimum de positions. Nous parlons alors de décodage par maximum de vraisemblance.

En pratique, les mots d'un code respectent une structure mathématique bien précise. Celle-ci permet d'une part de limiter certaines problématiques mémoires liées au stockage du code lui-même et d'autre part de corriger efficacement les mots reçus. Il existe de nombreux codes correcteurs d'erreurs et ceux-ci ont chacun leurs propres spécificités : rendement (rapport entre la longueur d'un message et la longueur d'un message codé), capacité de correction, résistance face à différents types d'erreurs (erreurs isolées, en rafales...), efficacité de l'algorithme de décodage... Le bon choix d'un code dépend alors du canal physique utilisé pour transporter l'information mais aussi des contraintes de qualité et de débit que l'on se fixe.

Noter que le codage canal ne se résume pas au code correcteur d'erreur. En effet, toutes les transformations numériques visant à mieux appréhender le canal physique font parties du codage canal. Par exemple :

— l'entrelaceur (*interleaver*) va permuter les symboles numériques pour mieux gérer les paquets d'erreurs ;

— l'embrouilleur (*scrambler*) va randomiser la donner pour éviter les flux de données constants ;

— le mapping va associer un symbole numérique à un symbole analogique.

La modulation. Avant de transiter sur le canal physique, le message codé est modulé ; c'est-à-dire que la suite numérique que forme ce message est transformée en un signal (par exemple une onde électromagnétique, un courant électrique...) adapté au canal physique utilisé. L'émetteur peut alors envoyer ce signal via ce canal.

Le destinataire traduit finalement le signal qu'il reçoit en inversant les différentes opérations effectuées par l'émetteur : démodulation, décodage canal et décodage source.

1.2 Petit focus sur le codage canal

Regardons un peu plus en détail ce qui constitue le codage canal.

1.2.1 L'entrelaceur (*interleaver*)

En pratique, on a souvent des flots d'erreurs (bruit plus important pendant une période relativement courte, rayure sur un disque, ...). Les codes correcteurs d'erreurs gèrent très mal ce type d'erreur. Pour répartir les erreurs plus équitablement, on peut permuter les symboles :

→ soit une permutation sur un seul mot de code. Cela permet de répartir les erreurs uniformément dans le mot,

→ soit une permutation sur plusieurs mots de code. Cela permet de distribuer les erreurs uniformément sur les mots reçus.

Il existe divers algorithmes d'entrelacement. Le plus utilisé en pratique est l'entrelaceur ligne/colonne. Il consiste à remplir un tableau bi-dimensionnel à c colonnes et r lignes. On remplit le tableau ligne par ligne, de gauche à droite et de haut en bas. Mais on lit la nouvelle suite binaire en lisant les symboles binaires dans le tableau colonne par colonne, de haut en bas et de gauche à droite :

entrée	**tableau**					**sortie**
	0	0	0	1	1	
	1	1	1	1	1	
00011111110000000000	0	0	0	0	0	01000100010011001100
	0	0	0	0	0	

Figure 1.2 – Exemple d'entrelaceur ligne/colonne avec 4 lignes et 5 colonnes.

1.2.2 L'embrouilleur (*scrambler*)

Imaginons que l'on doive lire une suite très longue de bits. Il est plus facile de savoir où l'on en est dans notre lecture si les bits varient régulièrement plutôt que si l'on est face à une longue suite de 0 ou de 1. On pourrait alors facilement sauter un symbole sans s'en rendre compte... Le démodulateur est confronté au même problème (perte de synchronisation). C'est pourquoi, une des opérations du codage canal consiste à XORer/additionner l'information avec une chaine pseudo-aléatoire déterminée (souvent un LFSR (*Linear Feedback Shift Register*)). Le but de cette opération n'est pas cryptographique et donc les faiblesses des LFSR face à des attaques de type Berlekamp-Massey ne sont pas un problème dans ce contexte.

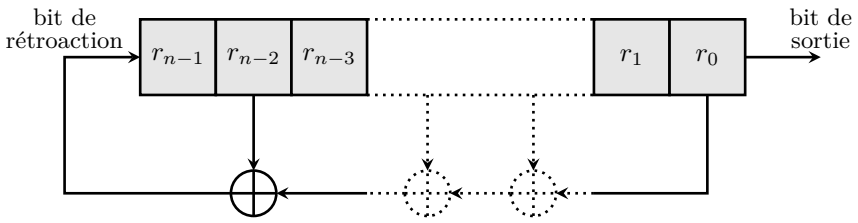


Figure 1.3 – Représentation d'un LFSR sous forme de circuit.

1.2.3 Le mapping

Il permet d'associer une valeur numérique à chaque symbole transporté par le canal physique. Voici deux exemples de mapping :

Exemple 1. L'information à transporter est une suite de bits et le canal physique est un cable de cuivre dans lequel un courant *passe* ou *ne passe pas*. Il y a alors 2 mapping possible :

symbole analogique	**symbole numérique**
courant passe	1
courant ne passe pas	0

ou

symbole analogique	**symbole numérique**
courant passe	0
courant ne passe pas	1

Exemple 2. Un canal portant une onde électromagnétique (onde radio, fibre optique...) peut transporter plus de 2 symboles différents. Les communications ne sont donc pas forcément binaires... En effet, avec une modulation Amplitude/Fréquence, chaque symbole analogique correspond à un couple (Amplitude, Fréquence) et correspondra donc à une onde qui aura ces paramètres. L'ensemble des couples (Amplitude, Fréquence) autorisés forment une **constellation** que l'on peut représenter sur le plan réel :

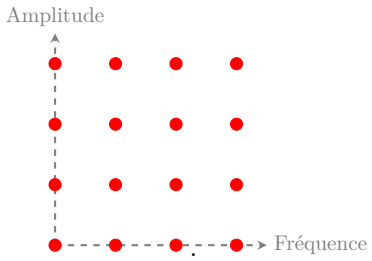


Figure 1.4 – Exemple d'une constellation 16-QAM (*Quadratique Amplitude Modulation*)

Sur la figure 1.4, les 16 points rouges correspondent à 16 symboles analogiques pouvant être transmis. Chacun de ces symboles doit être associé à un symbole numérique. Étant donné qu'il y a 16 symbole, on prendra les symboles numériques dans $\mathbf{F}_2^4$ (que l'on représente en décimal sur la figure 1.5).

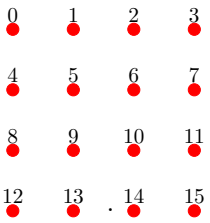


Figure 1.5 – Exemple de mapping pour une 16-QAM

Pour une constellation de N symboles, il y aura donc $N!$ mapping possibles. En particulier, pour la 16-QAM, il y a $16! \approx 2^{44}$ mapping possibles.

Un bon mapping est un mapping qui minimise le nombre de bits différents entre 2 symboles proches. En effet, une erreur de transmission correspondra souvent au fait qu'un symbole a été confondu avec un de ces proches voisins.

Définition 1.2.1 (Mapping de Gray)**.** *Si le mapping d'une constellation est tel que tous les couples de points voisins (à distance minimal) sont associés à des symboles binaires qui ne diffèrent que d'un bit, alors on dit que c'est un* ***mapping de Gray****.*

Avec un mapping de Gray, on minimise le nombre d'erreurs binaires puisque si on se trompe de symbole analogique (point de constellation), alors le symbole substitué sera généralement un des plus proches voisins et donc les symboles binaire associés ne différeront que d'un bit parmi les 4.

Remarque 1.2.1. Les systèmes de télécommunication les plus moderne peuvent aller jusqu'à 2048-QAM, soit des symboles numériques sur 11 bits!!!

1.2.4 Le code détecteur ou correcteur d'erreurs

Cette opération consiste à ajouter de la **redondance** à l'information dans le but de pouvoir détecter ou corriger d'éventuelles erreurs. C'est l'opération inverse d'une compression : là où une compression enlève de l'information inutile redondante, les code correcteur d'erreurs eux ajoute de l'information inutile (qui deviendra utile en cas d'erreurs).

Il y a 3 stratégie possibles :

1. On n'applique aucun code correcteur : l'information est transmise avec des erreurs mais ce n'est pas grave. Par exemple, si l'information est une image, une vidéo, un son... alors l'information reçue aura quelques pixels erronés ou le son sera très légèrement modifié mais cela n'empêchera pas de visualiser l'image ou de comprendre l'audio (téléphone analogique, fax...).
2. On applique un code détecteur d'erreurs. Un checksum ou un haché permet de détecter la présence d'erreurs mais ne permet pas de corriger les erreurs en question. On peut éventuellement demander une retransmission du message en cas d'erreurs (c'est un moyen de transformer un code détecteur d'erreurs en code correcteur d'erreurs).
3. On applique un code correcteur d'erreurs. On accepte d'ajouter de la redondance et donc d'augmenter la taille de la donnée à transmettre. Et on prend le temps de corriger les éventuelles erreurs s'il y en a.

Les erreurs auxquelles doivent faire face les codes correcteurs suivent un modèle qui dépend du canal physique de transmission et du mapping. Soit $\mathcal{A}$ l'ensemble des symboles numériques pouvant être émis. Soit X la variable aléatoire sur $\mathcal{A}$ représentant le symbole émis et soit Y la variable aléatoire représentant le symbole analogique reçu. La connaissance fine du canal de transmission ainsi qu'une démodulation appropriée permettra généralement d'obtenir en réception un vecteur de probabilités pour chaque symbole analogique reçu :

$$\Big(\mathbb{P}(X = a|Y)\Big)_{a\in\mathcal{A}}$$

Remarque 1.2.2. Plutôt que ce vecteur de probabilité, on travaillera plutôt avec des vecteurs de LLR (*Log Likehood Ratio*).

$$\left(\frac{\log_2\left(\mathbb{P}(X = a|Y)\right)}{\log_2\left(\mathbb{P}(X = 0|Y)\right)}\right)_{a\in\mathcal{A}}$$

Décodage Souple (*Soft Decoding*). L'utilisation de ces vecteurs de probabilités ou de ces LLR permet d'avoir une information fine de ce qui est reçu. Si nos algorithmes de correction d'erreurs utilisent ces informations fines, alors on parlera de Décodage Souple.

Décodage Dur (*Hard Decoding*). Plutôt que de considérer le vecteur de probabilités d'un symbole reçu, on peut considérer l'unique symbole dans $\mathcal{A}$ qui a la meilleure probabilité d'être le symbole émis. On considère alors soit que le symbole est juste, soit qu'il est faux (on a donc perdu de l'information par rapport aux vecteurs de probabilités ou aux LLR). Un algorithme de correction d'erreurs qui considère ce type d'information est appelé Décodeur Dur.

1.3 Les canaux de communication et les modèles d'erreur

Les codes correcteurs d'erreurs ont été créés pour permettre de transporter de l'information sans erreur malgré l'imperfection des canaux de communication à notre disposition.

En effet, pour transporter de l'information, de nombreux supports physiques peuvent être utilisés : l'air, des conducteurs électriques, de la fibre optique, des bandes magnétiques, des disques en aluminium, verre ou céramique... Quelle que soit la nature de ce support, il est imparfait ; c'est-à-dire que l'information est altérée. Le destinataire doit être en mesure d'interpréter les messages qu'il reçoit. D'où l'importance des codes correcteurs d'erreurs qui ajoutent de la redondance et permettent ainsi de détecter voir même de corriger d'éventuelles erreurs.

Avant d'aborder les codes correcteurs d'erreurs, nous allons brièvement discuter de la nature de l'erreur à corriger. Pour cela, nous commencerons par introduire des notions de la théorie de l'information introduites par Shannon. Puis nous décrirons différents canaux de communication parmi les plus couramment utilisés pour modéliser le bruit produit lors d'une transmission. Nous comparerons alors les modèles d'erreurs associés à ces canaux avec un autre modèle d'erreur particulièrement étudié en cryptologie.

1.3.1 L'entropie d'une source discrète

En 1948, Shannon pose les fondements de la théorie de l'information dans [Sha48]. Une des quantités les plus importantes qu'il introduit est l'entropie d'une source discrète. Soit une source produisant un symbole d'un alphabet $\mathscr{A}$. La sortie de la source est vue comme une variable aléatoire discrète X. Shannon définit alors l'entropie $H_q(X)$ de la source discrète modélisée par X comme la quantité d'information moyenne que cette source peut porter. Plus concrètement, c'est le plus petit nombre de symboles q-aires nécessaires en moyenne pour décrire un symbole de $\mathscr{A}$ produit par la source. Shannon montre alors que l'entropie $H_q(X)$ est :

$$H_q(X) = -\sum_{x\in\mathscr{A}} \mathbb{P}\left(X=x\right)\log_q\left(\mathbb{P}\left(X=x\right)\right) \tag{1.1}$$

Soient deux sources discrètes dépendantes l'une de l'autre produisant respectivement des symboles dans $\mathscr{A}$ et $\mathscr{B}$; par exemple, l'entrée et la sortie d'un canal discret. Soit X une variable aléatoire à valeurs dans $\mathscr{A}$ modélisant la première source. Et soit Y une variable aléatoire à valeurs dans $\mathscr{B}$ modélisant la seconde source. L'entropie conditionnelle $H_q(Y|X)$ est la moyenne des entropies de $Y|\{X=x\}$ pour $x\in\mathscr{A}$:

$$H_q(Y|X) := \sum_{x=\mathscr{A}} \mathbb{P}\left(X=x\right)\cdot H_q\big(Y|\{X=x\}\big) \tag{1.2}$$

où pour tout $x\in\mathscr{A}$:

$$H_q\big(Y|\{X=x\}\big) := -\sum_{y\in\mathscr{B}} \mathbb{P}\left(Y=y|X=x\right)\log_q\left(\mathbb{P}\left(Y=y|X=x\right)\right) \tag{1.3}$$

1.3.2 Les canaux discrets sans mémoire

Nous nous intéressons essentiellement à des canaux discrets sans mémoire que nous définissons ainsi :

Définition 1.3.1 (canal discret sans mémoire)**.** *Soient $\mathscr{A}:=\{a_1,\cdots,a_A\}$ et $\mathscr{B}:=\{b_1,\cdots,b_B\}$ deux alphabets et soit une matrice de transition $\mathbf{\Pi}$ de taille $A\times B$, à valeurs dans $[0,1]$ et telle que pour tout $i\in[\![1,A]\!]$, $\sum_{j=1}^{B}\mathbf{\Pi}_{i,j}=1$.*

Le canal discret sans mémoire *CDSM($\mathscr{A},\mathscr{B},\mathbf{\Pi}$) prend en entrée des symboles de $\mathscr{A}$ et sort des symboles de $\mathscr{B}$. Pour tout $(i,j)\in[\![1,A]\!]\times[\![1,B]\!]$, le coefficient $\mathbf{\Pi}_{i,j}$ représente la probabilité que le symbole reçu soit b_j sachant que le symbole émis est a_i.*

Le canal est dit *sans mémoire* car chaque symbole reçu ne dépend que du symbole émis correspondant et non des symboles reçus ou émis par le passé. En d'autres termes, la matrice de transition est constante d'une transmission à l'autre sur ce canal.

Dans ses travaux, Shannon définit la notion de capacité d'un canal de transmission et donne des outils pour la calculer. La capacité d'un canal discret peut s'interpréter comme une mesure de la quantité d'information – en nombre de symboles q-aires par unité de temps ($\text{symb}_q/\Delta t$) – qui peut être transmise via ce canal. Plus formellement, la capacité d'un canal discret sans mémoire est donnée par la définition suivante :

Définition 1.3.2 (capacité d'un canal discret sans mémoire)**.** *Soient $\mathscr{A}$, $\mathscr{B}$ et $\mathbf{\Pi}$ définissant un canal discret sans mémoire CDSM($\mathscr{A}, \mathscr{B}, \mathbf{\Pi}$). La capacité de ce dernier est :*

$$C_{CDSM}(\mathscr{A}, \mathscr{B}, \mathbf{\Pi}) := \max_{\mathcal{D}} \Big(H_q(Y) - H_q(Y|X) \Big) \quad symb_q/\Delta t \tag{1.4}$$

$$= \log_2(q) \cdot \max_{\mathcal{D}} \Big(H_q(Y) - H_q(Y|X) \Big) \quad bits/\Delta t \tag{1.5}$$

$$= \max_{\mathcal{D}} \Big(H_2(Y) - H_2(Y|X) \Big) \quad bits/\Delta t \tag{1.6}$$

où :

- *$\mathcal{D}$ est l'ensemble des distributions de probabilité d'émission possibles :*

$$\mathcal{D} := \left\{ P : \mathscr{A} \to [0,1] \ : \ \sum_{x=\mathscr{A}} P(x) = 1 \right\} \tag{1.7}$$

- *X est la variable aléatoire à valeurs dans $\mathscr{A}$ modélisant l'entrée du canal. La distribution de X est définie par $\mathbb{P}(X = x) := P(x)$ où $P \in \mathcal{D}$.*
- *Y est la variable aléatoire à valeurs dans $\mathscr{B}$ modélisant la sortie du canal.*

Remarque 1.3.1. En pratique, le bruit est souvent continu puisque les canaux physiques de transmission sont souvent analogiques. Des modulateurs et démodulateurs permettent de transformer un signal discret en un signal continu et inversement ; nous replaçant ainsi dans le contexte d'un canal discret. Cependant, certains codes permettent de traiter directement l'information continue. L'intérêt de tels codes est qu'ils permettent d'éviter la perte d'information provoquée par la démodulation. Il est donc parfois judicieux de considérer des canaux continus comme par exemple les canaux binaires à bruit blanc gaussien additif ou plus généralement les canaux gaussiens.

Théorème 1.3.3 (1er théorème de Shannon)**.** *Pour toute source discrète sans mémoire et pour tout réel $\varepsilon > 0$, il existe un codage pour lequel la probabilité d'erreur $\mathbb{P}(erreur) < \varepsilon$.*

Théorème 1.3.4 (2ème théorème de Shannon)**.** *Soit un canal discret sans mémoire CDSM($\mathscr{A}, \mathscr{B}, \mathbf{\Pi}$) de capacité C.*

i. Pour tout $R > C$, il existe une suite $\mathscr{C}_n$ de codes de longueur n et de rendement R_n pour lesquels il existe un algorithme de décodage ayant une probabilité d'erreur $\mathbb{P}_n(erreur)$ telle que :

$$\lim_{n\to\infty} R_n = R \qquad et \qquad \lim_{n\to\infty} \mathbb{P}_n(erreur) = 0$$

ii. Pour tout code $\mathscr{C}$ de rendement $R > C$ et pour tout décodeur de $\mathscr{C}$, on a

$$\mathbb{P}(erreur) > \text{cst}(R, C) > 0$$

où $\text{cst}(R, C)$ est une constante qui ne dépend que de R et C.

Démonstration.

La borne du point i. du 2ème théorème de Shannon est atteinte avec des codes aléatoires.

Pour le point ii., on peut montrer que si le rendement est plus grand que la capacité du canal, alors il existe deux mots de codes dont la distance est inférieure à la distance qu'il peut y avoir entre un mot émis et sa version erronée reçue.

Pour compléter les démonstrations des points i. et ii., il faut en dire un peu plus sur la distance minimale d'un code (*cf.* chapitre suivant). □

1.3.3 Les canaux symétriques sur $\mathbb{F}_q$ et la fonction entropie q-aire

Nous allons supposer ici que les alphabets d'entrée et de sortie sont tous les deux $\mathbb{F}_q$ (le corps fini à q éléments) et que toutes les erreurs sont équiprobables ; c'est-à-dire que pour tout $(i,j) \in [\![0, q-1]\!]^2$ tel que $i \neq j$, tous les $\mathbf{\Pi}_{i,j} := \mathbb{P}(Y = j | X = i)$ sont égaux. Notons alors p la probabilité d'erreur. Lorsqu'un symbole $x \in \mathbb{F}_q$ est émis, la probabilité que le symbole reçu y soit égal à x est $1-p$ et pour tout $z \in \mathbb{F}_q \backslash \{x\}$ la probabilité que y soit égal à z est $\frac{p}{q-1}$. Le canal ainsi obtenu est appelé canal q-aire symétrique de probabilité d'erreur p et est noté $\mathrm{SC}_q(p)$. Sa matrice de transition est la matrice de taille $q \times q$ suivante :

$$\mathbf{\Pi} = \begin{bmatrix} 1-p & \frac{p}{q-1} & \cdots & \cdots & \frac{p}{q-1} \\ \frac{p}{q-1} & 1-p & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \frac{p}{q-1} \\ \frac{p}{q-1} & \cdots & \cdots & \frac{p}{q-1} & 1-p \end{bmatrix} \tag{1.8}$$

Soit X et Y les variables aléatoires modélisant respectivement l'entrée et la sortie du canal $\mathrm{SC}_q(p)$. Lorsque la distribution de X est uniforme, $H(Y|X)$ est une fonction de p que l'on appellera fonction entropie q-aire de Shannon :

Définition 1.3.5 (fonction entropie q-aire de Shannon)**.** *La fonction entropie q-aire de Shannon est :*

$$\begin{array}{rccc} h_q : & [0,1] & \longrightarrow & [0,1] \\ & p & \longmapsto & p\log_q(q-1) - p\log_q(p) - (1-p)\log_q(1-p) \end{array} \tag{1.9}$$

Par la suite, cette fonction nous sera extrêmement utile ; notamment pour donner une approximation asymptotique des coefficients binomiaux :

Théorème 1.3.6.

$$\binom{n}{\alpha n}(q-1)^{\alpha n} \underset{n\to\infty}{\sim} \frac{q^{nh_q(\alpha)}}{\sqrt{2\pi n\alpha(1-\alpha)}} \tag{1.10}$$

Démonstration.

La preuve utilise essentiellement la *formule de Stirling* qui donne une approximation asymptotique de la fonction factorielle :

$$n! \underset{n\to\infty}{\sim} \sqrt{2\pi n}\, n^n\, e^{-n} \tag{1.11}$$

□

La capacité du canal $\mathrm{SC}_q(p)$ est atteinte pour une distribution de probabilité d'émission uniforme. On peut alors exprimer cette capacité à l'aide de la fonction entropie q-aire de Shannon :

Théorème 1.3.7. *La capacité du canal $SC_q(p)$ est :*

$$C_{SC_q(p)} = 1 - h_q(p) \tag{1.12}$$

Démonstration.

Soit $P \in \mathcal{D}$ la distribution de X et soit Q la distribution de Y ; c'est-à-dire

$$P(i) := \mathbb{P}(X = i) \quad \text{et} \quad Q(j) := \mathbb{P}(Y = j)$$

On a alors

$$\begin{aligned} H_q(Y|X) &:= -\sum_{i=0}^{q-1} P(i) \sum_{j=0}^{q-1} \mathbf{\Pi}_{i,j} \log_q(\mathbf{\Pi}_{i,j}) \\ &= -\sum_{i=0}^{q-1} P(i) \left((1-p)\log_q(1-p) + (q-1)\frac{p}{q-1}\log_q\left(\frac{p}{q-1}\right)\right) \\ &= h_q(p) \cdot \sum_{i=0}^{q-1} P(i) \\ &= h_q(p) \end{aligned}$$

Donc

$$\begin{aligned} C_{\mathrm{SC}_q(p)} &:= \max_{P\in\mathcal{D}} (H_q(Y) - H_q(Y|X)) \\ &= \max_{P\in\mathcal{D}} (H_q(Y) - h_q(p)) \\ &= \max_{P\in\mathcal{D}} (H_q(Y)) - h_q(p) \end{aligned}$$

D'autre part,

$$H_q(Y) = -\sum_{j=0}^{q-1} Q(j)\log_q(Q(j))$$

qui atteint sa valeur maximale égale à 1 pour Q uniforme ; c'est-à-dire pour $Q(j) = \frac{1}{q}$ pour tout $j \in [\![0, q-1]\!]$. Il ne reste donc plus qu'à montrer qu'il existe une distribution d'entrée P pour X telle que la distribution de sortie Q de Y soit uniforme. Or pour tout $j \in [\![0, q-1]\!]$, on a

$$\begin{aligned} Q(j) := \mathbb{P}(Y=j) &= \sum_{i=0}^{q-1} \mathbb{P}(X=i)\mathbb{P}(Y=j|X=i) \\ &= \sum_{i=0}^{q-1} P(i)\mathbf{\Pi}_{i,j} \\ &= P(j)(1-p) + \sum_{\substack{i=0 \\ i\neq j}}^{q-1} P(i)\frac{p}{q-1} \\ &= P(j)(1-p) + (1-P(j))\tfrac{p}{q-1} \end{aligned}$$

Et donc,

$$\begin{aligned} Q(j) = \tfrac{1}{q} &\iff P(j)(1-p) + (1-P(j))\tfrac{p}{q-1} = \tfrac{1}{q} \\ &\iff P(j)\left(1 - p - \tfrac{p}{q-1}\right) + \tfrac{p}{q-1} = \tfrac{1}{q} \\ &\iff P(j)\left(\tfrac{q-1-pq}{q-1}\right) = \tfrac{q-1-pq}{q(q-1)} \\ &\iff P(j) = \tfrac{1}{q} \end{aligned}$$

Et donc dans le cas du canal $\mathrm{SC}_q(p)$, la capacité donnée par le théorème est atteinte pour une distribution de probabilité d'émission P uniforme.

□

À titre d'exemple, la figure 1.6 donne une représentation schématisée du canal $\mathrm{SC}_3(p)$ lorsque la distribution de probabilité d'émission est uniforme :

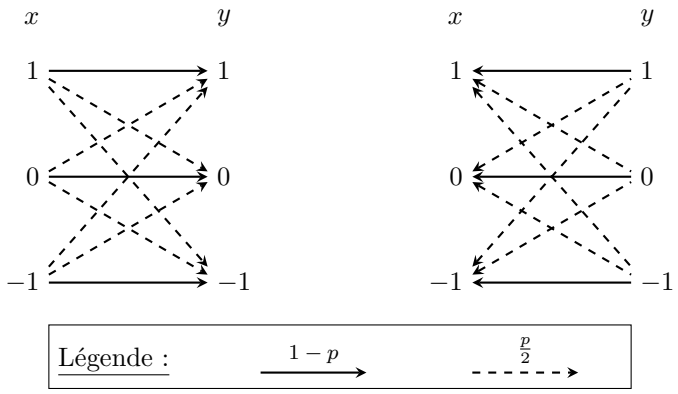


Figure 1.6 – Canal symétrique de probabilité d'erreur p sur un alphabet à 3 symboles.

Remarque 1.3.2. Les canaux symétriques sans mémoire sont une classe de canaux bien plus large que les canaux que nous venons de décrire. Ce sont plus généralement les canaux dont la capacité est atteinte pour une distribution de probabilité d'émission uniforme.

1.3.4 Les canaux binaires symétriques

Le canal binaire symétrique est probablement le modèle de canal le plus couramment utilisé. Il est simplement une instance particulière des canaux symétriques sur $\mathbb{F}_q$ où $q = 2$. Les symboles émis et reçus sont donc des valeurs binaires (que l'on appelle aussi *bits*). Notons BSC(p) le canal binaire symétrique de probabilité d'erreur p. Dans ce canal, lorsqu'un bit x est émis, la probabilité que le bit reçu y soit égal à x est $1 - p$ et donc la probabilité que y soit la valeur complémentaire de x est p. Nous avons alors la matrice de transition suivante :

$$\mathbf{\Pi} = \begin{bmatrix} 1-p & p \\ p & 1-p \end{bmatrix} \tag{1.13}$$

Le corolaire suivant se déduit directement du théorème 1.3.7 :

Corollaire 1.3.8. *La capacité du canal BSC(p) est :*

$$C_{BSC(p)} = 1 - h_2(p) \tag{1.14}$$

où $h_2(p) := -p\log_2(p) - (1-p)\log_2(1-p)$ *définit la fonction entropie binaire de Shannon.*

La figure 1.7 donne une représentation schématisée du canal BSC(p) lorsque la distribution de probabilité d'émission est uniforme :

Figure 1.7 – Canal binaire symétrique de probabilité d'erreur p.

1.3.5 Les canaux à effacement

Dans un canal à effacement, si l'alphabet d'entrée est $\mathscr{A}$, alors l'alphabet de sortie est $\mathscr{A} \cup \{e\}$ où e représente un bit effacé (par abus de notation, on pose $e := q$). Les exemples les plus simples de canaux à effacement sont ceux où les symboles émis sont soit reçus correctement, soit effacés. On note $\mathrm{EC}_q(p)$ le canal à effacement de probabilité d'effacement p sur le corps $\mathbb{F}_q$ dont la matrice de transition est :

$$\mathbf{\Pi} = \begin{bmatrix} 1-p & 0 & \cdots & 0 & p \\ 0 & \ddots & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & 1-p & p \end{bmatrix} \tag{1.15}$$

La capacité du canal $\mathrm{EC}_q(p)$ est atteinte pour une distribution de probabilité d'émission uniforme. Sa valeur est alors donnée par le théorème suivant :

Théorème 1.3.9. *La capacité du canal $EC_q(p)$ est :*

$$C_{EC_q(p)} = 1 - p \tag{1.16}$$

Démonstration.

Soit $P \in \mathcal{D}$ la distribution de X et soit Q la distribution de Y ; c'est-à-dire

$$P(i) := \mathbb{P}(X = i) \quad \text{et} \quad Q(j) := \mathbb{P}(Y = j)$$

Pour tout $j \in [\![0, q-1]\!]$, on a

$$\begin{aligned} Q(j) &:= \sum_{i=0}^{q-1} P(i)\mathbf{\Pi}_{i,j} \\ &= P(j)(1-p) \end{aligned}$$

Et pour $j = e$, on a

$$\begin{aligned} Q(e) &:= \sum_{i=0}^{q-1} P(i)\mathbf{\Pi}_{i,e} \\ &= p \end{aligned}$$

On a donc

$$\begin{aligned} H_q(Y) &= -p\log_q(p) - \sum_{j=0}^{q-1} P(j)(1-p)\log_q\left(P(j)(1-p)\right) \\ &= -p\log_q(p) - (1-p)\log_q(1-p) + (1-p)H_q(X) \end{aligned}$$

D'autre part,

$$\begin{aligned} H_q(Y|X) &:= -\sum_{i=0}^{q-1} P(i)\left(\mathbf{\Pi}_{i,e}\log_q\left(\mathbf{\Pi}_{i,e}\right) + \sum_{j=0}^{q-1}\mathbf{\Pi}_{i,j}\log_q\left(\mathbf{\Pi}_{i,j}\right)\right) \\ &= -\sum_{i=0}^{q-1} P(i)\left(p\log_q(p) + (1-p)\log_q(1-p)\right) \\ &= -p\log_q(p) - (1-p)\log_q(1-p) \end{aligned}$$

Et donc

$$\begin{aligned} C_{\mathrm{EC}_q(p)} &:= \max_{P\in\mathcal{D}}\left(H_q(Y) - H_q(Y|X)\right) \\ &= \max_{P\in\mathcal{D}}(1-p)H_q(X) \end{aligned}$$

Or $H_q(X)$ atteint sa valeur maximale égale à 1 pour P uniforme. D'où le théorème. □

Remarque 1.3.3.

$$\forall p < 1 - \frac{1}{q},\ \ C_{\mathrm{EC}_q(p)} > C_{\mathrm{SC}_q(p)}$$

La figure 1.8 donne une représentation schématisée du canal binaire à effacement de probabilité d'effacement p lorsque la distribution de probabilité d'émission est uniforme. Ce canal est noté $\mathrm{BEC}(p)$.

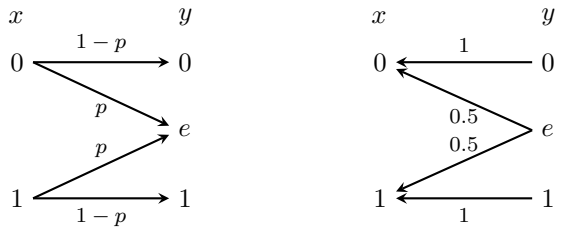


Figure 1.8 – Canal binaire à effacement de probabilité d'effacement p.

1.3.6 Le modèle d'erreur

Supposons que n symboles sont transmis via un canal. La forme de l'erreur que nous devons corriger est directement liée au canal de transmission. Par exemple, dans le canal à effacement $\mathrm{EC}_q(p)$, l'erreur se décrit par l'ensemble des positions effacées. Dans le canal symétrique $\mathrm{SC}_q(p)$, l'erreur peut être décrite par un vecteur $\mathbf{e} \in \mathbb{F}_q^n$ qui est la différence $\mathbf{y} - \mathbf{x}$ entre le vecteur émis $\mathbf{x} \in \mathbb{F}_q^n$ et le vecteur reçu $\mathbf{y} \in \mathbb{F}_q^n$. Chaque position de $\mathbf{e}$ a alors une probabilité p d'être non nulle et une probabilité $1 - p$ d'être nulle. Autrement dit, le vecteur d'erreur $\mathbf{e}$ est un vecteur de $\mathbb{F}_q^n$ dont le poids de Hamming suit une loi binomiale de paramètre (n, p).

Un autre modèle d'erreur est celui où le mot reçu diffère sur au plus w positions du mot original où w est un entier fixé. Sur $\mathbb{F}_q^n$ muni de la métrique de Hamming, l'erreur est alors un vecteur $\mathbf{e}$ de poids de Hamming $|\mathbf{e}| \leqslant w$ où $|\mathbf{e}|$ est le nombre de positions non nulles de $\mathbf{e}$. Remarquons que pour tout $w > pn$, le canal symétrique $\mathrm{SC}_q(p)$ produit avec une bonne probabilité un vecteur d'erreur de poids inférieur à w. Pour le montrer, il suffit d'appliquer l'inégalité de Bienaymé-Tchevychev ou celle de Hoeffding.

Plus précisément, l'inégalité de Hoeffding permet de montrer la proposition suivante :

Proposition 1.3.10. *Soit $\mathbf{e} \in \mathbb{F}_q^n$ un vecteur d'erreur produit par un canal symétrique $SC_q(p)$ et soit un entier $w \geqslant pn$. La probabilité que $|\mathbf{e}|$ soit inférieur à w est :*

$$\mathbb{P}\left(|\mathbf{e}| \leqslant w\right) \geqslant 1 - e^{-2n\left(\frac{w}{n} - p\right)^2} \tag{1.17}$$

Démonstration.

Remarquons tout d'abord que :

$$\mathbb{P}\left(|\mathbf{e}| > w\right) = \frac{1}{2}\mathbb{P}\left(\Big||\mathbf{e}| - pn\Big| > w - pn\right) \tag{1.18}$$

En outre, le poids de Hamming $|\mathbf{e}|$ du vecteur d'erreur $\mathbf{e}$ suit une loi binomiale de paramètre (n, p). Donc d'après l'inégalité de Hoeffding, nous avons :

$$\mathbb{P}\left(\Big||\mathbf{e}| - pn\Big| > w - pn\right) \leqslant 2e^{-2n\left(\frac{w}{n} - p\right)^2} \tag{1.19}$$

□

Les formes d'erreur décrites jusqu'ici modélisent des situations qui peuvent se produire en pratique. Nous allons toutefois nous intéresser également à un autre modèle d'erreur que nous appellerons le *modèle poids constant*. Sur le corps $\mathbb{F}_q$ muni de la distance de Hamming, ce modèle d'erreur consiste à supposer que le mot à corriger diffère sur **exactement** w positions du mot émis où w est un entier fixé. Autrement dit, on recherche ici un vecteur d'erreur de poids de Hamming exactement w. Noter qu'asymptotiquement, puisque tous les éléments d'une boule se concentrent sur la sphère, le modèle poids constant et le modèle où le poids de l'erreur est borné sont équivalents.

Attention, le modèle poids constant (ainsi que le modèle poids borné) n'est pas induit par un canal discret sans sans mémoire. En effet, pour chaque symbole, le canal doit se souvenir du nombre de symbole erronés qu'il a envoyé dans le passé avant de décider d'ajouter une erreur ou non. Pour tout $i \in [\![1, n]\!]$,

$$\mathbb{P}(X_i = 1 | \sum_{j=1}^{i-1} X_j = w_i) = \frac{w - w_i}{n - i}$$

où X_i est la variable de Bernoulli indiquant s'il y a une erreur sur la position i. Le modèle poids constant est un modèle d'erreur très peu réaliste dans l'univers des télécommunications mais très pertinent pour la cryptographie basée sur les codes que nous introduirons plus tard.

Chapitre 2

Les codes en bloc linéaires

2.1 Définition

Dans sa définition la plus générale, un code en bloc $\mathcal{C}$ de longueur n sur un alphabet $\mathscr{A}$ est tout simplement un sous-ensemble de $\mathscr{A}^n$ contenant les mots autorisés. Corriger un mot reçu par maximum de vraisemblance consiste à trouver le mot de $\mathcal{C}$ le plus proche du mot reçu ; c'est-à-dire le mot de $\mathcal{C}$ qui contient le plus de symboles identiques au mot reçu. Le manque de structure dans $\mathcal{C}$ nous force à stocker l'ensemble des mots du code pour être en mesure de corriger et le seul algorithme de correction envisageable est une recherche exhaustive du mot le plus proche. Heureusement pour le monde des télécommunications, des solutions plus performantes existent. Parmi toutes ces solutions, il y a une structure que l'on retrouve presque toujours : celle d'espace vectoriel.

Définition 2.1.1. *Un code linéaire de longueur n et de dimension k sur le corps fini de taille q est un sous-espace vectoriel de dimension k de $\mathbb{F}_q^n$. On le note généralement $[n,k]_q$ (ou plus simplement $[n,k]$ lorsqu'il n'y a pas d'ambiguïté sur le corps). La proportion d'information contenue dans un mot d'un $[n,k]_q$ code est appelée rendement du code et vaut le rapport $\frac{k}{n}$.*

Le deuxième théorème de Shannon [Sha48] stipule que pour n'importe quel canal discret sans mémoire de capacité C (exprimée en *nombre de symboles q-aire par unité de temps*) et pour $R < C$, il existe une suite de codes correcteurs $\mathcal{C}_n$ de longueur n et de rendement R_n telle que $\lim_{n\to\infty} R_n = R$ et telle que la probabilité d'erreur p_n pour un décodage par maximum de vraisemblance de $\mathcal{C}_n$ tend vers 0 lorsque n tend vers l'infini.

Distance de Hamming. Tous les codes correcteurs d'erreurs utilisés en pratique sont des codes linéaires sur un corps fini $\mathbb{F}_q$. Nous devons munir $\mathbb{F}_q$ d'une métrique. Nous avons précédemment défini la distance entre deux mots comme le nombre de positions sur lesquelles ils diffèrent ; sur $\mathbb{F}_q$, il s'agit de la distance de Hamming.

Notation 2.1.2. *Soient $\mathbf{x}, \mathbf{y} \in \mathbb{F}_q^n$. La distance de Hamming entre $\mathbf{x}$ et $\mathbf{y}$ est le nombre de positions sur lesquelles ces deux vecteurs diffèrent. Elle est notée $\Delta(\mathbf{x}, \mathbf{y})$:*

$$\Delta(\mathbf{x}, \mathbf{y}) := \#\{i \in [\![1, n]\!] : x_i \neq y_i\} \tag{2.1}$$

avec $\mathbf{x} := (x_1, \cdots, x_n)$ et $\mathbf{y} := (y_1, \cdots, y_n)$. Le poids de Hamming $|\mathbf{x}|$ est le nombre de positions non nulles de $\mathbf{x}$:

$$|\mathbf{x}| := \#\{i \in [\![1, n]\!] : x_i \neq 0\} = \Delta(\mathbf{x}, \mathbf{0}) \tag{2.2}$$

2.2 Matrices génératrices & Matrice de parité

Un code linéaire $\mathcal{C}[n,k]_q$ peut être représenté par une matrice génératrice :

Définition 2.2.1 (matrice génératrice)**.** *Soit $\mathcal{C}$ un code en bloc linéaire de longueur n et de dimension k sur le corps fini $\mathbb{F}_q$. Une matrice génératrice de $\mathcal{C}$ est une matrice $\mathbf{G} \in \mathbb{F}_q^{k\times n}$ dont les lignes forment une base de $\mathcal{C}$.*

Le code $\mathcal{C}$ peut aussi être défini comme le noyau d'une matrice de parité :

Définition 2.2.2 (matrice de parité)**.** *Soit $\mathcal{C}$ un code en bloc linéaire de longueur n et de dimension k sur le corps fini $\mathbb{F}_q$. Une matrice de parité de $\mathcal{C}$ est une matrice $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$ telle que $\mathcal{C}$ est le noyau de $\mathbf{H}$; c'est-à-dire que $\mathcal{C}$ est l'ensemble des mots $\mathbf{c} \in \mathbb{F}_q^n$ tels que $\mathbf{H}\mathbf{c}^\top = \mathbf{0}$.*

Remarque 2.2.1. On utilise parfois une définition plus générale de matrice génératrice et matrice de parité. En effet, soient $\mathbf{G} \in \mathbb{F}_q^{\ell\times n}$ et $\mathbf{H} \in \mathbb{F}_q^{(n-\ell)\times n}$ avec $\ell \geqslant k$. On dira que $\mathbf{G}$ est une matrice génératrice de $\mathcal{C}$ si les lignes de $\mathbf{G}$ forment une famille génératrice de $\mathcal{C}$. De manière duale, on dira que $\mathbf{H}$ est une matrice de parité de $\mathcal{C}$ si $\mathcal{C}$ est le noyau à droite de $\mathbf{H}$.

Dualité. L'espace vectoriel orthogonal de $\mathcal{C}$ généré par la matrice $\mathbf{H}$ est appelé code dual de $\mathcal{C}$ et est noté $\mathcal{C}^\perp$.

Définition 2.2.3. *Le code dual de $\mathcal{C}$ est*

$$\mathcal{C}^\perp := \left\{\mathbf{x} \in \mathbb{F}_q^n \ : \ \forall \mathbf{c} \in \mathcal{C}, \langle \mathbf{c}, \mathbf{x}\rangle = \mathbf{0}\right\} \tag{2.3}$$

Proposition 2.2.4.

$$\dim(\mathcal{C}) + \dim(\mathcal{C}^\perp) = n \tag{2.4}$$
$$\left(\mathcal{C}^\perp\right)^\perp = \mathcal{C} \tag{2.5}$$
$$\{\mathbf{0}\}^\perp = \mathbb{F}_q^n \tag{2.6}$$
$$\left(\mathbb{F}_q^n\right)^\perp = \{\mathbf{0}\} \tag{2.7}$$
$$\mathcal{C}' \subseteq \mathcal{C} \implies \mathcal{C}^\perp \subseteq \mathcal{C}'^\perp \tag{2.8}$$
$$(\mathcal{C} + \mathcal{C}')^\perp = \mathcal{C}^\perp \cap \mathcal{C}'^\perp \tag{2.9}$$
$$(\mathcal{C} \cap \mathcal{C}')^\perp = \mathcal{C}^\perp + \mathcal{C}'^\perp \tag{2.10}$$

Proposition 2.2.5. *Soient $\mathbf{G}$ et $\mathbf{H}$, respectivement une matrice génératrice et une matrice de parité de $\mathcal{C}$. Alors $\mathbf{G}$ est une matrice de parité de $\mathcal{C}^\perp$ et $\mathbf{H}$ en est une matrice génératrice.*

Démonstration des propositions 2.2.4 et 2.2.5.

Ce sont des conséquences directs de résultats bien connus sur les espaces vectoriel orthogonaux. □

Les éléments du code dual $\mathcal{C}^\perp$ (en particulier les lignes de $\mathbf{H}$) sont aussi appelés *équations de parité* du code $\mathcal{C}$.

Matrices systématiques. Par définition, une matrice génératrice du code $\mathcal{C}$ est de rang k. Une simple élimination gaussienne permet donc de transformer une matrice génératrice quelconque de $\mathcal{C}$ en l'unique matrice génératrice contenant l'identité sur ses k premières colonnes. Une telle représentation du code $\mathcal{C}$ est dite *systématique*.

Une matrice génératrice systématique $\mathbf{G} := [\mathbf{Id}_k|\mathbf{A}]$ permet de coder facilement les messages puisqu'un mot de code $\mathbf{x} \in \mathbb{F}_q^n$ associé à un message $\mathbf{m} \in \mathbb{F}_q^k$ contiendra le message lui-même sur ses k premières positions :

$$\mathbf{x} := \mathbf{mG} = (\mathbf{m}, \mathbf{mA}) \tag{2.11}$$

Un tel codage est lui aussi dit systématique.

Démonstration de l'unicité de la matrice systématique.

Soient $\mathbf{G}$ et $\mathbf{G}'$, deux matrices génératrice de systématiques de $\mathcal{C}$. $\mathbf{G}$ et $\mathbf{G}'$ génèrent le même espace vectoriel donc il existe $\mathbf{P} \in GL(k, \mathbb{F}_q)$ tel que

$$\begin{array}{rcl} \mathbf{G} = \mathbf{PG}' & \iff & [\mathbf{Id}_k|\mathbf{A}] = \mathbf{P}\left[\mathbf{Id}_k|\mathbf{A}'\right] \\ & \iff & \mathbf{Id}_k = \mathbf{PId}_k \quad \text{et} \quad \mathbf{A} = \mathbf{PA}' \\ & \iff & \mathbf{P} = \mathbf{Id}_k \quad \text{et} \quad \mathbf{A} = \mathbf{A}' \\ & \iff & \mathbf{G} = \mathbf{G}' \end{array}$$

□

Donner une matrice génératrice sous sa forme systématique permet de construire facilement une matrice de parité du code grâce à la propriété suivante :

Proposition 2.2.6. *Soit $\mathcal{C}$ un code en bloc linéaire de longueur n et de dimension k sur le corps fini $\mathbb{F}_q$. Soit $\mathbf{G} := [\mathbf{Id}_k|\mathbf{A}] \in \mathbb{F}_q^{k\times n}$ une matrice génératrice systématique de $\mathcal{C}$. La matrice $\mathbf{H} := \left[-\mathbf{A}^\perp|\mathbf{Id}_{n-k}\right] \in \mathbb{F}_q^{(n-k)\times n}$ est une matrice de parité du code $\mathcal{C}$.*

2.3 Distance minimale d'un code

Définition 2.3.1 (distance minimale)**.** *Soit $\mathcal{C}$ un code linéaire $[n,k]_q$. La distance minimale de $\mathcal{C}$, notée $d_\mathcal{C}$, est la distance de Hamming minimale entre deux mots de code distincts*

Comme $\mathcal{C}$ est linéaire, $d_\mathcal{C}$ est aussi la norme minimale des mots non nuls de $\mathcal{C}$. Soit $\mathbf{x}$ un mot de code émis et soit $\mathbf{y}$ le mot reçu. Notons $\mathbf{e}$ le vecteur d'erreur ; c'est-à-dire $\mathbf{y} = \mathbf{x} + \mathbf{e}$.

Lemme 2.3.2. *Soit $\mathcal{C}$ un code et $d_\mathcal{C}$ sa distance minimale. Pour tout $\mathbf{c} \neq \mathbf{c}' \in \mathcal{C}$, on a*

$$\mathsf{Ball}\left(\mathbf{c}, \left\lfloor \frac{d_\mathcal{C} - 1}{2} \right\rfloor\right) \cap \mathsf{Ball}\left(\mathbf{c}', \left\lfloor \frac{d_\mathcal{C} - 1}{2} \right\rfloor\right) = \varnothing \tag{2.12}$$

où $\mathsf{Ball}(\mathbf{c}, r) := \left\{\mathbf{x} \in \mathbb{F}_q^n \ : \ \Delta(\mathbf{x}, \mathbf{c}) \leqslant r\right\}$ est la boule sur $\mathbb{F}_q^n$ de rayon r et de centre $\mathbf{c}$.

Démonstration.

Supposons qu'il existe $\mathbf{x} \in \mathsf{Ball}\left(\mathbf{c}, \left\lfloor \frac{d_\mathcal{C}-1}{2} \right\rfloor\right) \cap \mathsf{Ball}\left(\mathbf{c}', \left\lfloor \frac{d_\mathcal{C}-1}{2} \right\rfloor\right)$. Alors

$$\begin{array}{rcll} \Delta(\mathbf{c}, \mathbf{c}') & \leqslant & \Delta(\mathbf{c}, \mathbf{x}) + \Delta(\mathbf{c}', \mathbf{x}) & \text{(inégalité triangulaire)} \\ & \leqslant & \left\lfloor \frac{d_\mathcal{C}-1}{2} \right\rfloor + \left\lfloor \frac{d_\mathcal{C}-1}{2} \right\rfloor & \\ & \leqslant & d_\mathcal{C} - 1 & \end{array}$$

Ce qui est impossible car $d_\mathcal{C}$ est la distance minimale du code et donc on a nécessairement $\Delta(\mathbf{c}, \mathbf{c}') \geqslant d_\mathcal{C}$. □

D'après le lemme précédent, si $|\mathbf{e}| < \frac{d_\mathcal{C}}{2}$, alors un décodage par maximum de vraisemblance permet effectivement de retrouver les données émises à partir des données reçues. C'est pourquoi l'émetteur et le récepteur doivent se mettre d'accord sur un code correcteur qui soit adapté au canal de transmission.

Lien entre matrice de parité et distance minimale. On peut retrouver la distance minimale en étudiant les relations linéaires entre les colonnes de la matrice de parité.

Lemme 2.3.3. *Soit $\mathcal{C}$ un $[n,k]_q$ code linéaire. Soit $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$ une matrice de parité. d est la distance minimale de $\mathcal{C}$* ***si et seulement si*** *i. et ii. sont vrais.*

i. $\forall I \subseteq [\![1,n]\!]$ *tel que* $\sharp I = d-1$, $(\mathbf{H}_{*,i})_{i\in I}$ *forme une famille libre de* $\mathbb{F}_q^{n-k}$

ii. $\exists I \subseteq [\![1,n]\!]$ *tel que* $\sharp I = d$ *et* $(\mathbf{H}_{*,i})_{i\in I}$ *sont linéairement liés*[1].

Démonstration.

Soit $\mathbf{c} \in \mathcal{C}$. On note $w = |\mathbf{c}|$. Soit $I = \mathrm{supp}\,(\mathbf{c}) := \{i \in [\![1,n]\!] \ : \ c_i \neq 0\}$ On a $\sharp I = w$/
De plus, $\mathbf{H}\mathbf{c}^\top = \mathbf{0} \Leftrightarrow \sum_{i\in I} c_i \mathbf{H}_{*,i} = \mathbf{0}$.

D'un côté,

$$\begin{array}{rcl} d = d_{\mathcal{C}} & \Longrightarrow & \nexists \mathbf{c} \in \mathcal{C} \ \text{ tel que } \ |\mathbf{c}| \leqslant d-1 \\ & & \text{et } \exists \mathbf{c} \in \mathcal{C} \ \text{ tel que } \ |\mathbf{c}| = d \\ & \Longrightarrow & \text{i. et ii.} \end{array}$$

Réciproquement,

$$\begin{array}{rcl} \text{i.} & \Longrightarrow & \nexists \mathbf{c} \in \mathcal{C} \ \text{ tel que } \ |\mathbf{c}| \leqslant d-1 \\ \text{ii.} & \Longrightarrow & \exists \mathbf{c} \in \mathcal{C} \ \text{ tel que } \ |\mathbf{c}| = d \end{array}$$

donc i. et ii. $\Longrightarrow d_{\mathcal{C}} = d$. □

Corollaire 2.3.4. *Soit $\mathcal{C}$ un $[n,k]_q$ code linéaire et $d_{\mathcal{C}}$ sa distance minimale. Soit $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$ une matrice de parité de $\mathcal{C}$.*

i. Si $\mathbf{H}$ n'a pas de colonne nulle, alors $d_{\mathcal{C}} > 1$.

ii. Si les colonnes de $\mathbf{H}$ sont 2 à 2 non colinéaires, alors $d_{\mathcal{C}} > 2$.

2.4 Borne de Singleton, codes MDS

Le théorème suivant donne une borne sur la distance minimale qu'un code en bloc linéaire peut prétendre avoir :

Théorème 2.4.1 (borne de Singleton)**.** *Soit $\mathcal{C}$ un code en bloc linéaire $[n,k]_q$. La distance minimale $d_{\mathcal{C}}$ de $\mathcal{C}$ respecte l'inégalité suivante :*

$$\boxed{d_{\mathcal{C}} \leqslant n-k+1} \tag{2.13}$$

Démonstration du théorème 2.5.1.

Première démonstration : À permutation près, on peut supposer qu'il existe une matrice génératrice systématique du code $\mathcal{C}$. Soit $\mathbf{G}$ une matrice génératrice systématique de $\mathcal{C}$ et soit $\mathbf{c} := (c_1, \cdots, c_n) \in \mathcal{C}$. On note $\mathbf{m} := (c_1, \cdots, c_k)$ les k premières positions de $\mathbf{c}$. Pour tout vecteur $\mathbf{x} \in \mathbb{F}_q^k$ tel que $|\mathbf{x}| = 1$, nous avons d'une part $(\mathbf{m}+\mathbf{x})\mathbf{G} \in \mathcal{C}$ et d'autre part $\Delta\left((\mathbf{m}+\mathbf{x})\mathbf{G}, \mathbf{c}\right) \leqslant n-k+1$.

Deuxième démonstration : Soit

$$\begin{array}{rccc} \phi & : & \mathbb{F}_q^n & \longrightarrow & \mathbb{F}_q^{n-(d_{\mathcal{C}}-1)} \\ & & (x_1, \cdots, x_n) & \longmapsto & (x_1, \cdots, x_{n-(d_{\mathcal{C}}-1)}) \end{array}$$

On remarque tout d'abord que $\phi|_{\mathcal{C}}$ est injective. En effet, pour tout $\mathbf{c} \in \mathcal{C}$ si $\phi(\mathbf{c}) = (c_1, \cdots, c_{n-(d_{\mathcal{C}}-1)}) = \mathbf{0}$ alors $|\mathbf{c}| \leqslant n - n + (d_{\mathcal{C}}-1) = d_{\mathcal{C}} - 1$. Or comme $d_{\mathcal{C}}$ est la distance minimale du code, la seule possibilité pour $\mathbf{c}$ est $\mathbf{c} = \mathbf{0}$. Finalement, puisque $\phi|_{\mathcal{C}}$ est injective, on a :

$$k = \dim(\mathcal{C}) = \dim\left(\phi|_{\mathcal{C}}\right) \leqslant n - d_{\mathcal{C}} + 1$$

Troisième démonstration : Conséquence directe du lemma 2.3.3 □

1. **Notation.** $\mathbf{H}_{*,i}$ dénote la $i^{\text{ème}}$ colonne de $\mathbf{H}$.

Définition 2.4.2 (code MDS)**.** *Un code en bloc linéaire* $[n,k]_q$ *est MDS (*Maximum Distance Séparable*) si et seulement s'il atteint sa borne de Singleton.*

Les codes MDS n'ont aucune redondance inutile ; ce qui se traduit par le théorème suivant :

Théorème 2.4.3. *Soit* $\mathcal{C}$ *un code en bloc linéaire* $[n,k]_q$ *défini par une matrice génératrice* $\mathbf{G}$ *et une matrice de parité* $\mathbf{H}$*. Le code* $\mathcal{C}$ *est MDS si et seulement si l'une des conditions suivantes est respectée :*

i. Tout ensemble de k *colonnes de* $\mathbf{G}$ *forme une matrice inversible (on dit alors que l'ensemble des indices des colonnes sélectionnées est un* ensemble d'information*).*

ii. Tout ensemble de $n-k$ *colonnes de* $\mathbf{H}$ *forme une matrice inversible (on dit alors que l'ensemble des indices des colonnes sélectionnées est un* ensemble d'information*).*

Démonstration.

∗) Tout d'abord que $\mathcal{C}$ est MDS si et seulement si ii.
En effet, si $d_{\mathcal{C}} = n-k+1$, alors d'après le lemme 2.3.3, on a bien ii.
Réciproquement, supposons que ii. est vérifié. Alors d'après le lemme 2.3.3, on a que $d_{\mathcal{C}} \geqslant n-k+1$. Or d'après la borne de Singleton, $d_{\mathcal{C}} \leqslant n-k+1$ et donc on a finalement $d_{\mathcal{C}} = n-k+1$.

∗) On peut ensuite montrer que $\mathcal{C}$ est MDS si et seulement si i.
Pour cela, il suffit de remarquer que $\mathbf{H}$ est une matrice génératrice du $[n,n-k]_q$ code linéaire $\mathcal{C}^\perp$ et que $\mathbf{G}$ en est une matrice de parité. On effectue alors la même démonstration que précédemment mais sur le code $\mathcal{C}^\perp$.

□

Corollaire 2.4.4. $\mathcal{C}$ *est MDS* $\iff$ $\mathcal{C}^\perp$ *est MDS.*

2.5 Borne de Hamming, codes parfaits

Pour une distance t donnée, nous souhaitons connaitre la taille maximale du code pour que tous les mots à distance au plus t du code (c'est-à-dire à distance au plus t d'un des mots du code) soient corrigibles. Pour que l'assertion précédente soit vérifiée, il faut et il suffit que les boules fermées de rayon t centrées sur les mots du code ne s'intersectent pas. Ce qui se traduit par le théorème suivant :

Théorème 2.5.1 (borne de Hamming)**.** *Soit* $\mathcal{C}$ *un code en bloc linéaire* $[n,k]_q$*. Le nombre* t *d'erreurs corrigibles respecte l'inégalité suivante :*

$$\boxed{q^k \leqslant \frac{q^n}{\sum_{i=0}^{t}\binom{n}{i}(q-1)^i}} \tag{2.14}$$

Démonstration.

Le nombre d'erreurs corrigible est t donc pour tout couple $(\mathbf{c}_1,\mathbf{c}_2) \in \mathcal{C}^2$ tel que $\mathbf{c}_1 \neq \mathbf{c}_2$, on a

$$\mathsf{Ball}(\mathbf{c}_1,t) \cap \mathsf{Ball}(\mathbf{c}_2,t) = \varnothing$$

Or cela signifie que

$$\begin{aligned}
\sharp \bigcup_{\mathbf{c}\in\mathcal{C}} \mathsf{Ball}(\mathbf{c},t) &= \sum_{\mathbf{c}\in\mathcal{C}} \sharp\mathsf{Ball}(\mathbf{c},t) \\
&= \sum_{\mathbf{c}\in\mathcal{C}} \sum_{i=0}^{t} \binom{n}{i}(q-1)^i \\
&= \sharp\mathcal{C} \cdot \sum_{i=0}^{t} \binom{n}{i}(q-1)^i \\
&\leqslant \sharp\mathbb{F}_q^n = q^n
\end{aligned}$$

□

Définition 2.5.2 (code parfait)**.** *Un code en bloc linéaire* $[n,k]_q$ *est* parfait *si et seulement s'il atteint la borne de Hamming.*

Les codes en bloc linéaires parfaits sont peu nombreux. Nous pouvons en dresser la liste exhaustive :

- Les codes contenant un unique mot. Ils peuvent corriger toutes les erreurs.
- Les codes qui contiennent tous les mots de l'espace ambiant. Leur distance minimale est égale à 1 et donc ces codes ne corrigent aucune erreur.
- Les codes de répétition binaires de longueur impaire contenant les mots $(0,\cdots,0)$ et $(1,\cdots,1)$.
- Les codes de Hamming $\left[\frac{q^m-1}{q-1}, \frac{q^m-1}{q-1}-m\right]_q$ où m est un entier strictement positif. Ce sont tous les codes parfaits de distance minimale 3 ; c'est-à-dire pouvant corriger une erreur.
- Le code de Golay ternaire $[11,6]_3$. Sa distance minimale est 5 et c'est l'unique code parfait corrigeant jusqu'à 2 erreurs.
- Le code de Golay binaire $[23,12]_2$. Sa distance minimale est 7 et c'est l'unique code parfait corrigeant jusqu'à 3 erreurs.

Remarquons qu'il n'existe aucun code parfait corrigeant strictement plus de trois erreurs.

On peut donner une forme asymptotique de la borne de Hamming. Soient $R := \frac{k}{n}$ et $\delta := \frac{d_{\mathcal{C}}}{n} \leqslant 1 - \frac{1}{q}$ tous les deux constants par rapport à n. On a alors

$$\begin{aligned}
\frac{q^n}{\sum_{i=0}^{t}\binom{n}{i}(q-1)^i} &= q^{n-nh_q\left(\frac{t}{n}\right)+o(1)} \\
&= q^{n-nh_q\left(\frac{d_{\mathcal{C}}-1}{2n}\right)+o(1)} \\
&= q^{n\left(1-h_q\left(\frac{\delta}{2}-\frac{1}{2n}\right)\right)+o(1)} \\
&= q^{n\left(1-h_q\left(\frac{\delta}{2}\right)\right)+o(1)}
\end{aligned}$$

Donc

$$\boxed{R \leqslant \left(1-h_q\left(\tfrac{\delta}{2}\right)\right)+o(1)}$$

2.6 La borne de Plotkin

Théorème 2.6.1 (Borne de Plotkin)**.** *Soit* $\mathcal{C}$ *un* $[n,k,d]_q$ *code linéaire, alors on a nécessairement*

$$\boxed{d \leqslant \left(n-\tfrac{n}{q}\right)\left(1+\tfrac{1}{q^k-1}\right)} \tag{2.15}$$

Démonstration.

Soit $\mathbf{M} \in \mathbb{F}_q^{(q^k-1)\times n}$ telle que les lignes de $\mathbf{M}$ sont tous les mots dans $\mathcal{C}\backslash\{\mathbf{0}\}$.
Soit $A := \sharp\left\{\mathbf{M}_{i,j} \ : \ \mathbf{M}_{i,j} \neq 0, i \in [\![1, q^k-1]\!], j \in [\![1,n]\!]\right\}$. Par définition, on a

$$A \geqslant d(q^k-1)$$

D'autre part, pour toute colonne j de $\mathbf{M}$, on a

$$\begin{aligned}
\sharp\left\{\mathbf{M}_{i,j} \ : \ \mathbf{M}_{i,j} \neq 0, i \in [\![1, q^k-1]\!]\right\} &= |\mathbf{M}_{*,j}| \\
&= \sharp\mathcal{C} - \sharp\mathcal{C}'
\end{aligned}$$

où $\mathcal{C}' := \{\mathbf{c} \in \mathcal{C} \ : \ c_j = 0\}$ est un code linéaire (facile à montrer).

Montrons que $\mathcal{C}'$ a perdu au plus une dimension par rapport à $\mathcal{C}$. S'il n'existe aucun mot $\mathbf{c}' \in \mathcal{C}$ tel que $c'_j \neq 0$, alors $\dim(\mathcal{C}') = \dim(\mathcal{C}) = k$. En revanche, s'il existe $\mathbf{c}' \in \mathcal{C}$ tel que $c'_j = 1$, alors $\mathcal{C}' := \left\{\mathbf{c} - \alpha\mathbf{c}' \ : \ \alpha = c_j \text{ et } \mathbf{c} \in \mathcal{C}\backslash\{\alpha\mathbf{c}' \ : \ \alpha \in \mathbb{F}_q^*\}\right\}$ et donc $\dim(\mathcal{C}') = \dim(\mathcal{C}) - 1 = k - 1$. Si on réunit les deux cas possibles, on a $\dim(\mathcal{C}') \geqslant k-1$.

Finalement, on a $|\mathbf{M}_{*,j}| = \sharp\mathcal{C} - \sharp\mathcal{C}' \leqslant q^k - q^{k-1}$ et comme il y a n colonnes dans $\mathbf{M}$, on a

$$d(q^k - 1) \leqslant A \leqslant n\left(q^k - q^{k-1}\right)$$

□

Théorème 2.6.2 (Borne de Plotkin asymptotique)**.** *Soient deux constantes $R \in [0,1]$ et $\delta \in [0,1]$. Si $\mathcal{C}$ est un $[n, \lfloor Rn \rfloor, \lfloor \delta n \rfloor]_q$ code linéaire, alors pour $n \to \infty$, on a nécessairement*

$$\boxed{R \leqslant \max\left(1 - \tfrac{q\delta}{q-1}, 0\right) + o(1)} \tag{2.16}$$

Démonstration.

Si $\delta > \frac{q-1}{q}$, alors d'après la borne de Plotkin (non-asymptotique), on a

$$q^k \leqslant \frac{\lfloor \delta n \rfloor}{\lfloor \delta n \rfloor - n\frac{q-1}{q}} = O(1)$$

La preuve du cas $\delta \leqslant \frac{q-1}{q}$ est laissé en exercice (plus difficile). □

2.7 La borne de Gilbert-Varshamov

Les bornes de Singleton, Hamming et Plotkin majorent le rendement d'un code pour une distance minimale donnée. Cependant, nous ne savons pas si pour tout couple (R, δ) satisfaisant ces bornes, il existe un code avec ces paramètres.

Par exemple, on a généralement Singleton > Hamming donc ça n'est pas parce qu'on respecte la borne de Singleton que l'on peut construire un code car les paramètres de celui-ci pourrait ne pas respecter la borne de Hamming.

La borne de Gilbert-Varshamov permet de donner une borne jusqu'à laquelle on est sûr de pouvoir construire un code.

Théorème 2.7.1 (Gilbert-Varshamov pour les codes non linéaires)**.** *Soient q, n, d et N des entiers positifs tels que $1 \leqslant d \leqslant n$ et*

$$N \sum_{i=0}^{d-1} \binom{n}{i} (q-1)^i \leqslant q^n. \tag{2.17}$$

Alors il existe un code $\mathcal{C} \subseteq \mathbb{F}_q^n$ (non-linéaire), de distance minimale $\geqslant d$ et de cardinalité $\sharp\mathcal{C} = N$.

Démonstration.

Soit la procédure suivante pour construire un code $\mathcal{C} \subseteq \mathbb{F}_q^n$:

Algorithme 2.7.1 : Construction d'un code non-linéaire respectant la borne de Gilbert-Varshamov.

Entrées : q, n, d, N respectant les hypothèses du théorème.
Sortie : un code $\mathcal{C}$ (non-linéaire) satisfaisant le théorème.

1 $\mathcal{C} \leftarrow \varnothing$;
2 $\mathcal{U} \leftarrow \mathbb{F}_q^n$;
3 **tant que** $\sharp\mathcal{C} < N$ **faire**
4 $\quad \mathbf{c} \leftarrow \mathrm{Unif}(\mathcal{U})$;
5 $\quad \mathcal{C} \leftarrow \mathcal{C} \cup \{\mathbf{c}\}$;
6 $\quad \mathcal{U} \leftarrow \mathcal{U} \backslash \left\{\mathbf{x} \in \mathbb{F}_q^n : \Delta(\mathbf{x}, \mathbf{c}) \leqslant d-1\right\}$;
7 **finTantQue**
8 **retourner** $\mathcal{C}$;

Pour que la procédure ci-dessus termine avec un code contenant exactement N mots, il faut qu'à chaque itération, nous puissions effectuer l'étape 4 de l'algorithme ; en d'autres termes, il faut qu'à chaque itération, nous ayons $\mathcal{U} \neq \varnothing$. Or par construction, on a

$$\begin{aligned} \sharp\mathcal{U} &\geqslant \sharp\mathbb{F}_q^n - \sharp\mathcal{C} \times \sharp\mathsf{Ball}(d-1) \\ &\geqslant q^n - \sharp\mathcal{C} \cdot \textstyle\sum_{i=0}^{d-1} \binom{n}{i}(q-1)^i \end{aligned}$$

Et donc tant que $\sharp\mathcal{C} < N$, on a $\sharp\mathcal{U} > 0$.

En outre, l'étape 6 de l'algorithme garantit que les mots de $\mathcal{C}$ sont 2 à 2 distants d'au moins d. □

Théorème 2.7.2 (Gilbert-Varshamov pour les codes linéaires)**.** *Soient q, n, k et d des entiers positifs tels que $1 \leqslant k \leqslant n$, $2 \leqslant d \leqslant n$ et*

$$q^k \sum_{i=0}^{d-2} \binom{n-1}{i} (q-1)^i \leqslant q^n. \tag{2.18}$$

Alors il existe un code linéaire $\mathcal{C}$ de paramètres $[n, k, d]_q$.

Démonstration.

On va construire le code en complétant une matrice de parité du code colonne par colonne.

Tout d'abord, rappelons que tout code linéaire $[n, k, d]_q$ peut être représenté (à permutation près) par son unique matrice de parité systématique :

$$\mathbf{H} = [\mathbf{A}|\mathbf{Id}_{n-k}] \qquad \text{avec } \mathbf{A} \in \mathbb{F}_q^{(n-k)\times k}$$

On va donc construire $\mathbf{A}$ colonne par colonne de telle sorte que le code de matrice de parité $\mathbf{H}$ soit de distance minimale $\geqslant d$.

Algorithme 2.7.2 : Construction d'un code linéaire respectant la borne de Gilbert-Varshamov.

Entrées : q, n, k, d respectant les hypothèses du théorème.
Sortie : une matrice de parité $\mathbf{H}$ d'un code linéaire satisfaisant le théorème.

1 $\mathbf{H} \leftarrow \mathbf{Id}_{n-k}$;
2 **tant que** $\mathbf{H} \notin \mathbb{F}_q^{(n-k)\times n}$ **faire**
3 $\quad \mathbf{x} \leftarrow$ choisir un vecteur colonne uniformément dans $\mathbb{F}_q^{n-k}$ qui ne soit pas une combinaison linéaire de $d-2$ colonnes de $\mathbf{H}$;
4 $\quad \mathbf{H} \leftarrow [\mathbf{H} \,||\, \mathbf{x}]$;
5 **finTantQue**
6 **retourner** $\mathbf{H}$;

L'algorithme termine bien car on peut toujours trouver un vecteur $\mathbf{x}$ à l'étape 3 car $\mathbf{x}$ peut être choisi parmi

$$\sharp\mathbb{F}_q^{n-k} - \sum_{i=0}^{d-2} \binom{N_{col}(\mathbf{H})}{i} (q-1)^i$$

vecteurs. Or tant que $N_{col}(\mathbf{H}) < n$, on a

$$\sharp\mathbb{F}_q^{n-k} - \sum_{i=0}^{d-2} \binom{N_{col}(\mathbf{H})}{i} (q-1)^i > \sharp\mathbb{F}_q^{n-k} - \sum_{i=0}^{d-2} \binom{n-1}{i} (q-1)^i \geqslant 0$$

En outre, tout ensemble de moins de $d-1$ colonnes de $\mathbf{H}$ sont linéairement indépendants car chaque colonne n'est pas combinaison linéaire de $d-2$ autres colonnes. Et donc la distance minimale du code est $\geqslant d$. □

ATTENTION, les codes construits dans les deux démonstrations précédentes sont inutilisables en pratique car il n'ont pas assez de structure pour être décodé efficacement. En effet, on verra plus tard que décoder un code linéaire aléatoire est un problème NP-complet.

Bien que les hypothèses des deux théorèmes précédents soient légèrement différents, elles sont en fait asymptotiquement équivalentes lorsque $n \to \infty$. On pose $R := \frac{k}{n}$ (ou $R := \frac{\log_q(N)}{n}$ si le code n'est pas linéaire) et $\delta := \frac{d}{n}$. On suppose que $R \in [0,1]$ et $\delta \in [0, 1-\frac{1}{q}]$ sont des constantes par rapport à n. Alors

$$\boxed{N \sum_{i=0}^{d-1} \binom{n}{i} (q-1)^i \leqslant q^n \quad \Longrightarrow \quad R + h_q(\delta) \leqslant 1 + o(1)} \tag{2.19}$$

et

$$\boxed{q^k \sum_{i=0}^{d-2} \binom{n-1}{i} (q-1)^i \leqslant q^n \quad \Longrightarrow \quad R + h_q(\delta) \leqslant 1 + o(1)} \tag{2.20}$$

Pour montrer les deux formules asymptotiques ci-dessus, il suffit d'utiliser l'approximation de Stirling du coefficient binomial et de remarquer que les éléments d'une boule de rayon d se concentrent sur la sphère.

On va maintenant spécifier la distance relative δ pour laquelle on a l'égalité $R + h_q(\delta) = 1$. On appellera cette distance la distance de Gilbert-Varshamov.

La borne de Gilbert-Varshamov est aussi le plus grand rayon de décodage d tel que l'espérance du nombre de mots de code présents dans une boule de rayon d soit inférieure à 1.

Définition 2.7.3 (distance de Gilbert-Varshamov)**.** *Nous considérons des codes définis sur $\mathbb{F}_q^n$. Soit k un entier positif. La borne de Gilbert-Varshamov $d_{GV}(n,k)$ associée aux codes de taille q^k est :*

$$d_{GV}(n,k) := \sup\left\{ d \geqslant 0 \ : \ q^k \cdot \mathbb{P}\left(\mathbf{u} \in \mathsf{Ball}(d)\right) \leqslant 1 \right\} \tag{2.21}$$

$$= \sup\left\{ d \geqslant 0 \ : \ \sum_{i=0}^{d} \binom{n}{i} (q-1)^i \leqslant q^{n-k} \right\} \tag{2.22}$$

où $\mathsf{Ball}(d) := \left\{ \mathbf{x} \in \mathbb{F}_q^n \ : \ |\mathbf{x}| \leqslant r \right\}$ *et* $\mathbf{u}$ *est tiré uniformément dans* $\mathbb{F}_q^n$.

La distance de Gilbert-Varshamov peut se généraliser à d'autres espaces métriques bornés. Mais sur l'espace $\mathbb{F}_q^n$ muni de la métrique de Hamming, c'est une quantité bien connue. La proposition suivante en donne une expression asymptotique :

Proposition 2.7.4. *Pour un rendement $R = \frac{k}{n}$ constant et lorsque n tend vers l'infini, la distance relative de Gilbert-Varshamov est :*

$$\boxed{\delta_{GV}(R) = h_q^{-1}(1-R) + o(1)} \tag{2.23}$$

où $h_q(x) := x\log_q(q-1) - x\log_q(x) - (1-x)\log_q(1-x)$ est la fonction entropie q-aire de Shannon.

2.8 Exercice

1. Tracer avec SageMath, les bornes de Singleton, Hamming, Plotkin et Gilbert-Varshamov pour différents paramètres $[15, k, d]_2$, $[15, k, d]_{16}$, $[23, k, d]_2$, $[11, k, d]_3$, $[+\infty, k, d]_2$, $[+\infty, k, d]_8$... On mettre d (ou $\frac{d}{n}$) en abscisse et k (ou $\frac{k}{n}$) en ordonnée.
2. Sur ces figures, repérer quelques codes bien connus ($\mathcal{C} = \mathbb{F}_q^n$, $\mathcal{C} = \{\mathbf{0}\}$, code de parité, code de répétition, code de Reed-Solomon, code de Hamming, code de Golay binaire et ternaire...).
3. Sur ces mêmes figures, représenter l'ensemble des codes **possibles** et l'ensemble des **bons codes**.

Chapitre 3

Construire de nouveaux codes à partir de codes parents

Dans ce chapitre, on décrit plusieurs opérations pour transformer un code en un autre. Le but est souvent de pouvoir moduler la longueur et/ou la dimension et ou la taille du corps.

3.1 Les codes étendus

3.2 Les codes poinçonnés

3.3 Les codes raccourcis

3.4 Les subfield-subcodes

3.5 Les codes traces

Chapitre 4

Le décodage générique

Le problème du décodage générique consiste essentiellement à décoder dans un code pour lequel nous ne supposons aucune structure particulière, hormis le fait qu'il soit linéaire. Ce problème est très étudié depuis les années 60, notamment dans le cas binaire. Une très large famille de crypto-systèmes se reposent sur la difficulté que nous avons à résoudre ce problème. Une des raisons pour laquelle le décodage générique est intéressant en cryptologie est qu'il résiste aujourd'hui au paradigme des ordinateurs quantiques ; or ces machines menacent nos systèmes de communication actuels et dont la sécurité se repose généralement sur des problèmes de théorie des nombres.

Dans la première section de ce chapitre nous expliquons le challenge que doit relever la cryptologie de demain face à la menace que représentent les ordinateurs quantiques. Dans cette section, nous évoquons notamment les dernières avancées dans le domaine de la conception de processeurs quantiques. Nous présentons aussi une compétition du NIST (*National Institute of Standards and Technologie*) qui sollicite la communauté internationale des cryptologues pour proposer les futurs standards cryptographiques qui devront être à l'épreuve des ordinateurs quantiques. Dans la section 9.2, nous présentons quelques crypto-systèmes basés sur le problème du décodage générique. La section 4.1 définit formellement le problème du décodage générique et reformule celui-ci dans une version dual. Enfin, les sections 4.2 à 4.5 sont consacrées à la description de méthodes algorithmiques permettant de résoudre le plus efficacement possible ce problème. Chacune de ces sections introduit une avancée majeure dans le domaine.

Les méthodes que nous présentons ont initialement été proposées pour les espaces binaires mais nous les généralisons ici aux espaces non-binaires. Pour la plupart d'entre elles, cette généralisation existe déjà dans la littérature, sauf pour la dernière méthode : le décodage générique de Both et May n'existe que dans une version binaire [BM18]. L(adaptation de cet algorithme aux espace de Hamming non-binaires est donc une contribution de cette thèse.

Dans la section 4.6, nous anticipons les travaux que nous présenterons dans les chapitres ultérieurs en donnant quelques résultats numériques sur nos propres méthodes de décodage générique. Cette section motive la suite du manuscrit puisqu'elle montre que nos méthodes de recherche de presque-collisions permettent d'améliorer les méthodes de décodage générique binaires et non-binaires et d'obtenir les meilleures complexités du moment pour résoudre ce problème.

Enfin la dernière section de ce chapitre traite le problème LPN (*Learning from Parity with Noise*) qui est une variante du problème du décodage générique. Cette section reprend essentiellement les travaux de [EKM17].

4.1 Le décodage par syndrome

Dans la suite de ce chapitre, nous supposons que les codes sont définis sur le corps $\mathbb{F}_q$ muni de la métrique de Hamming. Le problème du décodage générique (en anglais, *generic decoding problem*) est alors :

Problème 4.1.1 (décodage générique)**.** *Soit $\mathcal{C}$ un code en bloc binaire linéaire $[n,k]_q$. Étant donné un mot $\mathbf{y} \in \mathbb{F}_q^n$ et un entier $w \in [\![0,n]\!]$, le but est de trouver un mot de code $\mathbf{x} \in \mathcal{C}$ tel que $\Delta(\mathbf{x},\mathbf{y}) \leqslant w$.*

Ce problème est généralement un problème difficile : il a été montré que le problème de décision sous-jacent est NP-complet lorsqu'aucune hypothèse n'est faite sur une éventuelle structure du code. Nous avons vu dans la section précédente que c'est notamment sur cette difficulté que repose la sécurité de plusieurs crypto-systèmes basés sur les codes dont celui de McEliece [McE78].

En général, la valeur de w est relativement faible ; en particulier $w \in \left[\!\left[0, n - \frac{n}{q}\right]\!\right]$ (par exemple, de l'ordre de $\sqrt{n}$ dans le crypto-système BIKE). La question du décodage générique en gros poids – c'est-à-dire lorsque $w \in \left[\!\left[n - \frac{n}{q}, n\right]\!\right]$ – s'est posée très récemment dans [DST19, BCDL20]. Dans ce chapitre, nous ne traitons pas les gros poids.

Plutôt que de traiter le problème 4.1.1, nous nous intéresserons au problème équivalent du décodage par syndrome (en anglais, *syndrome decoding problem*) :

Problème 4.1.2 (décodage par syndrome)**.** *Soit une matrice $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$. Étant donné un vecteur $\mathbf{s} \in \mathbb{F}_q^{n-k}$ et un entier $w \in \left[\!\left[0, n - \frac{n}{q}\right]\!\right]$, le but est de trouver un vecteur $\mathbf{e} \in \mathbb{F}_q^n$ de poids w tel que $\mathbf{H}\mathbf{e}^\top = \mathbf{s}^\top$.*

Soit $\mathbf{H}$ une matrice de parité du code $\mathcal{C}$ et soit $\mathbf{y} \in \mathbb{F}_q^n$. Posons alors $\mathbf{s} = \mathbf{y}\mathbf{H}^\top$. On dit que $\mathbf{s}$ est le syndrome du mot $\mathbf{y}$. Le décodage par syndrome appliqué à $\mathbf{s}$ réalise alors bien un décodage du mot $\mathbf{y}$ dans le code $\mathcal{C}$. En effet, le vecteur $\mathbf{e}$ recherché dans le problème 4.1.2 est tel que $\mathbf{y} - \mathbf{e}$ est un mot du code $\mathcal{C}$ puisque par construction, $\mathbf{H}(\mathbf{y}-\mathbf{e})^\top = \mathbf{H}\mathbf{y}^\top - \mathbf{H}\mathbf{e}^\top = \mathbf{s} - \mathbf{s} = \mathbf{0}$.

Nous nous intéressons particulièrement au cas asymptotique où n tend vers l'infini. On suppose alors $k := \lfloor Rn \rfloor$ et $w := \lfloor \omega n \rfloor$ pour des constantes $R \in [0,1]$ et $\omega \in \left[01 - \frac{1}{q}\right]$.

Le décodage générique depuis 1962. La complexité du meilleur algorithme connu à ce jour pour résoudre le problème du décodage par syndrome est de la forme :

$$2^{(\alpha + o(1))n} \tag{4.1}$$

où le facteur $2^{o(1)n}$ est négligeable devant le facteur exponentiel $2^{\alpha n}$. L'exposant α quant-à lui ne dépend que du rendement R du code et du poids relatif ω de l'erreur à décoder. Les méthodes de décodage les plus efficaces sont des décodages par ensemble d'information. Cette famille d'algorithmes a été introduite par Prange en 1962 [Pra62]. Depuis, de nombreuses améliorations ont été proposées [Leo82, LB88, Ste88, Dum91, Bar97b, BLP11, MMT11, BJMM12, MO15, Hir16, GKH17, BM17b, BM18]. Toutefois, la meilleur méthode reste exponentielle en la dimension k du code et le poids w de l'erreur à décoder. La complexité du meilleur algorithme de décodage impacte directement les paramètres des crypto-systèmes basés sur les codes ; par exemple, il impose certaines bornes minimales pour les tailles de clés.

Sur la figure 4.1, nous avons représenté l'exposant α de l'équation (4.1) pour différentes méthodes de décodage générique dans le cas binaire. Les exposants en rouge sont ceux où le facteur $2^{o(1)n}$ est super-polynomial tandis que pour les autres, ce facteur est polynomial. De plus, pour chaque méthode représentée sur cette figure, le rendement du code considéré est celui pour lequel la méthode est la moins performante. En outre, la figure (a) concerne le décodage à la distance de Gilbert-Varshamov (*full decoding*) qui est le cas le plus difficile à traiter. En effet, le problème du décodage est de plus en plus difficile à mesure que la distance de décodage augmente, sauf lorsque cette distance dépasse $d_{\mathrm{GV}}(n,k)$ car alors le nombre de solutions est démultiplié. Enfin, la figure (b) représente la complexité du décodage à la moitié de la distance précédente (*half decoding*).

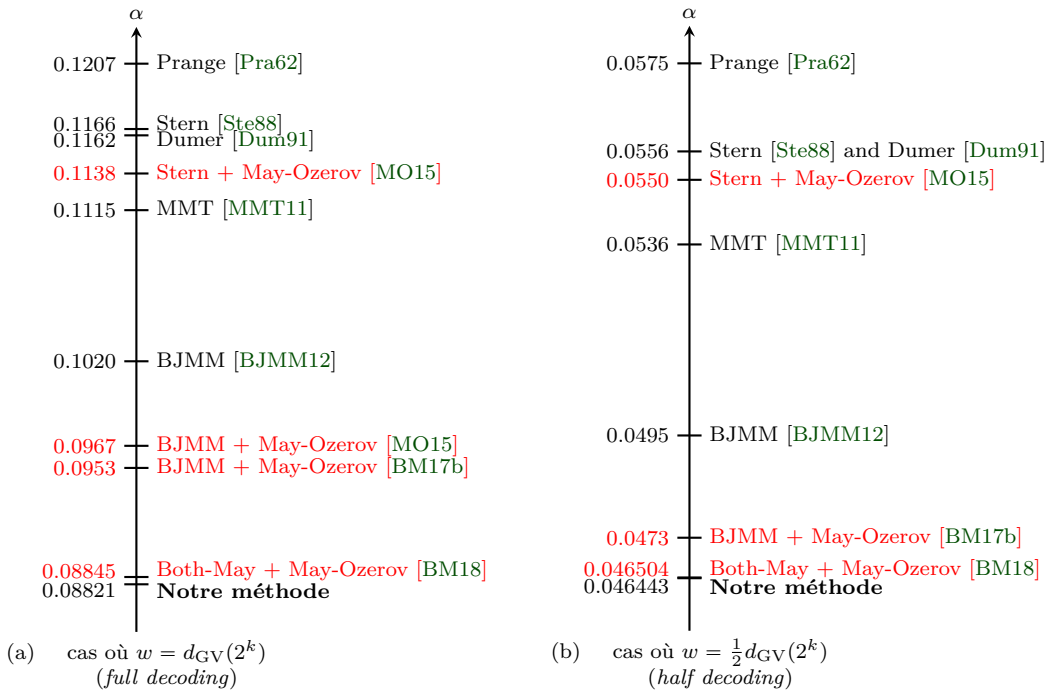


Figure 4.1 – Exposants des complexités asymptotiques des algorithmes de décodage par ensemble d'information dans le cas binaire.

Sur la figure 4.1, nous avons anticipé les résultats que nous détaillerons tout au long de ce manuscrit en donnant nos propres exposants pour résoudre le décodage générique. En *full decoding*, nous améliorons légèrement les meilleurs exposants connus à ce jour tandis qu'en *half decoding*, notre amélioration est trop faible pour être jugée réellement significative. Cependant, il faut noter que notre méthode ne souffre pas d'un surcoût super-polynomial contrairement aux dernières méthodes de la littérature. Nous conjecturerons même dans le dernier chapitre que notre facteur est polynomial.

Revenons quelques instants sur la taille du corps q. Dans la littérature, le cas binaire semble susciter beaucoup plus d'intérêt que les cas non-binaires. La première raison à cela est que les crypto-systèmes sont très rarement définis sur des corps non-binaires. De plus, il a été montré dans [Can17] qu'augmenter la taille du corps a un intérêt limité. En effet, il est montré dans ce papier que toutes les adaptations sur $\mathbb{F}_q$ des méthodes de décodage par ensemble d'information sont asymptotiquement équivalentes à la méthode de Prange lorsque q tend vers l'infini. Toutefois, le décodage par ensemble d'information sur des corps non-binaires peut avoir un intérêt lorsque le corps n'est pas trop grand ; notamment pour le

décodage en gros poids [DST19, BCDL20] ou encore dans le domaine de la reconnaissance de codes non-binaires tels que les codes LDPC non-binaires. Dans la section 4.6 de ce chapitre, nous donnons quelques résultats numériques qui montrent que nous améliorons significativement le décodage générique sur des corps non-binaires.

Il existe une autre famille de décodages : les décodages statistiques [Jab01, Ove06]. Ceux-ci sont une éventuelle alternative aux décodages par ensemble d'information. Cependant, Debris et Tillich ont montré que de tels décodages sont moins efficaces que la méthode de Prange dans les régimes les plus interessants ; à savoir pour des décodages à la distance de Gilbert-Varshamov ou même à des distances plus réalistes en cryptographie [DT17].

4.2 Le décodage par ensemble d'information

L'algorithme de Prange. Initiées par Prange en 1962 dans [Pra62], des méthodes de décodage par ensemble d'information (en anglais, *Information Set Decoding* ou ISD) sont développées pour résoudre le problème du décodage par syndrome. L'idée de Prange réside dans le caractère fondamental des codes linéaires ; à savoir qu'un mot d'un code linéaire $[n, k]$ est entièrement défini par k bits d'information.

Pour des questions de lisibilité, nous introduisons les notations suivantes :

Notation 4.2.1. *Soit* $\mathbf{x} := (x_1, \cdots, x_n) \in \mathbb{F}_q^n$ *et soit une liste d'indices* $I := (i_j)_{1 \leqslant j \leqslant t} \subseteq [\![1, n]\!]$. *On note* $\mathbf{x}_I$ *le vecteur suivant :*

$$\mathbf{x}_I := (x_{i_1}, \cdots, x_{i_t}) \in \mathbb{F}_q^t \tag{4.2}$$

De façon analogue, soit $\mathbf{M} \in \mathbb{F}_q^{m \times n}$. *On note* $\mathbf{M}_I$ *la matrice constituée des colonnes de* $\mathbf{M}$ *indexées par* I.

Il faut noter que l'ordre des indices dans la liste I *définit éventuellement une permutation des positions du vecteur ou des colonnes de la matrice.*

Définition 4.2.2. *Soit* $\mathcal{C}$ *un code linéaire* $[n, k]_q$. *Les ensemble d'information de* $\mathcal{C}$ *sont tous les ensembles d'indices* I *tels que* $\#I = k$ *et* $\#\{\mathbf{x}_I \ : \ \mathbf{x} \in \mathcal{C}\} = \#\mathcal{C} = q^k$.

Autrement dit, si $\mathbf{H}$ *est une matrice de parité du code* $\mathcal{C}$, *alors un ensemble d'indices* I *de taille* k *est un ensemble d'information du code* $\mathcal{C}$ *si et seulement si* $\mathbf{H}_J$ *est inversible avec* $J := [\![1, n]\!] \setminus I$.

Comme nous le disions précédemment, la méthode de Prange, donnée par l'algorithme 4.2.1, exploite l'idée que les k positions d'un ensemble d'information suffisent pour définir entièrement un unique mot du code $\mathcal{C}$. On pose donc I un ensemble d'information de $\mathcal{C}$. Puisque le vecteur d'erreur recherché est de poids au plus w avec w supposé relativement petit par rapport à n, il est tout à fait raisonnable de supposer que $\mathbf{e}_I = \mathbf{0}$. Sinon, il suffit d'itérer jusqu'à sélectionner un ensemble d'information qui vérifie bien cette propriété.

Algorithme 4.2.1 : Algorithme de Prange

Entrées : une matrice de parité $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$ d'un code $\mathcal{C}$;
un syndrome $\mathbf{s} \in \mathbb{F}_q^{n-k}$;
le poids maximal w du vecteur d'erreur recherché ;
Sortie : $\mathbf{e} \in \mathbb{F}_q^n$ tel que $\mathbf{He}^\top = \mathbf{s}^\top$ et $|\mathbf{e}| = w$.

1 **répéter**
2 choisir une liste $I \subseteq [\![1, n]\!]$ telle que $\#I = k$;
3 $J \leftarrow [\![1, n]\!] \setminus I$;
4 $\overline{\mathbf{s}} \leftarrow \mathbf{H}_J^{-1}\mathbf{s}^\top$; `/* si` $\mathbf{H}_J$ `n'est pas inversible revenir à l'étape 2 */`
5 **tant que** $|\overline{\mathbf{s}}| \neq w$;
6 **retourner** $\mathbf{e}$ tel que $\mathbf{e}_I = \mathbf{0}$ et $\mathbf{e}_J = \overline{\mathbf{s}}^\top$;

L'algorithme de Prange retourne bien le résultat attendu puisque $|\mathbf{e}| \leqslant w$ et :

$$\begin{aligned} \mathbf{He}^\top &= \mathbf{H}_J\mathbf{e}_J^\top \\ &= \mathbf{H}_J\overline{\mathbf{s}} \\ &= \mathbf{H}_J\mathbf{H}_J^{-1}\mathbf{s}^\top \\ &= \mathbf{s}^\top \end{aligned}$$

Remarque 4.2.1. En choisissant un ensemble de k indices aléatoirement lors d'une itération de l'algorithme 4.2.1, il se peut que celui-ci ne forme pas un ensemble d'information. Dans ce cas, un autre ensemble d'indices est choisi. Toutefois, nous avons dû effectuer une élimination gaussienne sur la matrice $\mathbf{H}$ pour nous rendre compte que le premier ensemble n'était pas valide. Or cette opération a un coût. Une astuce pour éviter d'avoir à effectuer plusieurs inversions de matrices lors d'une itération est de modifier dynamiquement l'ensemble d'indices I lors de l'élimination gaussienne pour remplacer les colonnes sans pivot par d'autres colonnes de $\mathbf{H}$. Cette façon de procéder ne produit pas un tirage uniforme sur l'ensemble des ensemble d'information mais le biais engendré est négligeable.

Théorème 4.2.3. *Soient un code linéaire $[n, k]_q$ et un entier w tels que $k := \lfloor Rn \rfloor$ et $w := \lfloor \omega n \rfloor$ pour des constantes $R \in [0, 1]$ et $\omega \in \left[0, \left(1 - \frac{1}{q}\right)(1 - R)\right]$. Lorsque n tend vers l'infini, la complexité en temps de l'algorithme de Prange est :*

$$T_{Prange} = O\left(\frac{\min\left(\binom{n}{w}(q-1)^w, q^{n-k}\right)}{\binom{n-k}{w}(q-1)^w}\right) \tag{4.3}$$

$$= q^{n\left(\min(1-R, h_q(\omega)) - (1-R)h_q\left(\frac{\omega}{1-R}\right)\right)(1+o(1))} \tag{4.4}$$

où le facteur $q^{o(1)n}$ est polynomial en n.

Remarque 4.2.2. Dans [BCDL20], il est proposé une généralisation de l'algorithme de Prange qui permet de traiter les cas où $\omega \geqslant \left(1 - \frac{1}{q}\right)(1 - R)$. Cet algorithme permet notamment de résoudre le problème du décodage par syndrome en un temps polynomial lorsque $\omega \in \left[\left(1 - \frac{1}{q}\right)(1 - R), R + \left(1 - \frac{1}{q}\right)(1 - R)\right]$.

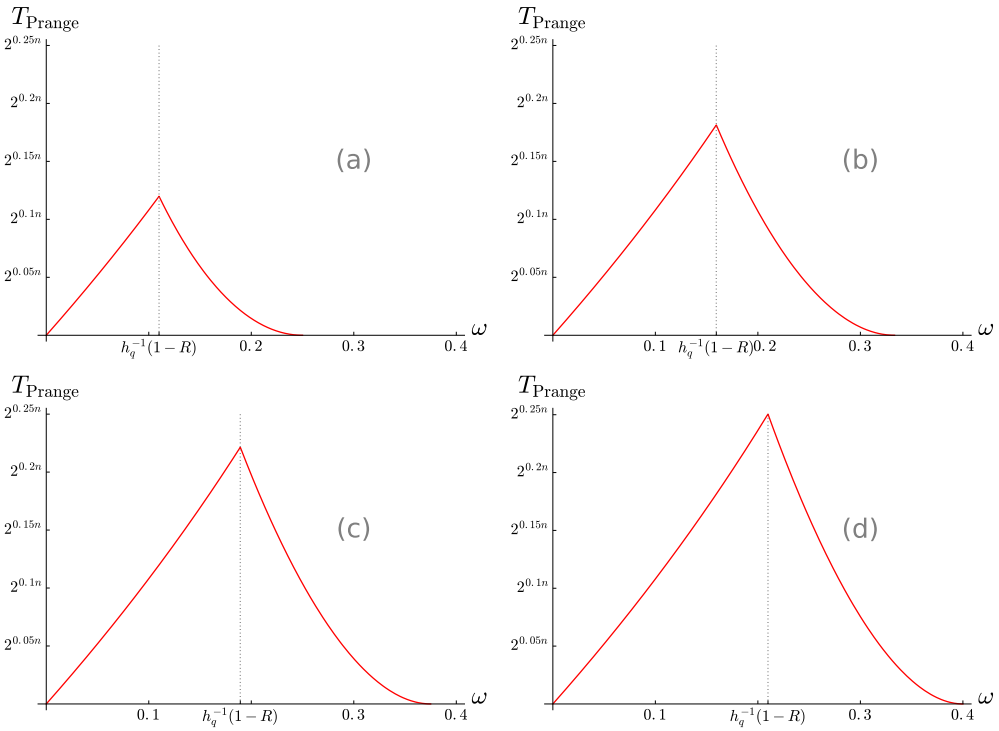


Figure 4.2 – Complexité asymptotique de l'algorithme de Prange pour $R = \frac{1}{2}$. (a) $q = 2$, (b) $q = 3$, (c) $q = 4$, (d) $q = 5$.

La figure 4.2 montre l'exposant asymptotique de l'algorithme de Prange pour différentes tailles de corps q et pour un rendement $R = \frac{1}{2}$. La figure 4.3 donne ce même exposant en fonction de R pour un décodage à la distance de Gilbert-Varshamov.

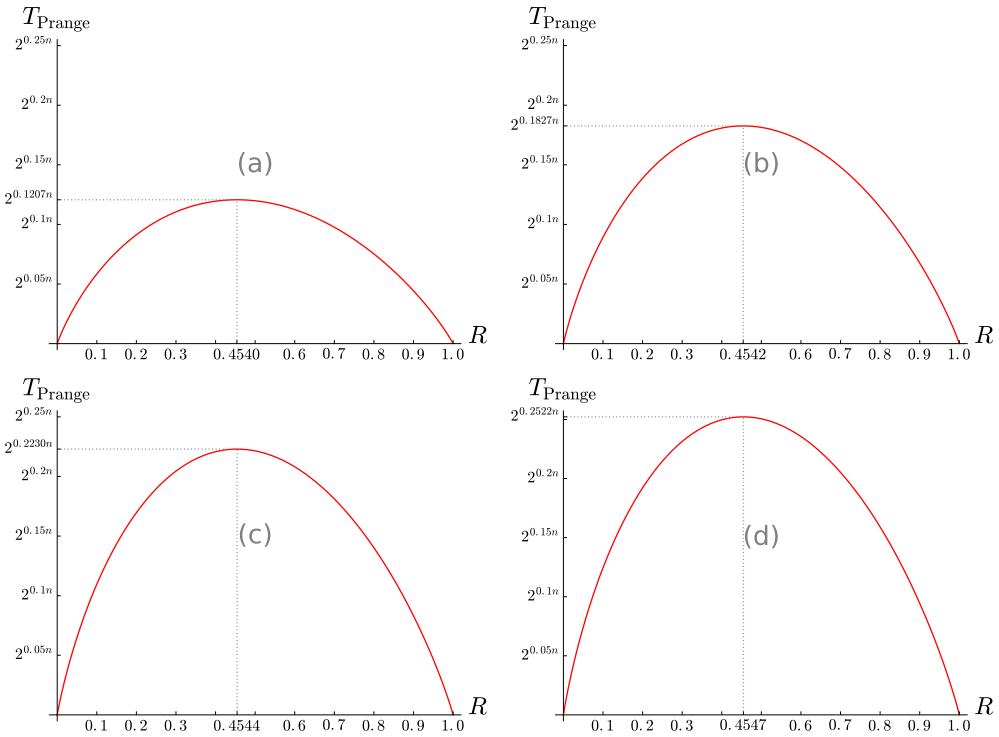


Figure 4.3 – Complexité asymptotique de l'algorithme de Prange pour $\omega = \frac{d^-_{\text{GV}}(n,k)}{n} := h_q^{-1}(1-R)$. (a) $q=2$, (b) $q=3$, (c) $q=4$, (d) $q=5$.

Dans la suite, nous supposons que w est plus petit que la distance de Gilbert-Varshamov $d^-_{\text{GV}}(n,k) = nh_q^{-1}(1-R)+o(1)$ qui est la distance la plus difficile à décoder. Cette distance correspond aussi au seuil au delà duquel le problème du décodage générique a un nombre exponentiel de solutions ; or dans nos applications, nous nous plaçons généralement dans un régime où la solution est unique. La complexité de l'algorithme de Prange devient alors :

$$T_{\text{Prange}} = O\left(\frac{\binom{n}{w}}{\binom{n-k}{w}}\right) \tag{4.5}$$

$$= \tilde{O}\left(q^{n\left(h_q(\omega)-(1-R)h_q\left(\frac{\omega}{1-R}\right)\right)}\right) \tag{4.6}$$

L'algorithme de Lee et Brickell. Dans l'algorithme de Prange, la contrainte $\mathbf{e}_I = \mathbf{0}$ est parfois trop forte. La méthode de Lee et Brickell [LB88] généralise l'algorithme 4.2.1 en supposant que $|\mathbf{e}_I| = p$ où p est un paramètre à déterminer. Les vecteurs de longueur k et de poids p sont alors parcourus exhaustivement jusqu'à trouver un vecteur $\mathbf{x} \in \mathbb{F}_q^k$ tel que $|\mathbf{x}| = p$, $\left|\overline{\mathbf{s}} - \overline{\mathbf{P}}\mathbf{x}^\top\right| = w - p$ où $\overline{\mathbf{P}} := \mathbf{H}_J^{-1}\mathbf{H}_I$. L'algorithme Lee-Brickell retourne alors le vecteur $\mathbf{e}$ tel que $\mathbf{e}_I := \mathbf{x}$ et $\mathbf{e}_J^\top := \overline{\mathbf{s}} - \overline{\mathbf{P}}\mathbf{x}^\top$; ce qui est bien le résultat attendu puisque par construction $|\mathbf{e}| \leqslant w$ et :

$$\begin{aligned}
\mathbf{H}\mathbf{e}^\top &= \mathbf{H}_I\mathbf{e}_I^\top + \mathbf{H}_J\mathbf{e}_J^\top \\
&= \mathbf{H}_I\mathbf{e}_I^\top + \mathbf{H}_J\left(\overline{\mathbf{s}} - \overline{\mathbf{P}}\mathbf{e}_I^\top\right) \\
&= \mathbf{H}_I\mathbf{e}_I^\top + \mathbf{H}_J\left(\mathbf{H}_J^{-1}\mathbf{s}^\top - \mathbf{H}_J^{-1}\mathbf{H}_I\mathbf{e}_I^\top\right) \\
&= \mathbf{s}^\top
\end{aligned}$$

Idéalement, on aimerait pouvoir fixer $p \simeq \omega k$ qui est le poids typique de $\mathbf{e}_I$ lorsque I est tiré uniformément parmi les ensemble d'information du code. Cependant, si p est trop grand, il peut être extrêmement coûteux de tester exhaustivement tous les vecteurs de poids p. Il faut donc trouver un compromis entre choisir un p suffisamment petit pour que la recherche exhaustive de $\mathbf{e}_I$ ne soit pas trop coûteuse et un p suffisamment proche de ωk pour que la probabilité de succès soit raisonnable. Le théorème suivant donne la complexité de l'algorithme Lee-Brickell pour une valeur de p optimale.

Théorème 4.2.4. *Soient un code linéaire $[n,k]_q$ et un entier w tels que $k := \lfloor Rn \rfloor$ et $w := \lfloor \omega n \rfloor$ pour des constantes $R \in [0,1]$ et $\omega \in \left[0, h_q^{-1}(1-R)\right]$. Lorsque n tend vers l'infini, la complexité en temps de l'algorithme Lee-Brickell est :*

$$T_{\text{Lee-Brickell}} = \tilde{O}(q^{\alpha n}) \tag{4.7}$$

où :

$$\alpha \quad := \quad \gamma \log_q(q-1) + h_q(\omega) - (1-R) h_q\left(\tfrac{\omega-\gamma}{1-R}\right) \tag{4.8}$$

avec $\gamma := \max\left(0, \omega - \left(1 - \frac{1}{q}\right)(1-R)\right)$.

4.3 Le décodage par recherche de collisions

4.3.1 La méthode de Stern

Dans la méthode Lee-Brickell, le coût de la recherche exhaustive est de l'ordre de $O\left(\binom{k}{p}(q-1)^p\right)$. La méthode proposée par Stern en 1988 [Ste88] s'inspire du paradoxe des anniversaires pour réduire ce coût. L'algorithme de Stern, détaillé dans l'algorithme 4.3.1, débute exactement comme celui de Lee-Brickell : un ensemble d'information I de taille k est choisi aléatoirement ; l'ensemble complémentaire est alors noté J. La matrice $\overline{\mathbf{P}} := \mathbf{H}_J^{-1}\mathbf{H}_I$ ainsi que le vecteur $\overline{\mathbf{s}} := \mathbf{H}_J^{-1}\mathbf{s}^\top$ sont produits au moyen d'une élimination gaussienne sur la matrice $\mathbf{H}$. Il nous reste alors à rechercher un vecteur $\mathbf{e}_I \in \mathbb{F}_q^k$ tel que $|\mathbf{e}_I| = p$ et $|\mathbf{e}_J| = w - p$ où $\mathbf{e}_J^\top := \overline{\mathbf{s}} - \overline{\mathbf{P}}\mathbf{e}_I^\top$. Notons tout d'abord que le vecteur $\mathbf{e}_J$ est de poids faible ; nous pouvons donc supposer avec une bonne probabilité qu'il est nul sur un ensemble L de ℓ positions si ℓ est suffisamment petit. Ainsi, dans la méthode de Stern, la matrice $\overline{\mathbf{P}}$ est subdivisée comme décrit sur la figure 4.4 (éventuellement à permutation près des lignes).

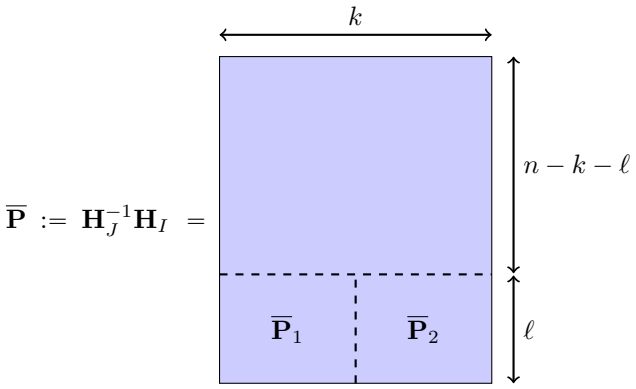


Figure 4.4 – Découpage de $\overline{\mathbf{P}}$ pour la méthode de Stern.

Une simple table de hachage permet alors de trouver les collisions entre les deux ensembles suivants :

$$\mathscr{L}_1 \quad := \quad \left\{ \bar{\mathbf{s}}_L - \overline{\mathbf{P}}_1 \mathbf{x}_1^\top \in \mathbb{F}_q^\ell \ : \ \mathbf{x}_1 \in \mathbb{F}_q^{\frac{k}{2}} \text{ et } |\mathbf{x}_1| = \tfrac{p}{2} \right\} \tag{4.9}$$

$$\mathscr{L}_2 \quad := \quad \left\{ \overline{\mathbf{P}}_2 \mathbf{x}_2^\top \in \mathbb{F}_q^\ell \ : \ \mathbf{x}_2 \in \mathbb{F}_q^{\frac{k}{2}} \text{ et } |\mathbf{x}_2| = \tfrac{p}{2} \right\} \tag{4.10}$$

Remarque 4.3.1. Nous avons implicitement supposé que k et p sont pairs. Des corrections triviales permettent de généraliser notre propos aux cas où k et p sont de parités quelconques.

Posons alors $(\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{F}_q^{\frac{k}{2}} \times \mathbb{F}_q^{\frac{k}{2}}$ un couple solution de la recherche de collisions sur $\mathscr{L}_1 \times \mathscr{L}_2$. Nous avons donc $|\mathbf{x}_1| = |\mathbf{x}_2| = \frac{p}{2}$ et $\bar{\mathbf{s}}_L - \overline{\mathbf{P}}_1 \mathbf{x}_1^\top = \overline{\mathbf{P}}_2 \mathbf{x}_2^\top$. Si $\left| \bar{\mathbf{s}} - \overline{\mathbf{P}}(\mathbf{x}_1, \mathbf{x}_2)^\top \right| = w - p$, alors le vecteur $\mathbf{e}$ tel que $\mathbf{e}_I = (\mathbf{x}_1, \mathbf{x}_2)$ et $\mathbf{e}_J^\top = \bar{\mathbf{s}} - \overline{\mathbf{P}}(\mathbf{x}_1, \mathbf{x}_2)^\top$ est une solution du problème de décodage par syndrome.

Par rapport à la méthode Lee-Brickell, deux contraintes supplémentaires apparaissent sur la répartition du poids du vecteur d'erreur $\mathbf{e}$ à décoder. La première est que la moitié du support de $\mathbf{e}_I$ doit être contenue sur les $\frac{k}{2}$ premières positions de $\mathbf{e}_I$ et l'autre moitié sur les $\frac{k}{2}$ positions suivantes. La seconde contrainte est que le vecteur $\mathbf{e}_J$ doit être nul sur ℓ positions fixées. Ainsi, la probabilité de succès d'une itération de la méthode de Stern est :

$$\mathbb{P}_{\text{succ}} = \frac{\binom{k/2}{p/2}^2 \binom{n-k-\ell}{w-p}}{\binom{n}{w}} \tag{4.11}$$

Algorithme 4.3.1 : Algorithme de Stern

Entrées : une matrice de parité $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$ d'un code $\mathcal{C}$;
un syndrome $\mathbf{s} \in \mathbb{F}_q^{n-k}$;
le poids maximal w du vecteur d'erreur recherché ;
Paramètres : un entier $\ell \in [\![0, n-k]\!]$;
un entier pair $p \in [\![0, \min(w,k)]\!]$;
Sortie : $\mathbf{e} \in \mathbb{F}_q^n$ tel que $\mathbf{H}\mathbf{e}^\top = \mathbf{s}^\top$ et $|\mathbf{e}| = w$.

répéter indéfiniment
  choisir un ensemble $I \subseteq [\![1, n]\!]$ tel que $|I| = k$;
  $J \leftarrow [\![1, n]\!] \setminus I$;
  $\bar{\mathbf{s}} \leftarrow \mathbf{H}_J^{-1} \mathbf{s}^\top$; `/* si H_J n'est pas inversible revenir à l'étape 2 */`
  $\overline{\mathbf{P}} \leftarrow \mathbf{H}_J^{-1} \mathbf{H}_I$;
  choisir un ensemble $L \subseteq [\![1, n-k]\!]$ tel que $|L| = \ell$;
  extraire les sous-matrices $\overline{\mathbf{P}}_1 \in \mathbb{F}_q^{\ell \times \frac{k}{2}}$ et $\overline{\mathbf{P}}_2 \in \mathbb{F}_q^{\ell \times \frac{k}{2}}$ tel que $\left[\overline{\mathbf{P}}_1 | \overline{\mathbf{P}}_2\right]^\top = \left(\overline{\mathbf{P}}\right)_L$;
  `/* cf. figure 4.4 */`
  $\mathscr{L}_1 \leftarrow \left\{ \bar{\mathbf{s}}_L - \overline{\mathbf{P}}_1 \mathbf{x}_1^\top \ : \ \mathbf{x}_1 \in \mathbb{F}_q^{\frac{k}{2}} \text{ et } |\mathbf{x}_1| = \frac{p}{2} \right\}$;
  $\mathscr{L}_2 \leftarrow \left\{ \overline{\mathbf{P}}_2 \mathbf{x}_2^\top \ : \ \mathbf{x}_2 \in \mathbb{F}_q^{\frac{k}{2}} \text{ et } |\mathbf{x}_2| = \frac{p}{2} \right\}$;
  pour tout $\bar{\mathbf{s}}_L - \overline{\mathbf{P}}_1 \mathbf{x}_1^\top = \overline{\mathbf{P}}_2 \mathbf{x}_2^\top \in \mathscr{L}_1 \cap \mathscr{L}_2$ **faire**
    $\mathbf{x} \leftarrow (\mathbf{x}_1, \mathbf{x}_2)$;
    construire $\mathbf{e}$ tel que $\mathbf{e}_I = \mathbf{x}$ et $\mathbf{e}_J^\top = \bar{\mathbf{s}} - \overline{\mathbf{P}}\mathbf{x}^\top$;
    si $|\mathbf{e}| = w$ **alors**
      retourner $\mathbf{e}$;
    finSi
  finPour
finRépéter

Théorème 4.3.1. *Soit un code linéaire $[n,k]_q$ et soit un entier w tels que $k := 2\left\lfloor \frac{Rn}{2} \right\rfloor$ et $w := \lfloor \omega n \rfloor$ pour des constantes $R \in [0,1]$ et $\omega \in \left[0, h_q^{-1}(1-R)\right]$. Lorsque n tend vers l'infini, la complexité en temps de l'algorithme de Stern après optimisation des paramètres est :*

$$T_{Stern} = \tilde{O}(q^{\alpha n}) \tag{4.12}$$

où :

$$\alpha := h_q(\omega) - \lambda - (1-R-\lambda) h_q\left(\frac{\omega-\gamma}{1-R-\lambda}\right) \tag{4.13}$$

avec $\lambda := \frac{R}{2} h_q\left(\frac{\gamma}{R}\right)$ et $\gamma \in [0, \min(\omega, R)]$ qui minimise la valeur de α.

Les paramètres optimaux de l'algorithme de Stern sont alors $\ell := \lfloor \lambda n \rfloor$ et $p := 2\lfloor \frac{\gamma n}{2} \rfloor$.

4.3.2 Des presque-collisions dans la méthode de Stern

Dans [MO15], May et Ozerov généralisent la méthode de Stern en traitant chaque itération comme un problème de recherche de presque-collisions. Ce dernier s'énonce ainsi :

Problème 4.3.2 (recherche de presque-collisions)**.** *Étant données deux listes $\mathscr{L}_1$ et $\mathscr{L}_2$ d'éléments de $\mathbb{F}_q^n$ et une distance $w \in [\![1,n]\!]$, trouver les couples $(\mathbf{x},\mathbf{y}) \in \mathscr{L}_1 \times \mathscr{L}_2$ tels que $\Delta(\mathbf{x},\mathbf{y}) = w$.*

Pour être précis, dans [MO15], les auteurs ne traitent que le cas binaire. Dans cette sous-section, nous généralisons leur approche aux cas non-binaires ; généralisation dont une version est présentée dans [Hir16]. Pour faire apparaître le problème 4.3.2 dans la méthode de Stern, nous devons tout d'abord modifier le découpage de la matrice $\overline{\mathbf{P}}$ pour obtenir celui de la figure 4.5.

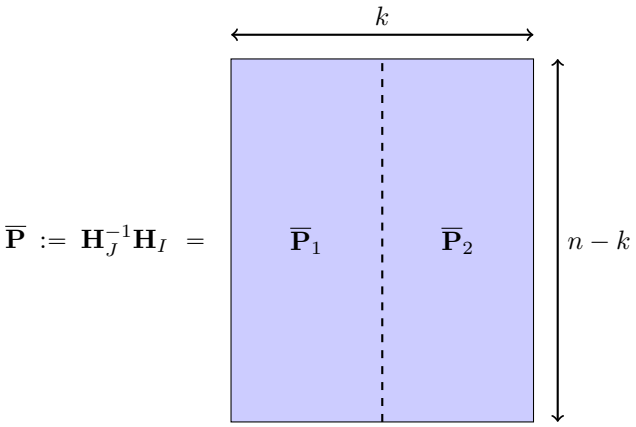


Figure 4.5 – Découpage de $\overline{\mathbf{P}}$ pour la version de May et Ozerov de la méthode de Stern.

Soient $\mathscr{L}_1$ et $\mathscr{L}_2$ les deux listes suivantes :

$$\mathscr{L}_1 := \left\{ \overline{\mathbf{s}} - \overline{\mathbf{P}}_1 \mathbf{x}_1^\top \in \mathbb{F}_q^{n-k} \ : \ \mathbf{x}_1 \in \mathbb{F}_q^{k/2} \text{ et } |\mathbf{x}_1| = \tfrac{p}{2} \right\}$$

$$\mathscr{L}_2 := \left\{ \overline{\mathbf{P}}_2 \mathbf{x}_2^\top \in \mathbb{F}_q^{n-k} \ : \ \mathbf{x}_2 \in \mathbb{F}_q^{k/2} \text{ et } |\mathbf{x}_2| = \tfrac{p}{2} \right\}$$

Soient $\mathbf{y}_1 := \overline{\mathbf{s}} - \overline{\mathbf{P}}_1 \mathbf{x}_1^\top \in \mathscr{L}_1$ et $\mathbf{y}_2 := \overline{\mathbf{P}}_2 \mathbf{x}_2^\top \in \mathscr{L}_2$ tels que $\Delta(\mathbf{y}_1,\mathbf{y}_2) = w - p$. Le vecteur $\mathbf{e} \in \mathbb{F}_q^n$ tel que $\mathbf{e}_I = (\mathbf{x}_1,\mathbf{x}_2)$ et $\mathbf{e}_J^\top = \overline{\mathbf{s}} - \overline{\mathbf{P}}\mathbf{e}_I^\top$ est une solution du problème de décodage par syndrome. Or résoudre le problème 4.3.2 permet de trouver le couple $(\mathbf{y}_1,\mathbf{y}_2) \in \mathscr{L}_1 \times \mathscr{L}_2$.

Dans le cas où $q = 2$, May et Ozerov proposent une solution pour résoudre le problème 4.3.2 dans [MO15]. Nous présenterons leur méthode plus en détail dans le chapitre **??** section **??**. Nous proposerons ensuite une méthode utilisant des codes dans les chapitres **??** à **??**. Dans ce chapitre, nous supposons une boite noire qui résout le problème 4.3.2.

Finalement, le pseudo-code 4.3.2 résume la version de May et Ozerov de l'algorithme de Stern généralisée aux cas non-binaires. La fonction PRESQUECOLLISIONS $(\mathscr{L}_1,\ \mathscr{L}_2,\ w)$ est l'oracle qui trouve les w-presque-collisions dans $\mathscr{L}_1 \times \mathscr{L}_2$.

Algorithme 4.3.2 : La version de May et Ozerov de l'algorithme de Stern

Entrées : une matrice de parité $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$ d'un code $\mathcal{C}$;
un syndrome $\mathbf{s} \in \mathbb{F}_q^{n-k}$;
le poids maximal w du vecteur d'erreur recherché.
Paramètre : un entier pair $p \in [\![0, \min(w,k)]\!]$.
Sortie : $\mathbf{e} \in \mathbb{F}_q^n$ tel que $\mathbf{He}^\top = \mathbf{s}^\top$ et $|\mathbf{e}| = w$.

répéter indéfiniment
  choisir un ensemble $I \subseteq [\![1,n]\!]$ tel que $|I| = k$;
  $J \leftarrow [\![1,n]\!] \setminus I$;
  $\overline{\mathbf{s}} \leftarrow \mathbf{H}_J^{-1}\mathbf{s}^\top$; `/* si H_J n'est pas inversible revenir à l'étape 2 */`
  $\overline{\mathbf{P}} \leftarrow \mathbf{H}_J^{-1}\mathbf{H}_I$;
  extraire les matrices $\overline{\mathbf{P}}_1$ et $\overline{\mathbf{P}}_2$ comme décrit sur la figure 4.5;
  $\mathscr{L}_1 \leftarrow \left\{\overline{\mathbf{s}} - \overline{\mathbf{P}}_1\mathbf{x}_1^\top \in \mathbb{F}_q^{n-k} \ :\ \mathbf{x}_1 \in \mathbb{F}_q^{k/2} \text{ et } |\mathbf{x}_1| = \frac{p}{2}\right\}$;
  $\mathscr{L}_2 \leftarrow \left\{\overline{\mathbf{P}}_2\mathbf{x}_2^\top \in \mathbb{F}_q^{n-k} \ :\ \mathbf{x}_2 \in \mathbb{F}_q^{k/2} \text{ et } |\mathbf{x}_2| = \frac{p}{2}\right\}$;
  $\mathscr{L}^* \leftarrow$ PRESQUECOLLISIONS $(\mathscr{L}_1,\ \mathscr{L}_2,\ w-p)$;
  si $\mathscr{L}^* \neq \varnothing$ **alors**
    choisir un couple $\left(\overline{\mathbf{s}} - \overline{\mathbf{P}}_1\mathbf{x}_1^\top\ ,\ \overline{\mathbf{P}}_2\mathbf{x}_2^\top\right) \in \mathscr{L}^*$;
    $\mathbf{x} \leftarrow (\mathbf{x}_1, \mathbf{x}_2)$;
    construire $\mathbf{e}$ tel que $\mathbf{e}_I = \mathbf{x}$ et $\mathbf{e}_J^\top = \overline{\mathbf{s}} - \overline{\mathbf{P}}\mathbf{x}^\top$;
    retourner e ;
  finSi
finRépéter

Théorème 4.3.3. *Soit un code linéaire $[n,k]_q$ et soit un entier w tels que $k := 2\left\lfloor \frac{Rn}{2} \right\rfloor$ et $w := \lfloor \omega n \rfloor$ pour des constantes $R \in [0,1]$ et $\omega \in \left[0, h_q^{-1}(1-R)\right]$. Lorsque n tend vers l'infini, la complexité en temps de l'algorithme 4.3.2 paramétré avec $p := 2\left\lfloor \frac{\gamma n}{2} \right\rfloor$ est :*

$$T_{Stern+MO} = \tilde{O}(q^{\alpha n}) \tag{4.14}$$

où :

$$\alpha := \beta + h_q(\omega) - Rh_q\left(\tfrac{\gamma}{R}\right) - (1-R)h_q\left(\tfrac{\omega-\gamma}{1-R}\right) \tag{4.15}$$

avec $q^{\beta n}$ la complexité moyenne pour trouver les couples à distance $w-p$ dans deux listes chacune constituée de $\binom{k/2}{p/2}(q-1)^{\frac{p}{2}}$ vecteurs tirés uniformément dans $\mathbb{F}_q^{n-k}$. On choisit alors la valeur de $\gamma \in [0, \min(\omega, R)]$ qui minimise l'exposant α.

4.3.3 La méthode de Dumer

Le premier algorithme de Dumer. En 1989, Dumer propose une méthode de décodage des codes linéaires utilisant essentiellement une recherche de collisions [Dum89]. Cette première proposition de Dumer ne fait pas partie de la famille des décodages par ensemble d'information.

Pour simplifier les explications, nous supposons que n et w sont pairs. Dans la méthode de Dumer que nous élargissons ici aux cas non-binaires, il est alors choisi deux sous-ensembles complémentaires dans $[\![1, n]\!]$ de même taille :

$$I_1 \subseteq [\![1, n]\!] \text{ tel que } \#I_1 = \tfrac{n}{2} \tag{4.16}$$
$$I_2 := [\![1, n]\!] \setminus I_1 \tag{4.17}$$

On note $\mathbf{P}_1 := \mathbf{H}_{I_1}$ et $\mathbf{P}_2 := \mathbf{H}_{I_2}$. Une itération de la méthode de Dumer consiste alors à effectuer une recherche de collisions sur les deux listes suivantes :

$$\mathscr{L}_1 := \left\{ \mathbf{s} - \mathbf{P}_1 \mathbf{x}_1^\top \in \mathbb{F}_q^{n-k} \ : \ \mathbf{x}_1 \in \mathbb{F}_q^{\frac{n}{2}} \text{ et } |\mathbf{x}_1| = \tfrac{w}{2} \right\} \tag{4.18}$$
$$\mathscr{L}_2 := \left\{ \mathbf{P}_2 \mathbf{x}_2^\top \in \mathbb{F}_q^{n-k} \ : \ \mathbf{x}_2 \in \mathbb{F}_q^{\frac{n}{2}} \text{ et } |\mathbf{x}_2| = \tfrac{w}{2} \right\} \tag{4.19}$$

On a alors $L := \#\mathscr{L}_1 = \#\mathscr{L}_2 = \binom{n/2}{w/2}(q-1)^{\frac{w}{2}}$. En rangeant simplement les éléments de $\mathscr{L}_1$ et $\mathscr{L}_2$ dans une table de hachage de taille L, nous pouvons trouver les collisions de $\mathscr{L}_1 \times \mathscr{L}_2$ en un temps de l'ordre de $O\left(L + \frac{L^2}{q^{n-k}}\right)$.

La méthode de Dumer de 1989 ne nécessite qu'un nombre polynomial d'itérations puisque la probabilité de succès de chacune d'elle est :

$$\mathbb{P}_{\text{succ}} = \frac{\binom{n/2}{w/2}^2}{\binom{n}{w}} = \text{poly}\,(n) \tag{4.20}$$

Ainsi, le coût de la méthode de Dumer est de l'ordre de :

$$T_{\text{Dumer89}} = \tilde{O}\left(\binom{n/2}{w/2}(q-1)^{\frac{w}{2}} + \frac{\binom{n}{w}(q-1)^w}{q^{n-k}} \right) \tag{4.21}$$

Les codes poinçonnés. En 1991, Dumer améliore sa méthode dans [Dum91]. Son nouvel algorithme consiste essentiellement à décoder un code poinçonné de $\mathcal{C}$ avec sa méthode de 1989. Dans la suite, lorsque nous évoquerons la méthode de Dumer, nous parlerons de cette version de 1991.

Un code poinçonné de $\mathcal{C}$ est obtenu en *effaçant* des positions dans les mots de $\mathcal{C}$. Plus exactement :

Définition 4.3.4. *Soit $\mathcal{C}$ un code linéaire $[n, k]_q$ et soit une liste d'indices $J \subseteq [\![1, n]\!]$. On note $r := \#J$ et $I := [\![1, n]\!] \setminus J$. Le code poinçonné $\mathcal{C}'$ de $\mathcal{C}$ en J est :*

$$\mathcal{C}' := \left\{ \mathbf{x} \in \mathbb{F}_q^{n-r} \ : \ \exists \mathbf{c} \in \mathcal{C}, \mathbf{c}_I = \mathbf{x} \right\} \tag{4.22}$$

Proposition 4.3.5. *Soit $\mathcal{C}$ un code linéaire $[n, k]_q$ et soit une liste d'indices $J \subseteq [\![1, n]\!]$. On note $r := \#J$ et $I := [\![1, n]\!] \setminus J$. Si I contient un ensemble d'information du code $\mathcal{C}$, alors le code poinçonné $\mathcal{C}'$ de $\mathcal{C}$ en J est un code linéaire $[n - r, k]_q$.*

Démonstration de la proposition 4.3.5.

Soit $\mathbf{H}$, une matrice de parité du code $\mathcal{C}$. Puisque I contient un ensemble d'information, les colonnes de $\mathbf{H}$ indexées par J sont libres ; autrement dit, la matrice $\mathbf{H}_J$ est de rang r. Ainsi, il existe une unique matrice $\mathbf{A}$ telle que $\mathbf{A}\mathbf{H}_J$ soit la matrice identité sur ses r premières lignes et la matrice nulle sur ses $n - r - k$ dernières lignes. On vérifie alors facilement que la sous-matrice $\mathbf{H}'$ formée des $n - r - k$ dernières lignes de $\overline{\mathbf{P}} := \mathbf{A}\mathbf{H}_I$ est une matrice de parité du code poinçonné $\mathcal{C}'$. La figure 4.6 résume la construction de $\mathbf{H}'$. □

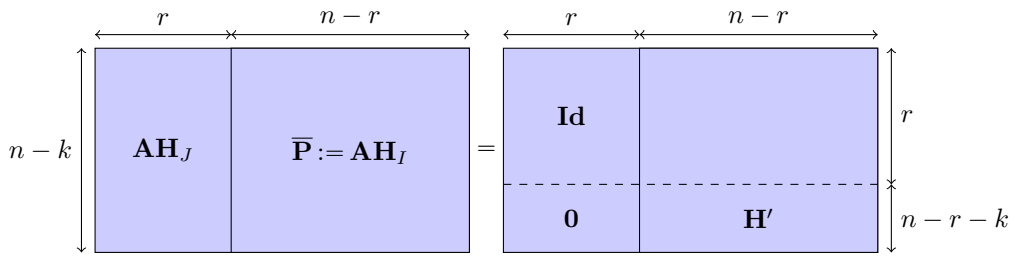


Figure 4.6 – Construction d'une matrice de parité $\mathbf{H}'$ du code poinçonné de $\mathcal{C}$ en J à partir d'une matrice de parité $\mathbf{H}$ de $\mathcal{C}$.

Le deuxième algorithme de Dumer. La méthode de décodage par syndrome proposée par Dumer en 1991 [Dum91] fait partie de la famille des décodages par ensemble d'information contrairement à celle qu'il a proposée en 1989. Ce nouvel algorithme est lui aussi itératif et chaque itération consiste à effectuer les instructions suivantes :

(1) sélectionner un ensemble $J \subseteq [\![1, n]\!]$ tel que $I := [\![1, n]\!] \setminus J$ contienne un ensemble d'information et tel que $\#I$ soit pair ;

(2) décoder le code poinçonné de $\mathcal{C}$ en J en utilisant la méthode de décodage par syndrome de Dumer de 1989 [Dum89] ;

(3) reconstruire les positions poinçonnées : puisque I est un ensemble d'information, chaque mot du code poinçonné est associé à un unique mot du code $\mathcal{C}$ que l'on retrouve en utilisant l'élimination gaussienne qui a été utilisée pour construire la matrice de parité du code poinçonné (cf. démonstration de la proposition 4.3.5) ;

(4) vérifier les poids des mots de code obtenus.

La complexité de l'algorithme de Dumer de 1991 est donnée par le théorème suivant :

Théorème 4.3.6. *Soit un code linéaire $[n, k]_q$ et soit un entier w tels que $k := \lfloor Rn \rfloor$ et $w := \lfloor \omega n \rfloor$ pour des constantes $R \in [0, 1]$ et $\omega \in \left[0, h_q^{-1}(1 - R)\right]$. Lorsque n tend vers l'infini, la complexité en temps de l'algorithme de Dumer après optimisation des paramètres est :*

$$T_{Dumer91} = \tilde{O}(q^{\alpha n}) \tag{4.23}$$

où :

$$\alpha := h_q(\omega) - 1 + \rho + R - \rho h_q\left(\tfrac{\omega - \gamma}{\rho}\right) \tag{4.24}$$

avec ρ tel que $2\left(1 - \frac{R}{1-\rho}\right) = h_q\left(\frac{\gamma}{1-\rho}\right)$ et $\gamma \in [0, 1 - \rho] \cap [\omega - \rho, \omega]$ qui minimise la valeur de α.

Les paramètres optimaux de l'algorithme de Dumer sont alors $r := n - 2\left\lceil \frac{(1-\rho)n}{2} \right\rceil$ et $p := 2\left\lfloor \frac{\gamma n}{2} \right\rfloor$.

Démonstration du théorème 4.3.6.

D'après l'équation (4.21), l'étape de décodage du code poinçonné a un coût de l'ordre de :

$$\tilde{O}\left(\binom{(n-r)/2}{p/2}(q-1)^{\frac{p}{2}} + \frac{\binom{n-r}{p}(q-1)^p}{q^{n-r-k}}\right)$$

En outre, le nombre d'itérations nécessaires pour trouver le vecteur d'erreur $\mathbf{e}$ est l'inverse de la probabilité de succès d'une itération qui est :

$$\mathbb{P}_{\text{succ}} = \frac{\binom{n-r}{p}\binom{r}{w-p}}{\binom{n}{w}} \tag{4.25}$$

Finalement, on a :

$$T_{\text{Dumer91}}(r,p) = \tilde{O}\left(\frac{\binom{n}{w}}{\binom{n-r}{p}\binom{r}{w-p}}\left(\binom{(n-r)/2}{p/2}(q-1)^{\frac{p}{2}} + \frac{\binom{n-r}{p}(q-1)^p}{q^{n-r-k}}\right)\right) \quad (4.26)$$

Nous choisissons alors r tel que $q^{n-r-k} = O\left(\binom{(n-r)/2}{p/2}(q-1)^{\frac{p}{2}}\right)$.

Nous terminons la preuve en appliquant la formule de Stirling. □

4.4 La technique des représentations

4.4.1 Définition et dénombrement des représentations

Dans la méthode de Dumer, supposons que nous avons bien choisi l'ensemble d'indices I de taille $n-r$; c'est-à-dire que le vecteur d'erreur $\mathbf{e}$ que nous recherchons est tel que $\mathbf{e}_I = (\mathbf{x}_1, \mathbf{x}_2) := (\mathbf{x}_1, \mathbf{0}) + (\mathbf{0}, \mathbf{x}_2)$ où $\mathbf{x}_1$ et $\mathbf{x}_2$ sont deux vecteurs de $\mathbb{F}_q^{\frac{n-r}{2}}$ et $|\mathbf{x}_1| = |\mathbf{x}_2| = \frac{p}{2}$. Remarquons alors qu'il n'existe qu'une seule et unique façon de décomposer le vecteur $\mathbf{e}_I$ comme somme de deux vecteurs de poids $\frac{p}{2}$ à supports respectivement inclus dans $[\![1, \frac{n-r}{2}]\!]$ et $[\![\frac{n-r}{2}+1, n-r]\!]$.

En nous inspirant de la technique des représentations de Howgrave-Graham et Joux dans [HJ10], nous allons démultiplier le nombre de façons de représenter le vecteur $\mathbf{e}_I$ comme somme de deux vecteurs.

Commençons par définir formellement la notion de *représentation* :

Définition 4.4.1. *Soit $\mathbf{x} \in \mathbb{F}_q^n$ de poids w et soit $\overline{w} \in [\![\frac{w}{2}, n]\!]$. Un couple $(\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{F}_q^n \times \mathbb{F}_q^n$ est une* $\overline{w}$-représentation *de $\mathbf{x}$ lorsque $|\mathbf{x}_1| = |\mathbf{x}_2| = \overline{w}$ et $\mathbf{x} = \mathbf{x}_1 + \mathbf{x}_2$.*

La proposition suivante dénombre alors le nombre de façons de représenter un vecteur $\mathbf{x} \in \mathbb{F}_q^n$ de poids w comme somme de deux vecteurs de poids $\overline{w}$:

Théorème 4.4.2. *Soit $\mathbf{x} \in \mathbb{F}_q^n$ de poids w et soit $\overline{w} \in [\![\frac{w}{2}, r]\!]$. On suppose que $w := \lfloor \omega n \rfloor$ et $\overline{w} := \lfloor \overline{\omega} n \rfloor$ pour des constantes $\omega \in [0,1]$ et $\overline{\omega} \in [\frac{\omega}{2}, 1]$.*

Le nombre de $\overline{w}$-représentations *du vecteur $\mathbf{x}$ est :*

$$R_q(n, w, \overline{w}) := \tilde{\Theta}\left(q^{\alpha n}\right) \quad (4.27)$$

où :

$$\begin{aligned} \alpha \quad := \quad & (1-\omega) h_q\left(\frac{\gamma}{1-\omega}\right) + \omega h_q\left(\frac{\overline{\omega}-\gamma}{\omega}\right) + (\overline{\omega}-\gamma) h_q\left(2 - \frac{\omega}{\overline{\omega}-\gamma}\right) \\ & + (2(\overline{\omega}-\gamma)-\omega)\log_q(q-2) - (3(\overline{\omega}-\gamma)-\omega)\log_q(q-1) \end{aligned} \quad (4.28)$$

avec $\gamma \in [0, 1-\omega] \cap \left[\overline{\omega}-\omega, \overline{\omega}-\frac{\omega}{2}\right]$ qui maximise α et qui peut être calculé analytiquement.

Démonstration du théorème 4.4.2.

Remarquons tout d'abord que dénombrer les $\overline{w}$-représentations de $\mathbf{x}$ revient à compter le nombre d'éléments sur la sphère de rayon $\overline{w}$ centrée en $\mathbf{x}$ que nous noterons $\mathcal{S}(\mathbf{x}, \overline{w})$:

$$\mathcal{S}(\mathbf{x}, \overline{w}) := \left\{\mathbf{x}_1 \in \mathbb{F}_q^n : \Delta(\mathbf{x}, \mathbf{x}_1) = \overline{w}\right\} \quad (4.29)$$

Soit $\mathbf{x}_1 \in \mathcal{S}(\mathbf{x}, \overline{w})$. Posons alors $a := \#\operatorname{supp}(\mathbf{x}_1) \setminus \operatorname{supp}(\mathbf{x})$. Nous représentons alors la situation à l'aide de la figure 4.7.

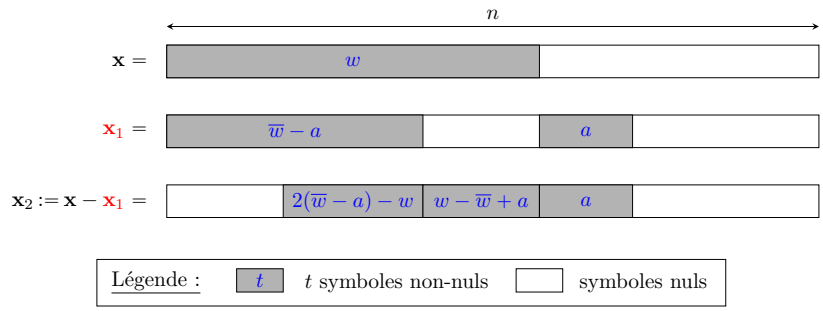


Figure 4.7 – Une configuration possible du vecteur $\mathbf{x}_1 \in \mathcal{S}(\mathbf{x}, \overline{w})$.

Tout d'abord, remarquons que a peut prendre n'importe quelle valeur entière telle que :

$$\begin{cases} 0 \leqslant & a & \leqslant n-w \\ 0 \leqslant & \overline{w}-a & \leqslant w \\ 0 \leqslant & 2(\overline{w}-a)-w & \leqslant \overline{w}-a \end{cases} \tag{4.30}$$

C'est-à-dire :

$$a \in [\![0, n-w]\!] \cap \left[\!\left[\overline{w}-w, \overline{w}-\frac{w}{2}\right]\!\right] \tag{4.31}$$

On a alors :

$$\#\left(\mathcal{S}(\mathbf{x}, \overline{w})\right) = \sum_{a=\max(0,\overline{w}-w)}^{\min\left(n-w, \overline{w}-\frac{w}{2}\right)} \binom{n-w}{a}\binom{w}{\overline{w}-a}\binom{\overline{w}-a}{2(\overline{w}-a)-w}(q-1)^a(q-2)^{2(\overline{w}-a)-w} \tag{4.32}$$

La formule de Stirling (cf. théorème 1.3.6) nous donne une version asymptotique de cette formule :

$$\#\left(\mathcal{S}(\mathbf{x}, \overline{w})\right) = \tilde{\Theta}\left(\sup_{\gamma=\max(0,\overline{w}-\omega)}^{\min\left(1-\omega, \overline{\omega}-\frac{\omega}{2}\right)} \left(q^{f(\gamma)n}\right)\right) \tag{4.33}$$

où f est définie par :

$$\begin{aligned} f(\gamma) \quad := \quad & (1-\omega)h_q\left(\tfrac{\gamma}{1-\omega}\right) + \omega h_q\left(\tfrac{\overline{\omega}-\gamma}{\omega}\right) + (\overline{\omega}-\gamma)h_q\left(2-\tfrac{\omega}{\overline{\omega}-\gamma}\right) \\ & +(2(\overline{\omega}-\gamma)-\omega)\log_q(q-2) - (3(\overline{\omega}-\gamma)-\omega)\log_q(q-1) \end{aligned} \tag{4.34}$$

avec $\gamma \in [0, 1-\omega] \cap \left[\overline{\omega}-\omega, \overline{\omega}-\frac{\omega}{2}\right]$. Une analyse de la dérivée de f nous permet de montrer que son maximum est atteint sur une des bornes de son domaine de définition ou bien en une solution réelle dans $\left[\max\left(0, \overline{\omega}-\omega\right), \min\left(1-\omega, \overline{\omega}-\frac{\omega}{2}\right)\right]$ de l'équation :

$$(q-1)(1-\omega-x)(2(\overline{\omega}-x)-\omega)^2 = x(q-2)^2(\omega-\overline{\omega}+x)^2 \tag{4.35}$$

On peut alors utiliser la méthode de Cardan pour déterminer les solutions de l'équation (4.35). □

Le cas binaire est un cas un peu particulier. Soient $\mathbf{x}$, $\mathbf{x}_1$ et $\mathbf{x}_2$ des vecteurs de $\mathbb{F}_2^n$ tels que $\mathbf{x} = \mathbf{x}_1 + \mathbf{x}_2$. Les supports de ces trois vecteurs binaires ont nécessairement une intersection vide. Si $\mathbf{x}_1$ et $\mathbf{x}_2$ sont de même poids $\overline{w}$ et $\mathbf{x}$ est de poids w, alors $a := \#\operatorname{supp}(\mathbf{x}_1) \setminus \operatorname{supp}(\mathbf{x})$ vaut nécessairement $\overline{w} - \frac{w}{2}$. Le nombre de $\overline{w}$-représentations de $\mathbf{x}$ est alors :

$$R_2(n, w, \overline{w}) = \binom{n-w}{\overline{w}-\frac{w}{2}}\binom{w}{\frac{w}{2}} = \tilde{\Theta}\left(2^{n\left((1-\omega)h_2\left(\frac{\overline{\omega}-\frac{\omega}{2}}{1-\omega}\right)+\omega\right)}\right) \tag{4.36}$$

On retrouve en fait exactement le théorème 4.4.2 où a vaut la seule valeur qui n'annule pas l'équation (4.27) ; à savoir $a = \overline{w} - \frac{w}{2}$. En outre, remarquons dans le cas binaire que

si $\overline{w} > n - \frac{w}{2}$ alors il n'existe aucune $\overline{w}$-représentation de $\mathbf{x}$. Sur $\mathbb{F}_2^n$, il faut donc choisir $\overline{w} \in [\![\frac{w}{2}, n - \frac{w}{2}]\!]$.

4.4.2 La méthode BJMM

Les différentes améliorations des méthodes de décodage ISD jusqu'à celle de Dumer ont permis d'augmenter le poids des vecteurs d'erreur à décoder. Cependant, ce poids reste encore un facteur limitant. La méthode de décodage par ensemble d'information de Becker, Joux, May et Meurer connue sous l'acronyme BJMM utilise la technique des représentations pour accélérer les techniques de décodage par ensemble d'information et ainsi atteindre des vecteurs d'erreur de poids beaucoup plus élevés [BJMM12].

La version de BJMM décrite ici est une version améliorée de la version originale de 2012. Nous tenons notamment compte des modifications de l'article [MO15] de May et Ozerov pour intégrer une recherche de presque-collisions dans le processus de décodage. Nous appliquons aussi les améliorations de l'article [BM17b] de Both et May où le nombre d'itérations de la technique des représentations est rendue paramétrable. En outre, La méthode BJMM a initialement été imaginée pour décoder des codes binaires. Nous étendons ici ses applications aux instances non-binaires [GKH17].

Comme pour la méthode de Dumer, la version de May et Ozerov de la méthode BJMM [MO15, BM17b] commence par sélectionner un ensemble d'indices I de taille $k+\ell$ contenant un ensemble d'information (par rapport à la méthode de Dumer, on pose $\ell := n-k-r$ où r est le nombre de positions poinçonnées). L'ensemble de redondance $[\![1,n]\!] \setminus I$ est alors noté J. D'autre part, on pose $L := [\![1,\ell]\!]$ et $\overline{L} := [\![\ell+1, n-k]\!]$. Une élimination gaussienne permet alors de transformer la matrice de parité $\mathbf{H}$ pour produire la matrice décrite dans la figure 4.8.

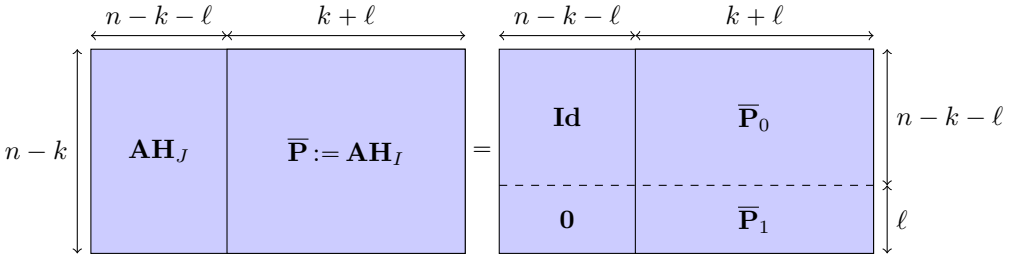


Figure 4.8 – Découpage de $[\mathbf{AH}_J|\mathbf{AH}_I]$ pour la méthode BJMM.

Dans la figure 4.8, $\mathbf{A}$ est la matrice de passage de l'élimination gaussienne. C'est une matrice carrée de taille $(n-k) \times (n-k)$. Soit $\overline{\mathbf{s}} := \mathbf{As}^\top$.

Supposons que le vecteur d'erreur $\mathbf{e}$ soit tel que $|\mathbf{e}_I| = p$. Cela arrive avec une probabilité $\mathbb{P}_{\text{succ}} := \frac{\binom{n-k-\ell}{w-p}\binom{k+\ell}{p}}{\binom{n}{w}}$ lorsque I est tiré uniformément parmi les ensembles d'indices de taille $k+\ell$. Soit $p_1 \in [\![\frac{p}{2}, k+\ell]\!]$ un paramètre à optimiser. Toutes les p_1-représentations $(\mathbf{x}_1, \mathbf{x}_2)$ de $\mathbf{e}_I$ vérifient les deux équations suivantes :

$$\overline{\mathbf{s}}_L - \overline{\mathbf{P}}_1 \mathbf{x}_1^\top = \overline{\mathbf{P}}_1 \mathbf{x}_2^\top \tag{4.37}$$

$$\Delta\left(\overline{\mathbf{s}}_{\overline{L}} - \overline{\mathbf{P}}_0 \mathbf{x}_1^\top, \overline{\mathbf{P}}_0 \mathbf{x}_2^\top\right) = w - p \tag{4.38}$$

Soit $\mathscr{L}^*$, l'ensemble des couples de vecteurs de poids p_1 vérifiant l'équation (4.37) :

$$\mathscr{L}^* := \left\{(\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{F}_q^{k+\ell} \times \mathbb{F}_q^{k+\ell} \text{ tel que } |\mathbf{x}_1| = |\mathbf{x}_2| = p_1 \text{ et } \overline{\mathbf{s}}_L - \overline{\mathbf{P}}_1 \mathbf{x}_1^\top = \overline{\mathbf{P}}_1 \mathbf{x}_2^\top\right\} \tag{4.39}$$

Seule une portion de $\frac{1}{R_q(k+\ell,p,p_1)}$ de $\mathscr{L}^*$ suffit pour que l'espérance du nombre de représentations de $\mathbf{x}$ encore présents dans la liste soit $\Omega(1)$.

Nous construisons alors les listes $\mathscr{L}_1$ et $\mathscr{L}_2$ suivantes :

$$\mathscr{L}_1 := \left\{ \bar{\mathbf{s}}_L - \overline{\mathbf{P}}_1 \mathbf{x}_1^\top \ : \ \mathbf{x}_1 \in \mathbb{F}_q^{k+\ell} \text{ et } |\mathbf{x}_1| = p_1 \text{ et } \left(\bar{\mathbf{s}}_L - \overline{\mathbf{P}}_1 \mathbf{x}_1^\top\right)_{[r_1]} = \mathbf{t}_1 \right\} \quad (4.40)$$

$$\mathscr{L}_2 := \left\{ \overline{\mathbf{P}}_1 \mathbf{x}_2^\top \ : \ \mathbf{x}_2 \in \mathbb{F}_q^{k+\ell} \text{ et } |\mathbf{x}_2| = p_1 \text{ et } \left(\overline{\mathbf{P}}_1 \mathbf{x}_2^\top\right)_{[\mathbf{r_1}]} = \mathbf{t}_1 \right\} \quad (4.41)$$

avec :

$r_1 = \lfloor \log_q(R_q(k+\ell,p,p_1)) \rfloor$;
$\mathbf{t}_1$ tiré aléatoirement dans $\mathbb{F}_q^{r_1}$;
$[r_1]$ un ensemble d'indices de taille r_1.

On note $\mathscr{L}_0$ l'ensemble des couples $(\mathbf{x}_1, \mathbf{x}_2)$ tels que $\bar{\mathbf{s}}_L - \mathbf{P}_1 \mathbf{x}_1{}^\top = \mathbf{P}\mathbf{x}_2{}^\top \in \mathscr{L}_1 \cap \mathscr{L}_2$. La liste produite $\mathscr{L}_0$ est bien une sous-liste de $\mathscr{L}^*$ de taille typique :

$$S_1 := \frac{\binom{k+\ell}{p_1}(q-1)^{p_1}}{q^{r_1}} = \frac{\#\mathscr{L}^*}{R_q(k+\ell,p,p_1)} \quad (4.42)$$

L'utilisation de tables de hachage permet alors de déterminer $\mathscr{L}_0$ en un temps de l'ordre de :

$$T_1 := O\left(S_1 + \frac{S_1^2}{q^{\ell - r_1}}\right) \quad (4.43)$$

Enfin, après avoir produit la liste $\mathscr{L}_0$, il reste à vérifier que les couples obtenus satisfassent bien la condition donnée par l'équation (4.38).

Itération de la technique des représentations. La méthode BJMM est en fait plus complète que la description que nous venons de donner. Remarquons tout d'abord que le processus décrit dans le paragraphe précédent peut être appliqué récursivement pour produire les listes $\mathscr{L}_1$ et $\mathscr{L}_2$. Ce procédé récursif peut être généralisé à une profondeur de récursion m arbitraire. On pose alors les m paramètres $p^{(m)} := p$, $p^{(m-1)}$, $\cdots$, $p^{(1)} := 2p^{(0)}$ que nous devront optimiser par la suite.

L'algorithme BJMM calcule des listes récursivement. Nous rangeons celles-ci dans un arbre binaire de profondeur m. Nous comptons les étages de l'arbre depuis les feuilles ; c'est-à-dire que les feuilles sont à l'étage 0 et la racine à l'étage m. On note $\mathscr{L}_1^{(m)}$ la liste à la racine et pour tout $i \in [\![1, m]\!]$ et $j \in [\![1, 2^{m-i}]\!]$ on note respectivement $\mathscr{L}_{2j-1}^{(i-1)}$ et $\mathscr{L}_{2j}^{(i-1)}$ les fils de gauche et droite de la liste $\mathscr{L}_j^{(i)}$.

Les listes $\mathscr{L}_1^{(0)}, \cdots, \mathscr{L}_{2^m}^{(0)}$ aux feuilles de l'arbre (donc à l'étage $i = 0$) sont produites différemment des autres : elles sont obtenues par un découpage particulier de l'ensemble d'information $I = I_1 \cup I_2$, où $\sharp I_1 = \left\lfloor \frac{k+\ell}{2} \right\rfloor$ et $\sharp I_2 = \left\lceil \frac{k+\ell}{2} \right\rceil$ (on retrouve en fait un découpage similaire à celui que l'on a déjà vu dans l'algorithme de Dumer). Ces listes sont alors définies ainsi :

$$\mathscr{L}_{2j}^{(0)} = \left\{ \mathbf{x} \in \mathbb{F}_q^n \ : \ \mathbf{x}_J = \overline{\mathbf{P}}_0 \mathbf{x}_I^\top \ , \ |\mathbf{x}_{I_1}| = p^{(0)} \text{ et } |\mathbf{x}_{I_2}| = 0 \right\} \quad \forall j \in [\![1, 2^m]\!] \quad (4.44)$$

$$\mathscr{L}_{2j-1}^{(0)} = \left\{ \mathbf{x} \in \mathbb{F}_q^n \ : \ \mathbf{x}_J = \overline{\mathbf{P}}_0 \mathbf{x}_I^\top \ , \ |\mathbf{x}_{I_1}| = 0 \text{ et } |\mathbf{x}_{I_2}| = p^{(0)} \right\} \quad \forall j \in [\![1, 2^m]\!] \quad (4.45)$$

Les autres listes sont quant à elles définies récursivement. Plus précisément, nous avons pour tout $i \in [\![1, m-1]\!]$ et $j \in [\![1, 2^{m-i}]\!]$:

$$\mathscr{L}_j^{(i)} := \left\{ \mathbf{x} \in \mathbb{F}_q^n \ : \ \mathbf{x}_I = \mathbf{x}_1 + \mathbf{x}_2 \ , \ \mathbf{x}_1 \in \mathscr{L}_1^{(i-1)} \ , \ \mathbf{x}_2 \in \mathscr{L}_2^{(i-1)} \ , \ |\mathbf{x}_I| = p^{(i)} \text{ et } \left(\overline{\mathbf{P}}_1 \mathbf{x}_I^\top\right)_{[r_i]} = \mathbf{t}_j^{(i)} \right\} \quad (4.46)$$

avec :

(a) $r_{m-1} = \ell$ et $[r_{m-1}] = L$ (sans perte de généralité, nous supposons $L = [\![1, \ell]\!]$) ;

(b) pour tout $i \in [\![1, m-1]\!]$, $r_i = \left\lceil \log_q(R_q(k+\ell, p^{(i+1)}, p^{(i)})) \right\rceil$ et $[r_i] \subseteq [r_{i+1}]$ avec $\#[r_i] = r_i$;

(c) $\mathbf{t}_1^{(m-1)} := \overline{\mathbf{s}}_L - \mathbf{t}_2^{(m-1)}$ est tiré uniformément dans $\mathbb{F}_q^\ell$;

(d) pour tout $i \in [\![2, m-1]\!]$ et tout $j \in [\![1, 2^{m-i}]\!]$, $\mathbf{t}_{2j-1}^{(i-1)}$ est tiré uniformément dans $\mathbb{F}_q^{r_{i-1}}$ et $\mathbf{t}_{2j}^{(i-1)} := \left(\mathbf{t}_j^{(i)}\right)_{[r_{i-1}]} - \mathbf{t}_{2j-1}^{(i-1)}$.

Notons que pour tout $i \in [\![1, m-1]\!]$, il nous faut choisir $p^{(i+1)}$ et $p^{(i)}$ de telle sorte que $R_q\left(k+\ell, p^{(i+1)}, p^{(i)}\right) \geqslant 1$. C'est pourquoi les paramètres doivent respecter les contraintes suivantes :

– si $q = 2$:

$$\frac{p^{(i+1)}}{2} \leqslant p^{(i)} \leqslant k + \ell - \frac{p^{(i+1)}}{2} \qquad \text{pour tout } i \in [\![1, m-1]\!] \tag{4.47}$$

– si $q > 2$:

$$\frac{p^{(i+1)}}{2} \leqslant p^{(i)} \leqslant k + \ell \qquad \text{pour tout } i \in [\![1, m-1]\!] \tag{4.48}$$

Ce qui est intéressant avec ces listes, c'est qu'elles peuvent être construites relativement efficacement grâce à des recherches de collisions. En effet, pour tout $i \in [\![1, m-1]\!]$ et $j \in [\![1, 2^{m-i}]\!]$, une recherche de collisions sur les listes $\mathscr{L}_{2j-1}^{i-1}$ et $\mathscr{L}_{2j}^{i-1}$ permet de produire la liste temporaire :

$$\begin{aligned} \overline{\mathscr{L}}_j^{(i)} \quad := \quad \Big\{ \mathbf{x} \in \mathbb{F}_q^n \ : \ & \mathbf{x}_I = \mathbf{x}_1 + \mathbf{x}_2 \ , \\ & \mathbf{x}_1 \in \mathscr{L}_1^{(i-1)} \ , \ \mathbf{x}_2 \in \mathscr{L}_2^{(i-1)} \text{ et } \left(\overline{\mathbf{P}}_1 \mathbf{x}_I^\top\right)_{[r_i]} = \mathbf{t}_j^{(i)} \Big\} \end{aligned} \tag{4.49}$$

Une simple opération de filtrage permet finalement d'obtenir la liste $\mathscr{L}_j^{(i)}$ à partir de $\overline{\mathscr{L}}_j^{(i)}$.

À ce stade, il nous reste à décrire la construction de la liste à la racine. Dans [MO15], il est proposé une construction qui permet de bénéficier d'une recherche de presque-collisions (cf. problème 4.3.2) à la dernière étape de l'algorithme plutôt que d'effectuer une recherche exhaustive comme dans [BJMM12]. Ainsi, une recherche de presque-collisions sur les positions J des listes $\mathscr{L}_1^{m-1}$ et $\mathscr{L}_2^{m-1}$ permet de construire la liste temporaire suivante :

$$\begin{aligned} \overline{\mathscr{L}}_1^{(m)} \quad := \quad \Big\{ \mathbf{x} \in \mathbb{F}_q^n \ : \ & \mathbf{x}_I = \mathbf{x}_1 + \mathbf{x}_2 \ , \ \mathbf{x}_1 \in \mathscr{L}_1^{(m-1)} \ , \\ & \mathbf{x}_2 \in \mathscr{L}_2^{(m-1)} \ , \ \text{et } \left|\overline{\mathbf{P}}_0 \mathbf{x}_I^\top - \overline{\mathbf{s}}_{\overline{L}}\right| = w - p \Big\} \end{aligned} \tag{4.50}$$

puis un simple filtrage sur les positions I nous donne la liste à la racine :

$$\mathscr{L}_1^{(m)} := \left\{ \mathbf{x} \in \overline{\mathscr{L}}_1^{(m)} \ : \ |\mathbf{x}_I| = p \right\} \tag{4.51}$$

Remarque 4.4.1. Dans [BJMM12], la liste à la racine est produite de la même façon que les autres listes. Des solutions du problème de décodage sont alors recherchées exhaustivement dans cette liste. Pour retrouver la construction des listes de l'algorithme BJMM d'origine, il suffit de remplacer $m-1$ par m dans les critères (a), (b), (c) et (d) donnés plus haut.

Finalement, la version de [MO15] de l'algorithme BJMM est décrite par le pseudo-code 4.4.1.

Algorithme 4.4.1 : La version de May et Ozerov de l'algorithme BJMM de profondeur m

Entrées : une matrice de parité $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$ d'un code $\mathcal{C}$;
un syndrome $\mathbf{s} \in \mathbb{F}_q^{n-k}$;
le poids maximal w du vecteur d'erreur recherché.
Paramètres : des entiers $p^{(m)} := p$, $p^{(m-1)}$, $\cdots$, $p^{(1)} := 2p^{(0)}$ vérifiant la contrainte (4.48) (ou (4.47)).
Sortie : $\mathbf{e} \in \mathbb{F}_q^n$ tel que $\mathbf{H}\mathbf{e}^\top = \mathbf{s}^\top$ et $|\mathbf{e}| = w$.

1 calculer $r_1, \cdots, r_{m-1} := \ell$ tels que $\forall i \in [\![1, m-1]\!]$, $r_i = \left\lceil \log_q(R_q(k+\ell, p^{(i+1)}, p^{(i)})) \right\rceil$;
2 **répéter**
3 choisir un ensemble $I \subseteq [\![1, n]\!]$ tel que $|I| = k + \ell$;
4 $J \leftarrow [\![1, n]\!] \setminus I$ calculer $\mathbf{A} \in \mathbb{F}_q^{(n-k)\times(n-k)}$ telle que $[\mathbf{A}|\mathbf{H}_J] = \begin{bmatrix} \mathbf{Id} \\ \hline \mathbf{0} \end{bmatrix}$; `/* si` $\mathbf{H}_J$ `est de rang` $< n-k-\ell$`, revenir à l'étape 3 */`
5 $\overline{\mathbf{s}} \leftarrow \mathbf{A}\mathbf{s}^\top$ $\overline{\mathbf{P}} \leftarrow \mathbf{A}\mathbf{H}_I$;
6 produire la liste $\mathscr{L}_1^{(m)}$ comme décrit par l'équation (4.51) ;
7 **tant que** $\mathscr{L}_1^{(m)} \neq \emptyset$;
8 **retourner** $\mathbf{e} \in \mathscr{L}_1^{(m)}$;

4.4.3 Complexité de l'algorithme

Le choix des paramètres r_i nous garantit qu'une itération de l'algorithme BJMM-MO sera fructueuse dès lors qu'une solution particulière $\mathbf{e}$ du problème de décodage que nous considérons a la même distribution de poids que les éléments de la liste $\mathscr{L}_1^{(m)}$; c'est-à-dire dès lors que $|\mathbf{e}_I| = p$ et $|\mathbf{e}_J| = w - p$ où I et J sont les ensembles d'indices choisis au début de l'itération. Ainsi, la probabilité de succès d'une itération de l'algorithme BJMM-MO est :

$$\mathbb{P}_{\text{succ}} := \frac{\binom{k+\ell}{p}\binom{n-k-\ell}{w-p}}{\binom{n}{w}} \tag{4.52}$$

Il nous faudra donc répéter en moyenne $\frac{1}{\mathbb{P}_{\text{succ}}}$ fois la procédure de l'algorithme 4.4.1 pour que celui-ci s'arrête.

De plus, les listes construites aux feuilles de l'arbre (à l'étage $i = 0$) sont de taille :

$$S^{(0)} := \binom{\frac{k+\ell}{2}}{p^{(0)}} (q-1)^{p^{(0)}} \tag{4.53}$$

et le coût pour produire l'une d'entre elles est :

$$T^{(0)} = S^{(0)} \tag{4.54}$$

Pour les étages $i \in [\![1, m-1]\!]$, les listes sont en moyenne de taille :

$$S^{(i)} := \frac{\binom{k+\ell}{p^{(i)}}(q-1)^{p^{(i)}}}{q^{r_i}} \tag{4.55}$$

et le coût pour produire l'une des listes de l'étage i est :

$$T^{(i)} := O\left(S^{(i-1)} + \frac{\left(S^{(i-1)}\right)^2}{q^{r_i - r_{i-1}}}\right) \tag{4.56}$$

(avec $r_0 := 0$).

Enfin, la liste à la racine est produite par une recherche de presque-collisions. Dans [MO15, BM17b, GKH17], il est proposé d'utiliser la méthode de May et Ozerov [MO15] dans le cas binaire ou celle de Hirose [Hir16] dans le cas non-binaire pour effectuer cette étape. Nous détaillerons l'algorithme May-Ozerov dans le chapitre **??** section **??** et dans les chapitres **??** à **??**, nous proposerons une méthode plus efficace pour résoudre le problème des presque-collisions. Pour le moment, nous supposons un oracle qui trouve les d-presque-collisions dans une liste constituée de L éléments tirés uniformément dans $\mathbb{F}_q^n$ en un temps $\beta(q, n, L, d)$. Ainsi, le coût pour produire la liste à la racine est :

$$T^{(m)} := \beta(q, n-k-\ell, S^{(m-1)}, w-p) \tag{4.57}$$

Finalement, la complexité en temps de la méthode BJMM est :

$$T_{\text{BJMM}} = \frac{1}{\mathbb{P}_{\text{succ}}} \sum_{i=0}^{m} 2^{m-i} T^{(i)} = O\left(\frac{\binom{n}{w} \cdot \max\limits_{i \in [\![0,m]\!]} \left(T^{(i)} \right)}{\binom{k+\ell}{p}\binom{n-k-\ell}{w-p}} \right) \tag{4.58}$$

et la complexité mémoire est :

$$S_{\text{BJMM}} = \sum_{i=0}^{m-1} 2^{m-i} S^{(i)} = O\left(\max_{i \in [\![0,m-1]\!]} \left(S^{(i)} \right) \right) \tag{4.59}$$

La formule de Stirling permet d'avoir une version asymptotique de ces formules.

Pour finir, le logiciel CaWoF [Can16] développé par Canto-Torres permet de donner l'exposant asymptotique optimal de l'algorithme de décodage binaire BJMM original [BJMM12]. Les améliorations de [MO15] et de [BM17b] peuvent être prises en compte en utilisant le code de Both (`https://github.com/LeifBoth/Decoding-LPN`) qu'il a publié dans le cadre de l'article [BM18]. Ici encore, seul le cas binaire est considéré mais de légères modifications permettent de traiter également les cas non-binaires.

4.5 La méthode de Both et May sur $\mathbb{F}_q$

Dans la version de May et Ozerov de l'algorithme BJMM, toutes les listes sauf une sont obtenues par une recherche de collisions. La dernière liste est quant à elle produite en résolvant un problème de presque-collisions. Nous aimerions toutefois exploiter un peu plus la recherche de presque-collisions pour effectuer un décodage par syndrome plus efficace. C'est le défi qu'ont réussi à relever Both et May dans [BM18] pour des codes binaires. Dans cette section, nous allons généraliser leur algorithme aux espaces de Hamming non-binaires.

4.5.1 Description de l'algorithme

Comme pour l'algorithme de Stern, la méthode de Both et May commence en choisissant un ensemble d'information I et son complémentaire $J := [\![1, n]\!] \setminus I$. Nous supposons alors que $|\mathbf{e}_I| = p$ où p est un poids fixé (relativement faible) entre 0 et k. Puisque nous avons supposé que I est un ensemble d'information du code de matrice de parité $\mathbf{H}$, nous pouvons inverser $\mathbf{H}_J$. Nous posons alors :

$$\overline{\mathbf{P}} := \mathbf{H}_J^{-1}\mathbf{H}_I \in \mathbb{F}_q^{(n-k)\times k} \tag{4.60}$$

$$\overline{\mathbf{s}} := \mathbf{H}_J^{-1}\mathbf{s}^\top \in \mathbb{F}_q^{n-k} \tag{4.61}$$

Remarquons que nous avons :

$$\begin{aligned}\overline{\mathbf{s}} &= \mathbf{H}_J^{-1}\mathbf{s}^\top \\ &= \mathbf{H}_J^{-1}\left(\mathbf{H}_I\mathbf{e}_I^\top + \mathbf{H}_J\mathbf{e}_J^\top\right) \\ &= \overline{\mathbf{P}}\mathbf{e}_I^\top + \mathbf{e}_J^\top.\end{aligned}$$

Ainsi, le vecteur d'erreur $\mathbf{e}$ est tel que $\mathbf{e}_J^\top = \overline{\mathbf{s}} - \overline{\mathbf{P}}\mathbf{e}_I^\top$. Puisque nous avons supposé $|\mathbf{e}_I| = p$, nous avons nécessairement $|\mathbf{e}_J| = w - p$. Dans l'algorithme de Both et May, nous recherchons alors un vecteur $\mathbf{e}_I \in \mathbb{F}_q^k$ tel que nous avons simultanément :

$$|\mathbf{e}_I| = p \tag{4.62}$$
$$\left|\overline{\mathbf{s}} - \overline{\mathbf{P}}\mathbf{e}_I^\top\right| = w - p \tag{4.63}$$

Dans la sous-section 4.3.2, nous avons déjà vu la procédure de Stern qui permet de trouver de tels vecteurs $\mathbf{e}_I$. En appliquant récursivement des recherches de presque-collisions dans des listes judicieusement construites, Both et May proposent une méthode beaucoup plus efficace que celle de Stern dans [BM18].

L'idée de Both et May est de *séparer* l'erreur en deux vecteurs tels que $\mathbf{e} = \mathbf{e}_1 + \mathbf{e}_2$ avec $|\mathbf{e}_1| = |\mathbf{e}_2| = w'$. Nous allons alors profiter de la multiplicité des w'-représentations de $\mathbf{e}$ (cf. section 4.4.1) lorsque le paramètre w' est judicieusement choisi. De plus, comme dans [MMT11, BJMM12, MO15, BM17b], les vecteurs $\mathbf{e}_1$ et $\mathbf{e}_2$ sont obtenus en appliquant récursivement la procédure.

Contrairement à la méthode BJMM que nous avons vu précédemment, dans l'algorithme Both-May, chacune des étapes récursives permettant de construire le vecteur $\mathbf{e}$ utilise une recherche de presque-collisions (et non une recherche de collisions exactes). Pour expliquer plus précisément cette procédure récursive, nous devons introduire certaines notations. Tout d'abord, l'algorithme Both-May dépend d'un paramètre m qui représente la profondeur de la récursion. L'ensemble J des positions de redondance est alors divisé en m morceaux $J_1, \ldots, J_m$ de tailles respectives $r_1, \cdots, r_m$. Nous avons aussi besoin de paramètres supplémentaires $p^{(0)}, \ldots, p^{(m)}$, $\left\{w_i^{(\ell)} \ : \ \ell \in [\![1, m]\!] \text{ et } i \in [\![1, \ell]\!]\right\}$. Tous ces paramètres devront être optimisés en respectant les contraintes suivantes :

$$\sum_{j=1}^m r_j = n - k \tag{4.64}$$
$$\sum_{i=1}^m w_i^{(m)} = w - p \tag{4.65}$$
$$p^{(1)} = 2p^{(0)} \tag{4.66}$$
$$\frac{p^{(\ell)}}{2} \leqslant p^{(\ell-1)} \leqslant k \qquad \text{pour tout } \ell \in [\![2, m]\!] \tag{4.67}$$
$$0 \leqslant p^{(m)} = p \leqslant k \tag{4.68}$$
$$\frac{w_i^{(\ell)}}{2} \leqslant w_i^{(\ell-1)} \leqslant r_i \qquad \text{pour tout } \ell \in [\![2, m]\!] \text{ et } i \in [\![1, \ell - 1]\!] \tag{4.69}$$
$$0 \leqslant w_i^{(m)} \leqslant r_i \qquad \text{pour tout } i \in [\![1, m]\!] \tag{4.70}$$

L'algorithme de Both et May calcule des listes récursivement. Nous pouvons ranger celles-ci dans un arbre binaire de profondeur m. Si aucune solution n'a été trouvée pour la partition $I \cup J = [\![1, n]\!]$ des positions du code, alors la liste à la racine de l'arbre est vide. Sinon, elle contient un ou plusieurs éléments $\mathbf{e}$ de poids w qui sont des solutions de notre

problème de décodage. Nous comptons les étages de l'arbre depuis les feuilles ; c'est-à-dire que les feuilles sont à l'étage 0 tandis que la racine est à l'étage m.

Les listes que l'on trouve aux nœuds de l'étage $\ell > 0$ contiennent des vecteurs $\mathbf{x}$ tels que :

$$\begin{aligned} |\mathbf{x}_I| &= p^{(\ell)} && (4.71)\\ |\mathbf{x}_{J_i}| &= w_i^{(\ell)} \quad \text{pour } i \in [\![1,\ell]\!] && (4.72)\\ \mathbf{x}_J^\top &= \overline{\mathbf{P}}\mathbf{x}_I^\top && (4.73) \end{aligned}$$

pour toutes les listes *sauf une* qui contient seulement des vecteurs $\mathbf{x}$ pour lesquels nous avons plutôt :

$$\begin{aligned} |\mathbf{x}_I| &= p^{(\ell)} && (4.74)\\ |\mathbf{x}_{J_i}| &= w_i^{(\ell)} \quad \text{pour } i \in [\![1,\ell]\!] && (4.75)\\ \mathbf{x}_J^\top &= \overline{\mathbf{s}}^\top - \overline{\mathbf{P}}\mathbf{x}_I^\top && (4.76) \end{aligned}$$

Ce qui explique que la liste à la racine de l'arbre ne contienne que des éléments $\mathbf{e}$ qui sont solutions du problème de décodage. En effet, d'une part, ces éléments sont de poids w puisque :

$$\begin{aligned} |\mathbf{e}| &= |\mathbf{e}_I| + |\mathbf{e}_J| \\ &= p^{(m)} + \sum_{i=1}^{m} w_i^{(m)} \ \text{(d'après les équations (4.74) et (4.75))} \\ &= p + w - p \ \text{(d'après les équations (4.68) et (4.65))} \\ &= w \end{aligned}$$

D'autre part, puisqu'il n'y a qu'une seule racine dans l'arbre et donc qu'une seule liste à l'étage m, les éléments $\mathbf{e}$ que contiennent celle-ci vérifient l'équation (4.73) :

$$\mathbf{e}_J^\top = \overline{\mathbf{s}} - \overline{\mathbf{P}}\mathbf{e}_I^\top$$

Ce qui montre que nous avons bien $\mathbf{H}\mathbf{e}^\top = \mathbf{s}^\top$.

Les listes $\mathscr{L}_1^{(0)}, \cdots, \mathscr{L}_{2^m}^{(0)}$ aux feuilles de l'arbre (donc à l'étage $\ell = 0$) sont produites différemment des autres : elles sont obtenues par un découpage particulier de l'ensemble d'information $I = I_1 \cup I_2$, où $\sharp I_1 = \left\lfloor \frac{k}{2} \right\rfloor$ et $\sharp I_2 = \left\lceil \frac{k}{2} \right\rceil$ (on retrouve en fait un découpage similaire à celui que l'on a déjà vu dans l'algorithme de Stern). Ces listes sont alors définies ainsi :

$$\begin{aligned} \mathscr{L}_1^{(0)} &= \left\{ \mathbf{x} \in \mathbb{F}_q^n \ : \ \mathbf{x}_J^\top = \overline{\mathbf{s}} - \overline{\mathbf{P}}\mathbf{x}_I^\top \ , \ |\mathbf{x}_{I_1}| = p^{(0)} \text{ et } |\mathbf{x}_{I_2}| = 0 \right\} && (4.77)\\ \mathscr{L}_{2j}^{(0)} &= \left\{ \mathbf{x} \in \mathbb{F}_q^n \ : \ \mathbf{x}_J^\top = \overline{\mathbf{P}}\mathbf{x}_I^\top \ , \ |\mathbf{x}_{I_1}| = 0 \text{ et } |\mathbf{x}_{I_2}| = p^{(0)} \right\} \quad \forall j \in [\![1, 2^m]\!] && (4.78)\\ \mathscr{L}_{2j-1}^{(0)} &= \left\{ \mathbf{x} \in \mathbb{F}_q^n \ : \ \mathbf{x}_J^\top = \overline{\mathbf{P}}\mathbf{x}_I^\top \ , \ |\mathbf{x}_{I_1}| = p^{(0)} \text{ et } |\mathbf{x}_{I_2}| = 0 \right\} \quad \forall j \in [\![2, 2^m]\!] && (4.79) \end{aligned}$$

La raison pour laquelle nous considérons toutes ces listes est double. Tout d'abord, nous avons déjà expliqué que la liste à la racine contient nécessairement des vecteurs solutions de notre problème de décodage. La seconde raison est que ces listes peuvent être calculées récursivement depuis les feuilles grâce à des recherches de presque-collisions. En effet, nous pouvons remarquer que pour tout $\ell \in [\![1, m]\!]$ et tout $j \in [\![1, 2^{m-\ell}]\!]$, la $j^{\text{ème}}$ liste de l'étage ℓ peut être obtenue à partir de la fonction récursive décrite par le pseudo-code 4.5.1 :

Algorithme 4.5.1 : La fonction récursive pour produire la $j^{\text{ème}}$ liste de l'étage ℓ

Fonction *ProduireListe(ℓ, j)*
  si $\ell = 0$ **alors**
    retourner $\mathscr{L}_j^{(0)}$;
  sinon
    $\mathscr{L}_1 \leftarrow$ ProduireListe$(\ell - 1, 2j - 1)$;
    $\mathscr{L}_2 \leftarrow$ ProduireListe$(\ell - 1, 2j)$;
    $\mathscr{L} \leftarrow$ PresqueCollisions$(\mathscr{L}_1, \mathscr{L}_2, \ell)$;
    Filtrer$(\mathscr{L}, \ell)$;
    retourner $\mathscr{L}$; `/* les éléments de` $\mathscr{L}$ `vérifient les équations (4.74), (4.75) et (4.76) si` $j = 1$ `et (4.71), (4.72) et (4.73) sinon. */`
  finSi
finFonction

Fonction *PresqueCollisions($\mathscr{L}_1$, $\mathscr{L}_2$, ℓ)*
  $\mathscr{L} \leftarrow \left\{ \mathbf{x} = \mathbf{x}_1 + \mathbf{x}_2 \ : \ \mathbf{x}_1 \in \mathscr{L}_1 \ , \ \mathbf{x}_2 \in \mathscr{L}_2 \text{ et } |\mathbf{x}_{J_\ell}| = w_\ell^{(\ell)} \right\}$; `/* dans [BM18], il est suggéré d'utiliser l'aglorithme de May et Ozerov [MO15] pour construire cette liste (seulement sur` $\mathbb{F}_2$`). Dans les chapitres ?? à ?? nous proposons une méthode plus efficace et qui se généralise aux espaces non-binaires. */`
  retourner $\mathscr{L}$;
finFonction

Fonction *Filtrer($\mathscr{L}$, ℓ)*
  $\mathscr{L} \leftarrow \left\{ \mathbf{x} \in \mathscr{L} \ : \ |\mathbf{x}_I| = p^{(\ell)} \text{ et } |\mathbf{x}_{J_i}| = w_i^{(\ell)} \text{ pour tout } i \in [\![1, \ell]\!] \right\}$;
finFonction

Finalement, l'algorithme Both-May est décrit par le pseudo-code 4.5.2.

Algorithme 4.5.2 : Algorithme Both-May de profondeur m

Entrées : une matrice de parité $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$ d'un code $\mathcal{C}$;
un syndrome $\mathbf{s} \in \mathbb{F}_q^{n-k}$;
le poids maximal w du vecteur d'erreur recherché.
Sortie : $\mathbf{e} \in \mathbb{F}_q^n$ tel que $\mathbf{H}\mathbf{e}^\top = \mathbf{s}^\top$ et $|\mathbf{e}| = w$.

répéter
  choisir un ensemble $I \subseteq [\![1, n]\!]$ tel que $|I| = k$;
  $J = J_1 \cup \cdots \cup J_m \leftarrow [\![1, n]\!] \setminus I$; `/* pour tout` $i \in [\![1, m]\!]$, $\#J_i = r_i$ `*/`
  $\overline{\mathbf{s}} \leftarrow \mathbf{H}_J^{-1}\mathbf{s}^\top$; `/* si` $\mathbf{H}_J$ `n'est pas inversible revenir à l'étape 2 */`
  $\overline{\mathbf{P}} \leftarrow \mathbf{H}_J^{-1}\mathbf{H}_I$;
  $\mathscr{L} \leftarrow$ ProduireListe$(m, 1)$;
tant que $\mathscr{L} \neq \emptyset$;
retourner $\mathbf{e} \in \mathscr{L}$;

La figure 4.9 illustre la construction récursive de l'ensemble des listes intervenant dans l'algorithme Both-May. Sur cette figure, il est supposé que $I = [\![1, k]\!]$, $J_1 = [\![k + 1, k + r_1]\!]$, $J_2 = [\![k + r_1 + 1, k + r_1 + r_2]\!]$ et $J_3 = [\![k + r_1 + r_2 + 1, k + r_1 + r_2 + r_3]\!]$. Cette figure illustre donc la situation à permutation près des positions.

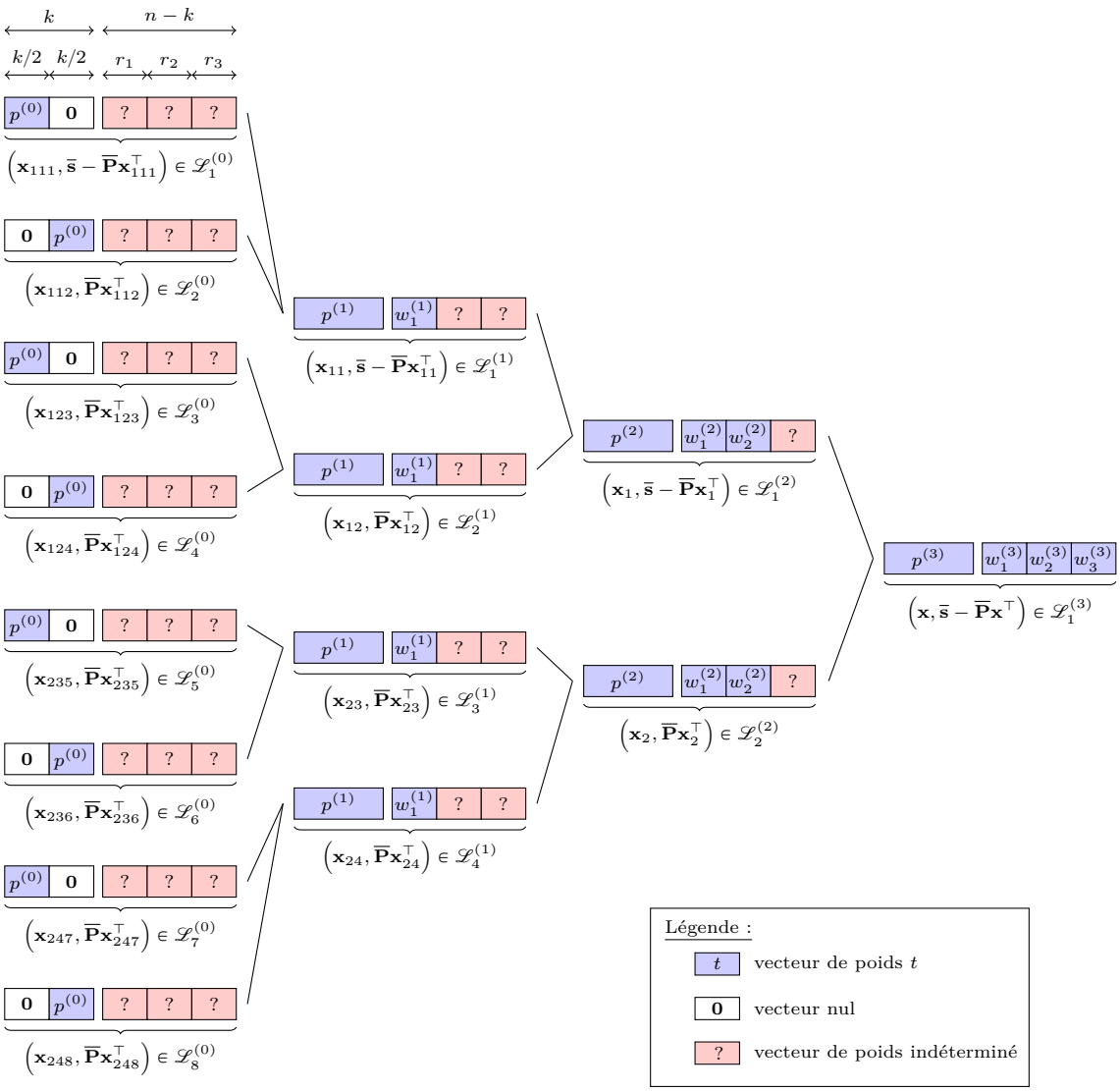


Figure 4.9 – Description de l'algorithme Both-May pour $m = 3$

Remarque 4.5.1. La version de May et Ozerov de l'algorithme de Stern que nous avons rappelée dans le pseudo-code 4.3.2 de la sous-section 4.3.2 est en fait l'instance de l'algorithme de Both et May où $m = 1$.

4.5.2 Choix des paramètres

Un inconvénient de l'algorithme de Both et May est qu'il est difficile à optimiser. En effet, le nombre de paramètres augmente rapidement avec la profondeur m de la récursion.

Nous rappelons ces paramètres :

$$\begin{array}{ccccccccccc}
 & & & & r_1 & ; & \cdots & \cdots & ; & r_m & ; \\
p^{(m)} & ; & w_1^{(m)} & ; & \cdots & \cdots & ; & w_m^{(m)} & ; & & \\
 & & & & \vdots & & & & & & \\
p^{(\ell)} & ; & w_1^{(\ell)} & ; & \cdots & ; & w_\ell^{(\ell)} & ; & & & \\
 & & & & \vdots & & & & & & \\
p^{(2)} & ; & w_1^{(2)} & ; & w_2^{(2)} & ; & & & & & \\
p^{(1)} & ; & w_1^{(1)} & ; & & & & & & & \\
p^{(0)} & . & & & & & & & & &
\end{array} \tag{4.80}$$

Ceux-ci sont contraints par les équations (4.64), (4.65), (4.66), (4.67), (4.68), (4.69) et (4.70). Nous ajoutons une autre contrainte inspirée par le *correctness lemma* [BM18, Lemma 2] de Both et May :

Lemme 4.5.1 (Correctness)**.** *Soit* $\mathbf{e} \in \mathbb{F}_q^n$ *une solution du problème de décodage par syndrome 4.1.2 tel que* $|\mathbf{e}_I| = p$ *et pour tout* $i \in [\![1, m]\!]$, $|\mathbf{e}_{J_i}| = w_i^{(m)}$. *Soit* $R_q(n, w, \overline{w})$ *le nombre de* $\overline{w}$*-représentations d'un vecteur de* $\mathbb{F}_q$. *de poids* w *(cf. sous-section 4.4.1). Si les paramètres de l'algorithme Both-May satisfont :*

$$E(\ell) := R_q\left(k, p^{(\ell)}, p^{(\ell-1)}\right) \cdot \prod_{i=1}^{\ell-1} \frac{R_q\left(r_i, w_i^{(\ell)}, w_i^{(\ell-1)}\right)}{q^{r_i}} \geqslant 1 \qquad \text{pour tout } \ell \in [\![2, m]\!] \tag{4.81}$$

alors nous pouvons espérer avoir au moins un exemplaire de $\mathbf{e}$ *dans* $\mathscr{L}_1^{(m)}$.

Démonstration du lemme 4.5.1.

Il suffit de remarquer que pour tout $j \in [\![1, 2^{m-\ell}]\!]$, $E(\ell)$ est l'espérance du nombre de couples $(\mathbf{x}, \mathbf{y}) \in \mathscr{L}_{2j-1}^{(\ell-1)} \times \mathscr{L}_{2j}^{(\ell-1)}$ tels que $\mathbf{x} + \mathbf{y} = \mathbf{z}$ où $\mathbf{z}$ est un élément particulier de $\mathscr{L}_j^{(\ell)}$.

Nous ne rentrons pas plus en détail dans cette preuve qui est finalement analogue à celle du *correctness lemma* [BM18, Lemma 2]. □

Un choix optimal des paramètres est celui qui minimise les tailles des listes et donc les $E(\ell)$. Nous remplaçons donc les contraintes données par le lemme 4.5.1 par les contraintes d'égalité suivantes :

$$E(\ell) = 1 \qquad \text{pour tout } \ell \in [\![2, m]\!] \tag{4.82}$$

Finalement, les paramètres de l'algorithme de Both et May (cf. équation (4.80)) sont soumis à de nombreuses contraintes d'égalité toutes données par les équations (4.82), (4.64), (4.65) et (4.66) (attention, l'équation (4.68) ne donne pas une contraintes d'égalité mais redéfinit juste la notation d'un des paramètre). Celles-ci permettent de fixer certains paramètres en fonction des autres et ainsi réduire leur nombre dans le processus d'optimisation. Par exemple, pour $m \in \{2, 3, 4\}$, les paramètres de l'algorithme Both-May sont :

$m = 2$	$m = 3$	$m = 4$
r_1 ; r_2 ; $p^{(2)}$; $w_1^{(2)}$; $w_2^{(2)}$; $p^{(1)}$; $w_1^{(1)}$; $p^{(0)}$.	r_1 ; r_2 ; r_3 ; $p^{(3)}$; $w_1^{(3)}$; $w_2^{(3)}$; $w_3^{(3)}$; $p^{(2)}$; $w_1^{(2)}$; $w_2^{(2)}$; $p^{(1)}$; $w_1^{(1)}$; $p^{(0)}$.	r_1 ; r_2 ; r_3 ; r_4 ; $p^{(4)}$; $w_1^{(4)}$; $w_2^{(4)}$; $w_3^{(4)}$; $w_4^{(4)}$; $p^{(3)}$; $w_1^{(3)}$; $w_2^{(3)}$; $w_3^{(3)}$; $p^{(2)}$; $w_1^{(2)}$; $w_2^{(2)}$; $p^{(1)}$; $w_1^{(1)}$; $p^{(0)}$.

où les paramètres en rouge sont ceux fixés par les contraintes d'égalité.

Bien que les contraintes d'égalité diminuent le nombre de paramètres de l'algorithme Both-May et bien que les contraintes d'inégalité données par les équations (4.67), (4.69) et (4.70) réduisent l'espace de définition des paramètres restant, le problème consistant à optimiser ces paramètres reste ardu. Souvent, nous ne dépasserons pas la profondeur $m = 3$. Dans [BM18], Both et May ont étudié le cas où $m = 4$ dans le cas binaire. Leur code permettant d'optimiser les paramètres de leur algorithme a été publié à l'adresse `https://github.com/LeifBoth/Decoding-LPN`. Ce programme ne traite que le cas binaire. En outre, les paramètres retournés par leur programme ne respectent pas toujours exactement le *correctness lemma* 4.5.1.

4.5.3 Complexité de l'algorithme

D'après le lemme 4.5.1, en choisissant nos paramètres de telle sorte que l'équation (4.81) (ou même (4.82)) soit vérifiée, nous garantissons qu'une itération de l'algorithme de Both et May sera fructueuse dès lors qu'une solution particulière $\mathbf{e}$ du problème de décodage que nous considérons a la même distribution de poids que les éléments de la liste $\mathscr{L}_1^{(m)}$; c'est-à-dire dès lors que :

$$\begin{aligned} |\mathbf{e}_I| &= p^{(m)} && (4.83)\\ |\mathbf{e}_{J_i}| &= w_i^{(m)} \quad \text{pour tout } i \in [\![1, m]\!] && (4.84)\\ & && (4.85) \end{aligned}$$

où $I, J_1, \cdots, J_m$ sont les ensembles d'indices choisis au début de l'itération. Ainsi, la probabilité de succès d'une itération de l'algorithme Both-May est :

$$\begin{aligned} \mathbb{P}_{\text{succ}} &:= \frac{\binom{k}{p} \cdot \prod_{i=1}^{m} \binom{r_i}{w_i^{(m)}}}{\binom{n}{w}} && (4.86)\\ &= \tilde{O}\left(q^{k h_q\left(\frac{p}{k}\right) + \sum_{i=1}^{m} r_i h_q\left(\frac{w_i^{(m)}}{r_i}\right) - n h_q\left(\frac{w}{n}\right)} \right) && (4.87) \end{aligned}$$

En outre, pour chaque itération, la compexité en temps de chaque récursion de l'algorithme Both-May est dominée par la recherche de presque-collisions. Afin de déterminer le coût de celle-ci, nous devons estimer les tailles de chaques listes. Pour tout $\ell \in [\![0, m]\!]$, nous notons $S^{(\ell)}$ la taille moyenne d'une des listes $\mathscr{L}_j^{(\ell)}$ (pour tout $j \in [\![1, 2^{m-\ell}]\!]$, cette taille est la même). Tout d'abord, nous avons :

$$S^{(0)} = \binom{k/2}{p^{(0)}} (q-1)^{p^{(0)}} = \tilde{O}\left(q^{\frac{k}{2} h_q\left(\frac{2p^{(0)}}{k}\right)} \right) \tag{4.88}$$

De plus, dans [BM18, section 4], Both et May donnent une borne supérieure de tous les $S^{(\ell)}$ dans le cas binaire. Nous généralisons leur calcul pour les cas non-binaires également. Nous obtenons alors pour tout $\ell \in [\![1, m]\!]$:

$$\begin{aligned} S^{(\ell)} \leqslant\ & \#\left\{\mathbf{x} \in \mathbb{F}_q^k \,|\, |\mathbf{x}| = p^{(\ell)}\right\} \cdot \mathbb{P}\left(|\mathbf{x}_0| = w_\ell^{(\ell)}\right) \\ & \cdot \prod_{i=1}^{\ell-1} \mathbb{P}\left(\Delta(\mathbf{x}_i, \mathbf{y}_i) = w_i^{(\ell)} \;\middle|\; |\mathbf{x}_i| = |\mathbf{y}_i| = w_i^{(\ell-1)}\right) \end{aligned} \tag{4.89}$$

Ce qui nous donne :

$$\begin{aligned} S^{(\ell)} \quad \leqslant \quad & \binom{k}{p^{(\ell)}}(q-1)^{p^{(\ell)}} \cdot \frac{\binom{r_\ell}{w_\ell^{(\ell)}}(q-1)^{w_\ell^{(\ell)}}}{q^{r_\ell}} \\ & \cdot \prod_{i=1}^{\ell-1} \frac{\binom{r_i}{w_i^{(\ell)}}(q-1)^{w_i^{(\ell)}} \cdot R\left(r_i, w_i^{(\ell)}, w_i^{(\ell-1)}\right)}{\left(\binom{r_i}{w_i^{(\ell-1)}}(q-1)^{w_i^{(\ell-1)}}\right)^2} \end{aligned} \tag{4.90}$$

où $\mathbf{x}_0$ est tiré uniformément dans $\mathbb{F}_q^{r_\ell}$ et pour tout $i \in [\![1, \ell-1]\!]$, $\mathbf{x}_i$ et $\mathbf{y}_i$ sont tirés uniformément dans $\mathbb{F}_q^{r_i}$. La formule de Stirling nous permet alors de montrer que pour tout $\ell \in [\![1, m]\!]$:

$$S^{(\ell)} = \tilde{O}(q^\gamma) \tag{4.91}$$

avec :

$$\begin{aligned} \gamma \quad := \quad & k h_q\left(\tfrac{p^{(\ell)}}{k}\right) - r_\ell\left(1 - h_q\left(\tfrac{w_\ell^{(\ell)}}{r_\ell}\right)\right) \\ & + \sum_{i=1}^{\ell-1} r_i\left(h_q\left(\tfrac{w_i^{(\ell)}}{r_i}\right) - 2h_q\left(\tfrac{w_i^{(\ell-1)}}{r_i}\right)\right) + \log_q\left(R\left(r_i, w_i^{(\ell)}, w_i^{(\ell-1)}\right)\right) \end{aligned} \tag{4.92}$$

D'autre part, pour tout $\ell \in [\![1, m]\!]$, nous notons $T^{(\ell)}$ la complexité en temps pour produire une des listes $\mathscr{L}_j^{(\ell)}$. Nous notons $\beta(q, n, L, d)$ le coût pour trouver les d-presque-collisions dans une liste constituée de L éléments tirés uniformément dans $\mathbb{F}_q^n$. Nous avons alors pour tout $\ell \in [\![1, m]\!]$:

$$T^{(\ell)} = \beta\left(q, r_\ell, S^{(\ell-1)}, w_\ell^{(\ell)}\right) \tag{4.93}$$

Finalement la complexité en temps de l'algorithme de décodage de Both et May est :

$$T_{\mathrm{BM}} = \tilde{O}\left(\frac{1}{\mathbb{P}_{\mathrm{succ}}} \cdot \sum_{\ell=1}^{m} 2^{m-\ell} T^{(\ell)}\right) \tag{4.94}$$

$$= \tilde{O}\left(\frac{1}{\mathbb{P}_{\mathrm{succ}}} \cdot \max_{\ell \in [\![1,m]\!]}\left(T^{(\ell)}\right)\right) \tag{4.95}$$

Nous avons aussi sa complexité en mémoire qui est :

$$S_{\mathrm{BM}} = O\left(\sum_{\ell=0}^{m} 2^{m-\ell} S^{(\ell)}\right) \tag{4.96}$$

$$= O\left(\max_{\ell \in [\![0,m]\!]}\left(S^{(\ell)}\right)\right) \tag{4.97}$$

4.6 Nos améliorations du décodage générique

Dans ce qui suit, nous nous intéressons au problème du décodage par syndrome où la distance de décodage w est $d_{\mathrm{GV}}^-(n, k)$. Ce régime correspond au cas où le problème est le plus difficile.

Pour une méthode de décodage donnée, nous exprimons sa complexité par un réel α tel que la complexité de l'algorithme est de l'ordre de $2^{\alpha n(1+o(1))}$ lorsque n tend vers l'infini. Nous appelons alors α l'exposant de la complexité asymptotique de la méthode de décodage.

Précédement dans ce chapitre, nous avons présenté les algorithmes de décodage de [MO15, Hir16, GKH17] que nous avons rappelés dans les pseudo-codes 4.3.2 et 4.4.1. Nous avons aussi présenté une version généralisée aux espaces de Hamming non-binaires de l'algorithme de [BM18] dans le pseudo-code 4.5.2. Dans cette section, nous cherchons à évaluer ces différentes méthodes de décodage. Toutefois, nous avons vu que celles-ci nécessitent toutes une méthode de recherche de presque-collision. Nous devançons ici le lecteur en utilisant des méthodes que nous ne décrirons que plus loin dans ce manuscrit.

Dans la sous-section 4.6.1, nous nous concentrons essentiellement sur l'algorithme de décodage Stern-May-Ozerov. Nous comparons notamment les performances de cette méthode lorsqu'elle est instanciée avec différentes méthodes de recherche de presque-collisions. Dans la sous-section 4.6.2 nous évaluons la complexité du décodage de Both et May que nous avons généralisé aux espaces non-binaires dans le pseudo-code 4.5.2. Nous verrons alors que ce-dernier, associé à notre meilleure méthode de recherche de presque-collisions, nous permet d'obtenir les meilleures complexités du moment pour résoudre le problème de décodage générique.

4.6.1 Notre amélioration de la méthode Stern-May-Ozerov

Dans un premier temps nous nous concentrons sur l'algorithme Stern-May-Ozerov que nous avons rappelé dans le pseudo-code 4.3.2. Il est loin d'être le plus performant en terme de complexité mais il reste relativement simple à comprendre et il fait intervenir la recherche de presque-collisions de façon naturelle. Ainsi, nous pouvons nous faire une première idée de l'impact de nos méthodes de recherche de presque-collisions sur le décodage générique.

La complexité de l'algorithme Stern-May-Ozerov est donnée par le théorème 4.3.3. Cette complexité est intrinsequement liée à la méthode de recherche de presque-collisions utilisée. La méthode LSH des projections (cf. section **??**) nous permet d'obtenir la méthode de Stern originale généralisée aux espaces de Hamming non-binaires. La méthode de Hirose [Hir16] s'inspire de la méthode de May et Ozerov [MO15] sur $\mathbb{F}_2$. Cette solution ne permet pas d'améliorer la méthode de Stern originale. La méthode des codes (cf. section **??**) nous donne un meilleur décodage que celui de Hirose dans [Hir16] mais elle ne permet pas d'améliorer systématiquement la méthode de Stern. Et enfin, notre méthode de recherche de presque-collisions hybride décrite par le pseudo-code **??** du chapitre **??** nous permet finalement d'avoir les meilleures complexités. Tous ces résultats sont résumés dans le tableau 4.1. Dans ce tableau, R est le rendement permettant d'obtenir la pire complexité pour la méthode de décodage considérée.

q	[Hir16]		algo. 4.3.2 et **??** (projections)		algo. 4.3.2 et **??** (codes)		algo. 4.3.2 et **??** (hybride)	
	R	α_{hirose}	R	α_{proj}	R	α_{code}	R	α_{hyb}
2	0.4465	**0.11377**	0.4467	0.11657	0.4465	**0.11377**	0.4465	**0.11377**
3	0.4478	0.18107	0.4483	0.17767	0.4472	0.17560	0.4476	**0.17547**
4	0.4524	0.22268	0.4494	0.21783	0.4482	0.21679	0.4488	**0.21616**
5	0.4542	0.25217	0.4503	0.24718	0.4491	0.24703	0.4499	**0.24587**
7	0.4552	0.29322	0.4517	0.28853	0.4508	0.28962	0.4514	**0.28766**
8	0.4555	0.30851	0.4522	0.30401	0.4516	0.30551	0.4519	**0.30326**
9	0.4557	0.32154	0.4527	0.31722	0.4523	0.31903	0.4525	**0.31658**
11	0.4561	0.34282	0.4535	0.33883	0.4534	0.34104	0.4533	**0.33834**
16	0.4570	0.37961	0.4550	0.37625	0.4555	0.37880	0.4549	**0.37596**
32	0.4589	0.43861	0.4577	0.43635	0.4586	0.43850	0.4576	**0.43625**
64	0.4609	0.48774	0.4602	0.48632	0.4609	0.48772	0.4602	**0.48629**

Tableau 4.1 – Complexités de l'algorithme Stern-May-Ozerov 4.3.2 instancié avec différentes méthodes de recherche de presque-collisions pour $w = d^{-}_{\text{GV}}(n, k)$.

Le décodage en grandes distances. L'algorithme de Stern peut être adapté pour résoudre le problème du décodage à grandes distances. Nous avons vu dans la section 9.2 que la sécurité de certains nouveaux cryptosytèmes comme WAVE [DST19] pouvaient être directement reliés à ce problème. Pour décoder en gros poids avec l'algorithme 4.3.2, nous devons utiliser une méthode de recherche de *lointaines-collisions* ; c'est-à-dire une méthode qui trouve des couples d'éléments éloignés plutôt que proches. Nous verrons dans la suite de ce manuscrit que nos méthodes de recherche de presque-collisions s'adaptent très bien à ce nouveau problème.

Dans [BCDL20] et [Deb19, chapitre 3], diverses méthodes sont proposées pour résoudre le décodage à grandes distances. L'une d'entre elle est une généralisation de la méthode de Prange que nous avons vu précédemment. Une version de la méthode de Dumer en gros poids est aussi présentée ainsi que des améliorations de la méthode de Wagner (une sorte de Dumer à deux niveaux ou de BJMM sans représentation). La figure 4.10 permet de comparer ces méthodes avec le décodage Stern-May-Ozerov instancié avec notre algorithme **??** pour la recherche de lointaines-collisions. Nous constatons alors que cette solution améliore la méthode de Dumer mais pas celle de Wagner (et ses variantes). Nous pouvons toutefois espérer surpasser cette dernière en adaptant le décodage de Both et May aux grandes distances.

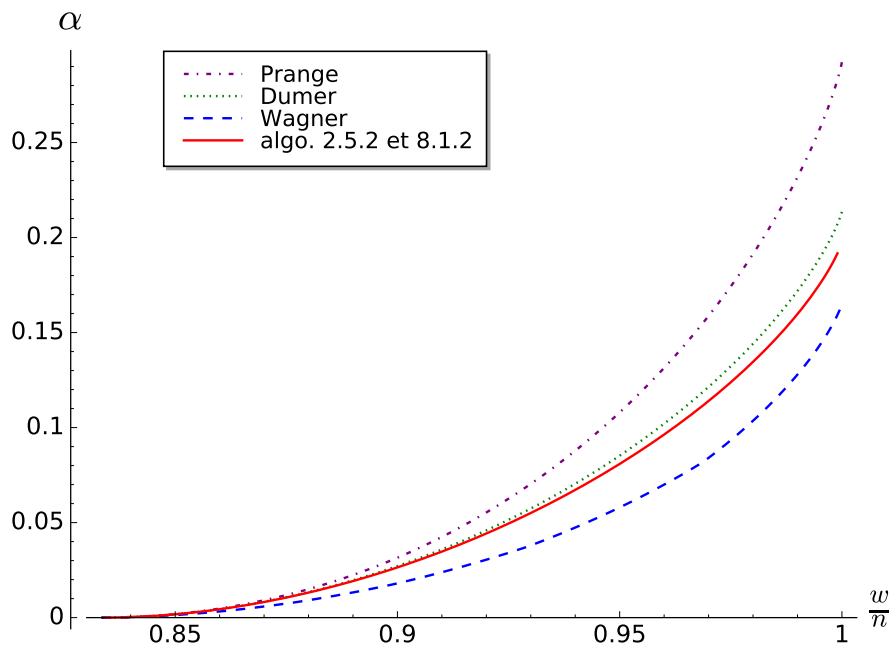


Figure 4.10 – Exposant de la complexité asymptotique pour le décodage en en grandes distances avec Stern-May-Ozerov et notre méthode de recherche de couples éloignés ($q = 3$ et $\frac{k}{n} = \frac{1}{2}$).

4.6.2 Notre amélioration de la méthode Both-May

Le décodage générique binaire. Comme nous l'avons précisé précedemment, l'algorithme Stern-May-Ozerov n'est pas le décodage le plus efficace de la littérature. Notamment, dans le cas binaire, l'algorithme de décodage de Both et May [BM18] permet d'atteindre les meilleures complexités en temps connues à ce jour ; celle-ci est de l'ordre de $2^{0.088453n(1+o(1))}$ pour un décodage à la distance de Gilbert-Varshamov dans le cas du pire rendement

$(R = 0.46)$ [1].

Nous avons adapté l'algorithme de Both et May avec notre recherche de presque-collisions hybride **??** du chapitre **??**. La méthode ainsi obtenue permet d'améliorer la méthode de Both et May originale. En effet, nous pouvons décoder en un temps de l'ordre de $2^{0.088214n(1+o(1))}$ à $d^-_{\mathrm{GV}}(n,k)$ dans le cas du pire rendement. Le tableau 4.3 donne les paramètres de l'algorithme de Both et May qui ont permis d'obtenir cette complexité. Notons que le surcoût $2^{o(n)}$ est nettement meilleur avec notre méthode puisque dans [BM18] il est super-polynomial tandis que dans notre cas, il est polynomial sous l'hypothèse que la conjecture **??** est vraie (dans le cas contraire, ce facteur reste très inférieur à celui de [BM18]).

Le décodage générique non-binaire. Dans les espaces de Hamming non-binaires, le meilleur algorithme de décodage connu à ce jour est celui de Meurer dans [Meu12]. Nous allons voir que notre version de l'algorithme de Both et May combinée à notre meilleur méthode de recherche de presque-collisions permet de dépasser largement les résultats de [Meu12].

Dans la section 4.5, nous avons généralisé la méthode de décodage générique de Both et May aux espaces de Hamming non-binaires ; obtenant ainsi le pseudo-code 4.5.2. Nous combinons cet algorithme avec notre recherche de presque-collisions hybride **??** du chapitre **??**. Pour simplifier la phase d'optimisation des paramètres de l'algorithme 4.5.2, nous nous concentrons exclusivement sur le cas où la profondeur m de la récursion vaut 3. Il faut noter que nous pourrions atteindre de meilleurs résultats en choisissant une valeur de m plus grande (notamment $m = 4$).

Finalement, nous comparons les complexités obtenues avec celles de [Meu12]. Nos résultats sont résumés dans le tableau 4.2. Encore une fois, nous présentons les complexités pour les pires rendements R. Nous constatons que notre méthode améliore significativement le décodage générique non-binaire à la distance de Gilbert-Varshamov. En outre, comme pour le cas binaire, il faut noter que notre méthode ne souffre pas d'un surcoût super-polynomial.

Remarque 4.6.1. Dans [GKH17], il est présenté une généralisation des travaux de May et Ozerov [MO15] pour adapter ceux-ci aux espaces de Hamming non-binaires. Cependant, nous avons relevé des erreurs dans l'analyse présentée dans ce papier et les paramètres proposés ne respectent pas les contraintes exigées par le théorème [GKH17, Theorem 1]. Nous avons donc recalculé les complexités du décodage de [GKH17] en combinant notre analyse de la section 4.4.2 et les travaux de Hirose dans [Hir16]. Nous avons finalement conclu que la méthode de [GKH17] n'est pas plus performante que celle de [Meu12].

	[Meu12]		algo. 4.5.2 et **??**	
q	R	α_{meurer}	R	α
4	0.4355	0.2028	0.4494	**0.18208**
8	0.4417	0.2907	0.4552	**0.26578**
16	0.4468	0.3672	0.4582	**0.34298**
32	0.4525	0.4315	0.4606	**0.41045**
64	0.4576	0.4836	0.4635	**0.47019**

Tableau 4.2 – Complexités de notre méthode de décodage générique et de celle de [Meu12] pour $w = d^-_{\mathrm{GV}}(n,k)$.

1. L'exposant donné dans [BM18] est légèrement optimiste. En effet, il a été obtenu avec des paramètres qui ne vérifient pas exactement le *correctness lemma* 4.5.1 ; nous avons notamment avec leurs paramètres $E(4) = 2^{-0.023435n} < 1$.

	avec la méthode May-Ozerov pour la recherche de presque-collisions	avec notre algorithme hybride **??** pour la recherche de presque-collisions					
q	2	2	4	8	16	32	64
R	0.46	0.46	0.4494	0.4552	0.4582	0.4606	0.4635
w/n	0.123734	0.123734	0.217761	0.278297	0.322530	0.354966	0.378443
m	4	4	3	3	3	3	3
r_1/n	0.037100	0.039380	0.032730	0.032220	0.031570	0.023450	0.021060
r_2/n	0.054900	0.068180	0.072380	0.066630	0.057990	0.047860	0.039790
r_3/n	0.093200	0.088200	0.445490	0.445950	0.452240	0.468090	0.475650
r_4/n	0.354800	0.344240	-	-	-	-	-
$p^{(0)}/n$	0.002809	0.003025	0.008380	0.011285	0.012515	0.012090	0.011445
$p^{(1)}/n$	0.005618	0.006050	0.016760	0.022570	0.025030	0.024180	0.022890
$p^{(2)}/n$	0.010755	0.010950	0.032700	0.042240	0.044650	0.042200	0.038650
$p^{(3)}/n$	0.020310	0.020710	0.051150	0.064220	0.065880	0.060790	0.054550
$p^{(4)}/n$	0.034700	0.035240	-	-	-	-	-
$w_1^{(4)}/n$	0.006600	0.007190	-	-	-	-	-
$w_2^{(4)}/n$	0.009900	0.012280	-	-	-	-	-
$w_3^{(4)}/n$	0.011900	0.010560	-	-	-	-	-
$w_4^{(4)}/n$	0.060630	0.058464	-	-	-	-	-
$w_1^{(3)}/n$	0.004000	0.017600	0.011320	0.014470	0.016900	0.014370	0.014130
$w_2^{(3)}/n$	0.033472	0.041690	0.009890	0.009580	0.007760	0.005020	0.002910
$w_3^{(3)}/n$	0.010608	0.008520	0.145401	0.190027	0.231990	0.274786	0.306853
$w_1^{(2)}/n$	0.023391	0.022060	0.026710	0.030000	0.030950	0.023250	0.021010
$w_2^{(2)}/n$	0.016736	0.020845	0.006825	0.006359	0.005115	0.003240	0.001829
$w_1^{(1)}/n$	0.011696	0.011031	0.013355	0.015171	0.016008	0.011653	0.010632
$\log_q\left(\frac{1}{\mathbb{P}_{\text{succ}}}\right)/n$	0.014620	0.014261	0.024123	0.025142	0.029628	0.035122	0.039012
$\log_q\left(S^{(0)}\right)/n$	0.021880	0.023237	0.032453	0.032154	0.029735	0.035222	0.030519
$\log_q\left(S^{(1)}\right)/n$	0.040019	0.040788	0.058723	0.056997	0.051426	0.044096	0.037767
$\log_q\left(S^{(2)}\right)/n$	0.059352	0.060689	0.052470	0.049790	0.044082	0.036992	0.031155
$\log_q\left(S^{(3)}\right)/n$	0.046383	0.060089	0.018198	0.018399	0.011857	0.002667	−0.004876
$\log_q\left(S^{(4)}\right)/n$	0.034304	0.025270	-	-	-	-	-
$\log_q\left(T^{(1)}\right)/n$	0.045201	0.040788	0.058723	0.056997	0.051426	0.044096	0.037767
$\log_q\left(T^{(2)}\right)/n$	0.073841	0.073954	0.066918	0.063451	0.056118	0.046968	0.039353
$\log_q\left(T^{(3)}\right)/n$	0.073844	0.073953	0.066918	0.063451	0.056118	0.046968	0.039354
$\log_q\left(T^{(4)}\right)/n$	0.056817	0.073954	-	-	-	-	-
$\log_q\left(E^{(2)}\right)/n$	0.000015	$3.6 \cdot 10^{-9}$	$6.8 \cdot 10^{-7}$	$2.9 \cdot 10^{-10}$	$1.4 \cdot 10^{-10}$	$1.9 \cdot 10^{-10}$	$2.3 \cdot 10^{-10}$
$\log_q\left(E^{(3)}\right)/n$	0.003384	$6.0 \cdot 10^{-9}$	$2.1 \cdot 10^{-10}$	$4.5 \cdot 10^{-11}$	$5.5 \cdot 10^{-11}$	$1.4 \cdot 10^{-10}$	$3.2 \cdot 10^{-11}$
$\log_q\left(E^{(4)}\right)/n$	−0.023435	$2.2 \cdot 10^{-9}$	-	-	-	-	-
$\log_q\left(T_{\text{BM}}\right)/n$	0.088453	0.088214	0.091041	0.088594	0.085746	0.082091	0.078366

Tableau 4.3 – Paramètres de l'algorithme 4.5.2.

Le tableau 4.3 donne les paramètres de l'algorithme 4.5.2 qui nous ont permis d'obtenir les complexités données dans le tableau 4.2.

Remarque 4.6.2. Nous donnons les exposants des complexités pour en base 2 et non en base q comme c'est le cas dans [Meu12] et [GKH17].

Remarque 4.6.3. Pour une recherche de presque-collisions sur $\mathbb{F}_q^r$, si $L > q^r$ alors nous utilisons la méthode *meet-in-the-middle* décrite dans [BM18] et adaptée aux espaces non-binaires.

4.7 Le problème LPN

Le problème LPN (ou *Learning from Parity with Noise*) est très proche du problème du décodage générique binaire. C'est un problème bien connu en cryptographie et en théorie des codes. Le problème LPN s'énonce comme suit :

Problème 4.7.1 (LPN (k, τ))**.** *Soit $k \in \mathbb{N}$, $\tau \in [0, 1]$ et un vecteur inconnu $\mathbf{s} \in \mathbb{F}_2^k$. Étant donné un oracle qui produit des échantillons de la forme :*

$$(\mathbf{x}, \widetilde{c}) := (\mathbf{x}, \langle \mathbf{x}, \mathbf{s} \rangle + e) \tag{4.98}$$

où $\mathbf{x}$ est tiré uniformément dans $\mathbb{F}_2^k$, $e = 1$ avec probabilité τ et $e = 0$ avec probabilité $1 - \tau$.
Le but est de trouver le vecteur secret $\mathbf{s} := (s_1, \cdots, s_k)$.

Le problème LPN se réduit trivialement à un problème de décodage générique de rendement arbitraire. Pour cela, il suffit de produire n échantillons $(\mathbf{x}_i, \widetilde{c}_i)$ via l'oracle de LPN (avec n fixé arbitrairement). On pose alors :

$$\mathbf{G} := \left[\begin{array}{c|c|c|c} \mathbf{x}_1^\top & \mathbf{x}_2^\top & \cdots & \mathbf{x}_n^\top \end{array} \right] \tag{4.99}$$

et :

$$\widetilde{\mathbf{c}} := (\widetilde{c}_1, \widetilde{c}_2, \cdots, \widetilde{c}_n) \tag{4.100}$$

On retrouve le vecteur $\mathbf{sG} \in \mathbb{F}_2^n$ en résolvant un décodage générique du vecteur $\widetilde{\mathbf{c}}$ vu comme un mot de code bruité d'un code $\mathcal{C}[n, k]$ dont une matrice génératrice est $\mathbf{G}$. Le vecteur secret $\mathbf{s}$ est alors calculé en inversant k colonnes indépendantes de la matrice $\mathbf{G}$.

Appliquer nos méthodes de décodage ISD directement sur le code généré par $\mathbf{G}$ ne permettrait pas de tirer avantage du caractère arbitraire du rendement du code. Nous allons voir trois astuces pour améliorer notre première approche de résolution du problème LPN. Ces astuces sont tirées de [EKM17].

Astuce 1 : réduction triviale de la dimension. Remarquons qu'il est possible de réduire la dimension k du problème en n'acceptant de l'oracle, que les mots $\mathbf{x}_i$ qui sont nuls sur un certain nombre de positions fixées ; disons les k_1 derniers bits (avec $k_1 \in [\![0, k]\!]$ un paramètre à optimiser). La matrice $\mathbf{G}$ est donc de la forme :

$$\mathbf{G} = \left[\begin{array}{c} \mathbf{G}' \\ \hline \mathbf{0}_{k_1 \times n} \end{array} \right] \tag{4.101}$$

Un décodage du mot $\widetilde{\mathbf{c}}$ dans le code linéaire $[n, k - k_1]$ généré par la matrice $\mathbf{G}'$ nous permet de trouver le mot de code $\mathbf{sG} = \mathbf{s}'\mathbf{G}' \in \mathbb{F}_2^n$ où $\mathbf{s}' := (s_1, \cdots, s_{k-k_1})$. La partie $\mathbf{s}'$ du vecteur secret $\mathbf{s}$ est alors calculée en inversant $k - k_1$ colonnes indépendantes de la matrice $\mathbf{G}'$.

Cette astuce permet de déterminer les $k - k_1$ premiers bits du vecteur secret. Il faut toutefois choisir judicieusement le paramètre k_1 puisque s'il est choisi trop grand, alors il faudra appeler l'oracle un très grand nombre de fois pour avoir suffisamment d'échantillons possédant la bonne propriété. Typiquement, il nous faudra appeler l'oracle $O\big(n2^{k_1}\big)$ fois pour pouvoir construire notre matrice $\mathbf{G}$.

Astuce 2 : élimination gaussienne partielle. Une autre astuce permet de réduire la dimension k du problème LPN. Celle-ci consiste à effectuer une élimination gaussienne partielle en opérant sur les colonnes de la matrice $\mathbf{G}$. On réduit ainsi d'autant la longueur n et la dimension k du problème de décodage générique auquel on s'est ramené. Notons k_2, le nombre de pivots de Gauss effectués. L'élimination gaussienne permet alors de produire une matrice inversible $\mathbf{A}$ de taille $n \times n$ telle que :

$$\mathbf{GA} := \left[\begin{array}{c|c} \mathbf{G'} & \mathbf{B} \\ \hline \mathbf{0}_{k_2 \times (n-k_2)} & \mathbf{Id}_{k_2} \end{array}\right]$$

où $\mathbf{G'}$ est une matrice de taille $(k-k_2) \times (n-k_2)$ et $\mathbf{B}$ est une matrice de taille $(k-k_2) \times k_2$.

Remarque 4.7.1. Pour simplifier notre propos, nous avons omis de parler d'une éventuelle permutation des lignes lors de l'élimination gaussienne. Si une telle permutation est nécessaire pour obtenir une matrice de la forme souhaitée, alors elle doit aussi être appliquée sur le vecteur secret $\mathbf{s}$.

Soit $\tilde{\mathbf{c}}' := (\tilde{c}'_1, \cdots, \tilde{c}'_{n-k_2})$ avec $(\tilde{c}'_1, \cdots, \tilde{c}'_n) := \tilde{\mathbf{c}}\mathbf{A}$. Un décodage du mot $\tilde{\mathbf{c}}'$ dans le code linéaire $[n - k_2, k - k_2]$ généré par la matrice $\mathbf{G'}$ nous permet alors de trouver le mot $\mathbf{s'G'} \in \mathbb{F}_2^{n-k_2}$ où $\mathbf{s'} := (s_1, \cdots, s_{k-k_2})$. Puis, comme précédemment, la partie $\mathbf{s'}$ du vecteur secret $\mathbf{s}$ est calculée en inversant $k - k_2$ colonnes indépendantes de $\mathbf{G'}$.

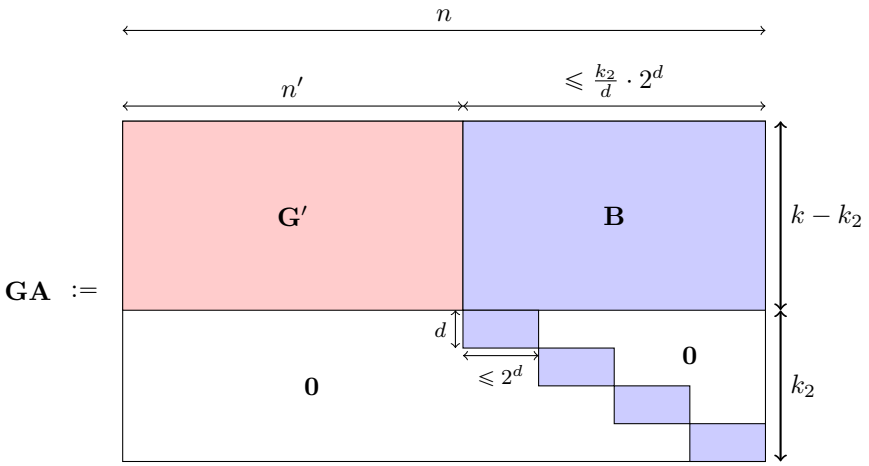
Le problème de cette deuxième astuce est que le mot $\tilde{\mathbf{c}}'$ est beaucoup plus bruité que le mots $\tilde{\mathbf{c}}$. On a donc réduit la dimension du problème LPN au prix d'une augmentation importante du taux d'erreur. Il est possible de quantifier exactement cette augmentation grâce au *piling-up lemma* [Mat93] :

$$\tau' := \mathbb{P}\left(e' = 1\right) = \frac{1 - (1 - 2\tau)^{k_2/2}}{2} \tag{4.102}$$

où e' est un bit de $\mathbf{s'G'} + \tilde{\mathbf{c}}'$.

Astuce 3 : algorithme BKW. L'algorithme BKW [BKW03] permet de produire de nombreux 0 par combinaisons linéaires de colonnes dans la matrice en minimisant le nombre d'opérations à effectuer sur celle-ci. L'algorithme BKW consiste simplement à effectuer une élimination gaussienne par bloc. Cet algorithme ne peut être appliqué que sur des matrices ayant un grand nombre de colonnes ; typiquement, si on choisit une taille de bloc égale à d (d sera aussi un paramètre à optimiser), alors il faudra $\simeq \frac{k_2}{d} \cdot 2^d$ colonnes pour effectuer une élimination gaussienne sur k_2 lignes de la matrice.

L'algorithme BKW permet de produire une matrice inversible $\mathbf{A}$ de taille $n \times n$ telle que :



où $\mathbf{G}'$ est une matrice de taille $(k-k_2)\times n'$ et $\mathbf{B}$ est une matrice de taille $(k-k_2)\times(n-n')$.

Remarque 4.7.2. La même remarque que pour l'astuce 2 sur une éventuelle permutation des lignes lors de l'élimination gaussienne peut être faite ici.

Les opérations à effectuer après application de l'algorithme BKW sont essentiellement les mêmes que dans l'astuce 2. Soit $\widetilde{\mathbf{c}}' := (\widetilde{c}'_1, \cdots, \widetilde{c}'_{n'})$ avec $(\widetilde{c}'_1, \cdots, \widetilde{c}'_n) := \widetilde{\mathbf{c}}\mathbf{A}$. Un décodage du mot $\widetilde{\mathbf{c}}'$ dans le code linéaire $[n', k-k_2]$ généré par la matrice $\mathbf{G}'$ nous permet alors de trouver le mot $\mathbf{s}'\mathbf{G}' \in \mathbb{F}_2^{n'}$ où $\mathbf{s}' := (s_1, \cdots, s_{k-k_2})$. Puis, encore une fois, la partie $\mathbf{s}'$ du vecteur secret $\mathbf{s}$ est calculée en inversant $k-k_2$ colonnes indépendantes de $\mathbf{G}'$.

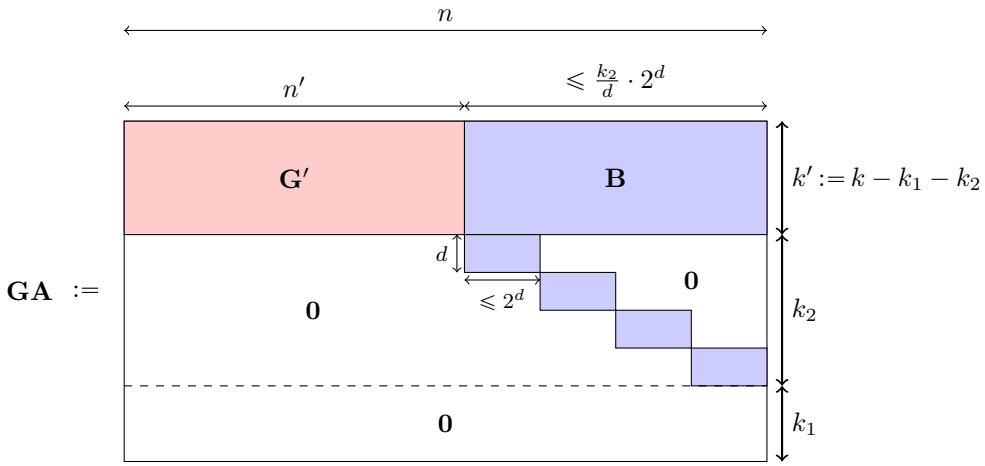
Notre objectif de limiter l'augmentation du taux d'erreur dans le vecteur $\widetilde{\mathbf{c}}\mathbf{A}$ et donc dans le vecteur $\widetilde{\mathbf{c}}'$ est bien rempli puisque cette fois-ci, le *piling-up lemma* nous dit que :

$$\tau' := \mathbb{P}\left(e' = 1\right) = \frac{1-(1-2\tau)^{k_2/d}}{2} \tag{4.103}$$

où e' est un bit de $\mathbf{s}'\mathbf{G}' + \widetilde{\mathbf{c}}'$.

Gardons toutefois à l'esprit que la méthode utilisant BKW a l'inconvénient majeur de nécessiter un grand nombre de requêtes à l'oracle. En pratique, nous n'avons pas toujours accès à une infinité d'échantillons produits par cet oracle. Il est toutefois possible d'outrepasser cette limite en produisant de nouveaux échantillons par combinaisons linéaires de ceux que l'on possède déjà. Il faut cependant être prudent : le taux d'erreur augmente avec la taille du support de la combinaison linéaire. Le *piling-up lemma* permet d'estimer le taux d'erreur des échantillons produits : par exemple, si l'on produit un nouvel échantillon en sommant t échantillons dont le taux d'erreur est τ, alors le nouveau taux d'erreur sera $\frac{1-(1-2\tau)^t}{2}$.

Algorithme Hybride. Les astuces 1 et 3 peuvent être combinées pour obtenir de meilleures performances. On produit alors une matrice $\mathbf{A}$ de taille $n \times n$ telle que :



où $\mathbf{G}'$ est une matrice de taille $k' \times n'$ et $\mathbf{B}$ est une matrice de taille $k' \times (n - n')$.

Les k' premiers bits du vecteur secret $\mathbf{s}$ sont alors déterminés de la même façon que dans l'astuce 3.

Notons à ce stade que seulement une partie du vecteur secret n'est découvert à l'issue de la méthode. Il faut alors itérer celle-ci en poinçonnant les lignes de la matrice $\mathbf{G}$ qui correspondent aux bits déjà trouvés dans le vecteur secret $\mathbf{s}$. À chaque itération, l'instance du problème LPN est plus simple qu'à l'itération précédente puisque sa dimension est réduite d'autant que le nombre de bits déjà trouvés dans $\mathbf{s}$. Un nombre polynomial en k suffit alors pour trouver le secret.

Chapitre 5

Les Codes de Reed-Solomon

Chapitre 6

Les Codes LDPC

6.1 Les codes LDPC vs. les turbo-codes

Les codes LDPC (*Low-Density Parity-Check*) ont été développés par Gallager dans les années 60 dans sa thèse au MIT (*Massachusetts Institute of Technology*) [Gal63]. Comme leur nom l'indique, les codes LDPC sont des codes en bloc linéaires caractérisés par une matrice de parité creuse ; c'est-à-dire une matrice de parité contenant une très grande majorité de coefficients nuls.

Définition 6.1.1 (code LDPC)**.** *Un code LDPC est un code en bloc linéaire possédant une matrice de parité creuse où chaque ligne et chaque colonne possède un nombre borné (et faible) de* 1 *par rapport à la longueur du code.*

Les codes LDPC sont très performants en terme de capacité de correction. Une conjecture prétend même qu'ils sont capables d'atteindre la capacité théorique de Shannon des canaux binaires symétriques lorsque les poids des lignes et des colonnes de la matrice de parité sont correctement choisis. Kudekar, Richardson et Urbanke ont prouvé cette conjecture en 2013 pour une sous-famille de codes LDPC [KRU13]. Malheureusement, Gallager ne connaissait pas ces résultats lorsqu'il publia ses travaux en 1963. De plus, les codes LDPC étaient de longueurs trop importantes pour être utilisables par les technologies de l'époque. C'est pourquoi ils ont suscité moins d'interêts que les codes algébriques, au moins pendant les 30 premières années de leur existance. En outre, du temps de Gallager, les codes LDPC étaient peut-être trop originaux pour plaire. En effet, d'une part, ils étaient très différents des codes algébriques en vogue à son époque et d'autre part, l'algorithme de décodage qu'il proposait était probabiliste ; ce qui était relativement avant-gardiste dans le domaine. Nous avons précisé plus avant que les codes LDPC ont été délaissés en raison de leur manque de praticité. Notons toutefois que l'algorithme de décodage de Gallager n'était pas très différent des algorithmes de décodage simples et efficaces que l'on connait aujourd'hui.

Quelques années avant d'être remis au goût du jour par MacKay et Neal, les codes LDPC se sont vu voler la vedette par les turbo-codes. Ces codes ont été proposés par Berrou au début des années 90 et présentés en juin 1993 par Berrou, Glavieux et Thitimajshima dans [BGT93]. Berrou breveta les turbo-codes pour le compte de France Télécom et TéléDiffusion de France. Au même titre que les codes LDPC, les turbo-codes opèrent avec succès, même pour des rendements proches de la capacité des canaux binaires symétriques et possèdent des algorithmes de décodage itératifs probabilistes efficaces. Ces deux atouts ont permis aux turbo-codes de s'imposer dans de nombreux standards depuis les années 90. On les retrouve par exemple dans les technologies de téléphonie mobile de 3$^{\text{ème}}$ et

$4^{\text{ème}}$ génération (UMTS (3G), LTE (4G)). Ils sont aussi présents dans les communications satellites (Inmarsat, DVB-RCS) et dans les réseaux internet (ADSL2). Les turbo-codes ont également été choisis dans la technologie de communication par courants porteurs en ligne (CPL) qui permet d'utiliser un réseau électrique local pour construire un réseau informatique. En outre, l'Agence Spatiale Européenne (ESA) les a utilisés en 2003 dans une sonde lunaire et depuis, la NASA les utilise aussi dans toutes ses sondes spatiales.

Les codes LDPC ont connu une forte expansion ces 20 dernières années et sont aujourd'hui extrêmement populaires. Ils ont même en quelque sorte supplanté les turbo-codes que l'on trouve de moins en moins dans les normes actuelles. Une des raisons du succès des codes LDPC est la simplicité de leur analyse. En outre, l'essor des turbo-codes a probablement été fortement ralenti par le brevet français auquel ils sont soumis. Les turbo-codes sont toutefois encore préférés aux codes LDPC pour des faibles rendements.

Une liste presque exhaustive des utilisations des codes LDPC avant 2015 a été élaborée par Tixier dans sa thèse [Tix15]. En voici un bref résumé :

— les réseaux filaires Ethernet (normes IEEE 802.3)

— les réseaux locaux sans fil WiFi et WiGig (normes IEEE 802.11) ;

— les réseaux privés sans fil WPAN comme le Bluetooth (normes IEEE 802.15) ;

— les communications mobiles WiMAX (normes IEEE 802.16) ;

— les réseaux sans fil à bande large (normes IEEE 802.20) ;

— les réseaux régionaux sans fil WRAN (normes IEEE 802.22) ;

— les courants porteurs en ligne CPL (normes IEEE 1901-2010) ;

— la technique de modulation radio Ultra Widebande UWB (norme WiMedia) ;

— les transmissions câblées (normes ETSI DVB-C) ;

— les transmissions par satellite (normes ETSI DVB-S et DVB-RCS) ;

— la télévision numérique par liaisons hertziennes terrestres (normes ETSI DVB-T et DVB-NGH) ;

— la téléphonie par satellite (normes ETSI GMR) ;

— les réseaux domestiques câblés à haute vitesse (normes ITU-T G.hn) ;

— les équivalents à l'international des normes DVB (normes ATSC, ISDB, DTMB, CMMB) ;

— l'échange de données spatiales (recommandation CCSDS de la NASA).

Ces 4 dernières années, les codes LDPC ont continué à s'imposer dans les mises à jour des normes où ils étaient déjà présents en 2015. En outre, ils ont fait leur apparition dans la $5^{\text{ème}}$ génération des standards pour la téléphonie mobile (5G).

6.2 Représentation des codes LDPC binaires

Les graphes de Tanner [Tan81] permettent de représenter une matrice binaire sous forme de graphes bipartis. Un premier ensemble de nœuds est indexé par les colonnes de la matrice tandis que l'autre ensemble de nœuds est indexé par ses lignes. Pour tout couple de coordonnées (i, j) de la matrice, le nœud associé à la ligne i est relié au nœud associé à la colonne j si et seulement si le bit de coordonnées (i, j) de la matrice vaut 1. Nous représentons un code en bloc linéaire par le graphe de Tanner associé à une de ses matrices de parité.

Définition 6.2.1 (graphe de Tanner)**.** *Soit $\mathcal{C}$ un code en bloc linéaire $[n, k]$ binaire et soit $\mathbf{H}$ une matrice de parité de ce code. Le graphe de Tanner associé à $\mathbf{H}$ est le graphe biparti composé de n nœuds d'information représentant les colonnes de $\mathbf{H}$ et $n-k$ nœuds de parité représentant les équations de parité (autrement dit les lignes de $\mathbf{H}$). Pour tout $(i, j) \in [\![1, n-k]\!] \times [\![1, n]\!]$, une arête relie le $i^{\text{ème}}$ nœud de parité au $j^{\text{ème}}$ nœud d'information si et seulement si $\mathbf{H}_{i,j} = 1$.*

La figure 6.1 donne un exemple de graphe de Tanner d'un code en bloc binaire linéaire $[6, 2]$.

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

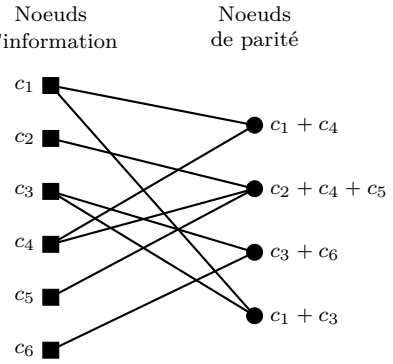


Figure 6.1 – Exemple de graphe de Tanner d'un code binaire linéaire $[6, 2]$.

Un mot binaire $\mathbf{c} := (c_1, \cdots, c_n)$ de longueur n est un mot du code $\mathcal{C}$ si et seulement si $\mathbf{H}\mathbf{c}^\top = \mathbf{0}$. Le graphe de Tanner associé à $\mathbf{H}$ permet de vérifier graphiquement cette égalité. En effet, si on associe chaque nœud d'information i à la valeur c_i, alors $\mathbf{c}$ est un mot de $\mathcal{C}$ si et seulement si pour chaque nœud de parité, la somme modulo 2 des valeurs associées à ses voisins est nulle. Dans le cas d'un code LDPC, cette vérification peut être linéaire en la longueur du code car il existe toujours un graphe de Tanner de degré borné associé à un tel code.

6.3 Les algorithmes de décodage à décisions dures

Les algorithmes de décodage des codes LDPC sont des algorithmes itératifs que l'on peut ranger dans deux grandes catégories : les décodages à décisions souples et dures. Dans [Gal63], Gallager proposait déjà des algorithmes des deux types. Dans cette sous-section, nous parlerons essentiellement des décodages à décisions dures tandis que les décodages à décisions souples seront l'objet de la sous-section suivante.

Nous commençons par présenter un premier algorithme de décodage des codes LDPC que l'on appelle *bit-flipping*. Cet algorithme est dit à décisions dures dans le sens où les décisions qui sont prises tout au long du décodage font essentiellement évoluer des vecteurs binaires (par opposition, les algorithmes de décodage à décisions souples manipulent des vecteurs de probabilités). À chaque itération t de l'algorithme *bit-flipping*, on pose une fonction d'inversion f_t qui prend en entrée un mot binaire $\mathbf{y}$ à corriger et une position $i \in [\![1, n]\!]$ et indique en sortie s'il faut changer la valeur du $i^{\text{ème}}$ bit de $\mathbf{y}$ ou non. Dans la version la plus simple, la fonction f_t est la même pour toutes les itérations :

$$f_t(\mathbf{y}, i) := \begin{cases} \text{oui} & \text{si } \left|\mathbf{H}(\mathbf{y} + \mathbf{e}_i)^\top\right| = \inf\limits_{j \in [\![1,n]\!]} \left(\left|\mathbf{H}(\mathbf{y} + \mathbf{e}_j)^\top\right|\right) \neq \left|\mathbf{H}\mathbf{y}^\top\right| \\ \text{non} & \text{sinon} \end{cases} \tag{6.1}$$

où $\mathbf{H}$ est la matrice de parité du code LDPC et $\mathbf{e}_i$ est le $i^{\text{ème}}$ vecteur de la base canonique. Finalement, une première version de l'algorithme *bit-flipping* est donnée par le pseudo-code 6.3.1.

Algorithme 6.3.1 : La méthode *bit-flipping*

Entrées : une matrice de parité $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$ d'un code LDPC ;
un vecteur binaire $\mathbf{y} \in \mathbb{F}_q^n$ à corriger.
Sortie : un vecteur $\mathbf{c} \in \mathbb{F}_q^n$ tel que $\mathbf{Hc}^\top = \mathbf{0}$ et $\Delta(\mathbf{y}, \mathbf{c})$ est minimal.

$t \leftarrow 0$;
tant que $\mathbf{Hy}^\top \neq \mathbf{0}$ **faire**
  $\mathbf{y}_{\text{tmp}} \leftarrow \mathbf{y}$;
  pour tout $i \in [\![1, n]\!]$ *tel que* $f_t(\mathbf{y}, i) = oui$ **faire**
    inverser le $i^{\text{ème}}$ bit de $\mathbf{y}_{\text{tmp}}$; `/* nous manipulons un vecteur temporaire pour que la condition de la boucle soit indépendante de cette opération */`
  finPour
  $\mathbf{y} \leftarrow \mathbf{y}_{\text{tmp}}$;
finTantQue
retourner $\mathbf{y}$;

Notons que les multiplications matrice/vecteur peuvent être effectuées efficacement en utilisant le graphe de Tanner associé à la matrice creuse $\mathbf{H}$.

De nombreuses améliorations de l'algorithme *bit-flipping* existent [Mac02, ZF04, GH04, MF05, JZSC05, WZX07, LLYS09, WNY+10, HU12, Che13, SWB14, ZYF14, TWS15, CS15, DWV+16, LDG+17, JP17, Mas17, RJ18, BSFZ18, LGK+18] (certaines de ces améliorations ne concernent que des canaux à bruit additif blanc gaussien). Voici quelques idées essentielles à retenir de ces améliorations :

- ***bit-flipping* généralisé.** La fonction d'inversion de l'algorithme *bit-flipping* peut être définie à partir d'un ensemble de fonctions $\{\Delta_i\}_{i\in[\![1,n]\!]}$:

$$f_t(\mathbf{y}, i) := \begin{cases} \text{oui} & \text{si } \Delta_i(\mathbf{y}) = \inf\limits_{j\in[\![1,n]\!]} \{\Delta_j(\mathbf{y})\} \\ \text{non} & \text{sinon} \end{cases} \tag{6.2}$$

  Chaque fonction Δ_i a pour rôle de quantifier la fiabilité du $i^{\text{ème}}$ bit d'un mot binaire. Dans la version originale de Gallager, Δ_i est définie par :

$$\Delta_i(\mathbf{y}) := \left|\mathbf{H}(\mathbf{y} + \mathbf{e}_i)^\top\right| \tag{6.3}$$

  La plupart des améliorations de l'algorithme *bit-flipping* consistent essentiellement à redéfinir les fonctions Δ_i. On peut par exemple pondérer les symboles du syndrome en utilisant l'information reçue (noter que l'information reçue n'est pas nécessairement $\mathbf{y}$: ce peut être par exemple un vecteur de probabilités qui a permis d'engendrer $\mathbf{y}$) :

$$\Delta_i(\mathbf{y}) := \sum_{s=1}^{n-k} \Phi\left(\mathbf{H}_{s,*}\right) \cdot \mathbf{H}_{s,*} \cdot (\mathbf{y} + \mathbf{e}_i)^\top \tag{6.4}$$

  où, pour tout $s \in [\![1, n-k]\!]$, $\mathbf{H}_{s,*}$ est la $s^{\text{ème}}$ équation de parité (ou ligne de $\mathbf{H}$) et Φ est une fonction qui calcule le coefficient de pondération d'une équation de parité. Cette fonction est définie à partir de l'information reçue. La méthode du *bit-flipping* pondéré permet de donner plus d'importance aux mesures de parité en lesquels nous pouvons avoir le plus confiance.

On peut aussi ajouter un terme mesurant la fiabilité des décisions prises :

$$\Delta_i(\mathbf{y}) := \Psi(\mathbf{y} + \mathbf{e}_i) + \sum_{s=1}^{n-k} \Phi\left(\mathbf{H}_{s,*}\right) \cdot \mathbf{H}_{s,*} \cdot (\mathbf{y} + \mathbf{e}_i)^\top \tag{6.5}$$

où Ψ est une fonction du vecteur binaire à tester qui dépend aussi de l'information reçue.

- ***bit-flipping* avec seuil constant.** La fonction d'inversion peut être redéfinie comme suit :

$$f_t(\mathbf{y}, i) := \begin{cases} \text{oui} & \text{si } \Delta_i(\mathbf{y}) = \theta \\ \text{non} & \text{sinon} \end{cases} \tag{6.6}$$

où θ est un seuil à définir. Cette version du *bit-flipping* permet de réduire le nombre d'itérations en corrigeant un plus grand nombre de bits en même temps. Cependant, changer trop de bits à la fois peut engendrer de nouvelles erreurs et empêcher l'algorithme de converger. Il faut donc choisir le seuil θ très judicieusement.

- ***bit-flipping* avec seuil adaptatif.** La méthode est similaire à la précédente sauf que cette fois-ci, le seuil peut varier d'une itération à l'autre et de façon différente d'une position à l'autre :

$$f_t(\mathbf{y}, i) := \begin{cases} \text{oui} & \text{si } \Delta_i(\mathbf{y}) = \theta_t(i) \\ \text{non} & \text{sinon} \end{cases} \tag{6.7}$$

où θ_t est à définir à chaque itération.

- ***bit-flipping* multiple.** Le nombre de bits à inverser lors d'une itération peut parfois être trop élevé ; engendrant alors de nouvelles erreurs. Pour éviter cela, nous pouvons borner le nombre de bits à inverser. Dans l'algorithme 6.3.1, la ligne 4 est alors remplacée par :

```
4 | Pour tout i ∈ I' faire
```

où $I' \subseteq I := \{i \ : \ f_t(\mathbf{y}, i) = \text{oui}\}$ et $\#I' = \min(\#I, M)$ avec M un paramètre à définir.

- ***bit-flipping* probabiliste.** Dans le but encore une fois de réduire le nombre de bits à inverser lors de chaque itération, on peut choisir aléatoirement si un bit vérifiant la fonction d'inversion sera effectivement inversé ou non. Dans l'algorithme 6.3.1, on peut par exemple remplacer la ligne 5 par :

```
5   | | Si Bernoulli(p) = 1 alors
5.1 | |   | inverser le i^ème bit de y_tmp ;
5.2 | | FinSi
```

où $p \in [0, 1]$ est un paramètre à définir et Bernoulli$(p) \in \{0, 1\}$ est tiré selon une loi de Bernoulli de paramètre p.

6.4 Les algorithmes de décodage à décisions souples

Les décodages à décisions dures sont souvent très simples à implémenter et ne demandent que peu de ressources matérielles. Toutefois, malgré les récentes améliorations de ces algorithmes, ils restent moins performants que les décodages à décisions souples.

Un algorithme de décodage à décisions souples est un algorithme itératif. À chaque itération, le graphe de Tanner associé à une matrice de parité creuse du code LDPC est utilisé pour échanger des informations entre les nœuds d'information et les nœuds de parité.

Cet échange d'information a pour objectif de faire évoluer un vecteur de probabilités $\mathbf{p} := (p_1, \cdots, p_n)$ tel que pour tout $i \in [\![1, n]\!]$, p_i représente la probabilité que le $i^{\text{ème}}$ bit du vecteur recherché vaille 1. L'échange entre les nœuds du graphe de Tanner s'effectue en deux temps. D'abord, les nœuds d'information transmettent l'état courant du vecteur $\mathbf{p}$ aux nœuds de parité via les arêtes du graphe. Chaque nœud de parité reçoit donc exclusivement les informations concernant les bits impliqués dans l'équation de parité correspondante. Dans un second temps, les nœuds de parité traitent les informations qu'ils ont reçues pour en produire de nouvelles et les transmettre aux nœuds d'information qui mettent alors à jour le vecteur $\mathbf{p}$. Pour finir, à chaque itération de l'algorithme, le vecteur binaire le plus probable est calculé ; à savoir, le vecteur $\mathbf{x} := (x_1, \cdots, x_n) \in \mathbb{F}_2^n$ qui maximise $\sum_{i=1}^{n} (x_i p_i + (1 - x_i)(1 - p_i))$. Si $\mathbf{x}$ est un mot du code – c'est-à-dire $\mathbf{H}\mathbf{x}^\top = \mathbf{0}$ – alors l'algorithme s'arrête.

Remarque 6.4.1. Le vecteur de probabilités initial peut être obtenu grâce à une démodulation souple. L'intérêt d'une telle démodulation est qu'elle permet de minimiser la perte d'information. Elle peut par exemple se modéliser par un canal à bruit blanc gaussien additif.

Remarque 6.4.2. Les algorithmes à décisions souples trouvent des applications au-delà du décodage des codes LDPC. Ils sont par exemple utilisés en intelligence artificielle [Pea82, KP83, Pea88].

Dans [Gal63], Gallager propose un algorithme de décodage à décisions souples des codes LDPC. Nous présentons ici une version de son algorithme que l'on peut retrouver dans [Moo05]. Cet algorithme est différent de celui introduit plus haut mais l'idée sous-jacente en est analogue. On le retrouve aussi sous l'appellation de décodage *sum-product*. En outre, bien qu'il puisse s'appliquer à divers modèles d'erreur, nous le présentons ici dans le cadre du modèle d'erreur d'un canal binaire symétrique.

On pose de nouveau $\mathcal{C}$ un code LDPC binaire $[n, k]$ de matrice de parité $\mathbf{H}$. Soit un mot de code $\mathbf{c} := (c_1, \cdots, c_n) \in \mathcal{C}$ émis via un canal binaire symétrique de probabilité d'erreur p. Le mot reçu est noté $\mathbf{y} := (y_1, \cdots, y_n) \in \mathbb{F}_2^n$. Nous posons alors le vecteur de probabilités $\mathbf{p} := (p_i, \cdots, p_n)$ tel que pour tout $i \in [\![1, n]\!]$, p_i est la probabilité que $c_i = 1$ sachant la valeur du bit reçu y_i :

$$p_i := \mathbb{P}(c_i = 1 | y_i) = y_i(1 - p) + (1 - y_i)p \tag{6.8}$$

L'algorithme *sum-product* est un décodage à décisions souples qui utilise essentiellement les deux lemmes suivants dont on trouvera les démonstrations dans la thèse de Cluzeau [Clu06] :

Lemme 6.4.1. *Soit $n \in \mathbb{N}^*$ et pour tout $i \in [\![1, n]\!]$, soit b_i une variable aléatoire dans $\mathbb{F}_2$ suivant une loi de Bernoulli de paramètre p_i ; c'est-à-dire $p_i := \mathbb{P}(b_i = 1)$.*

$$\mathbb{P}\left(\sum_{i=1}^{n} b_i \equiv 1 \mod 2\right) = \frac{1 - \prod_{i=1}^{n} (1 - 2p_i)}{2} \tag{6.9}$$

Lemme 6.4.2. *Soit b une variable aléatoire suivant une loi uniforme sur $\mathbb{F}_2$ (donc $\mathbb{P}(b = 1) = \frac{1}{2}$). Soient $n \in \mathbb{N}^*$ et $E_1, \cdots, E_n$ des événements conditionnellement indépendants par rapport à b ; c'est-à-dire $\mathbb{P}(E_1, \ldots, E_n | b) = \mathbb{P}(E_1 | b) \ldots \mathbb{P}(E_n | b)$.*

$$\mathbb{P}(b = 1 | E_1, \cdots, E_n) = \frac{\prod_{i=1}^{n} \mathbb{P}(b = 1 | E_i)}{\prod_{i=1}^{n} \mathbb{P}(b = 1 | E_i) + \prod_{i=1}^{n} (1 - \mathbb{P}(b = 1 | E_i))} \tag{6.10}$$

L'algorithme de décodage *sum-product* se décompose en quatre étapes. La première consiste en une étape d'initialisation. Deux autres étapes seront respectivement appelées étape horizontale et étape verticale car elles nécessitent respectivement un parcours horizontal (lecture ligne par ligne) et vertical (lecture colonne par colonne) de la matrice de parité. Une étape intermédiaire est l'étape de décision qui donne une condition d'arrêt à l'algorithme.

- **Initialisation :** Pour tout nœud d'information i, nous notons V_i les nœuds de parité voisins à i dans le graphe de Tanner associé à $\mathbf{H}$. Pour tout $i \in [\![1, n]\!]$ et pour tout $j \in V_i$, nous initialisons $q_{i,j}$ avec la probabilité p_i. Graphiquement, $q_{i,j}$ peut être vue comme la donnée envoyée par le nœud d'information i au nœud de parité j.
- **Étape horizontale :** Pour tout nœud de parité j, nous notons W_j les nœuds d'information voisins à j dans le graphe de Tanner associé à $\mathbf{H}$. Pour tout $j \in [\![1, n-k]\!]$ et pour tout $i \in W_j$, nous calculons la probabilité $r_{j,i}$ suivante :

$$\begin{aligned} r_{j,i} &:= \mathbb{P}\left(c_i = 1 \mid (q_{s,j})_{s \in W_j \setminus \{i\}}\right) \\ &= \mathbb{P}\left(\sum_{s \in W_j \setminus \{i\}} c_s = 1 \mod 2 \;\middle|\; (q_{s,j})_{s \in W_j \setminus \{i\}}\right) \\ &= \frac{1 - \prod_{k \in W_j \setminus \{i\}} (1 - 2q_{k,j})}{2} \qquad \text{d'après le lemme 6.4.1} \end{aligned}$$

- **Décision :** Pour tout nœud d'information i, les valeurs de p_i et de $(r_{j,i})_{j \in V_i}$ nous donnent des informations sur la valeur de c_i. En effet, pour tout nœud d'information i, le lemme 6.4.2 nous permet de calculer la probabilité q_i suivante :

$$\begin{aligned} q_i &:= \mathbb{P}\left(c_i = 1 \mid p_i, (r_{j,i})_{j \in V_i}\right) \\ &= \frac{p_i \prod_{j \in V_i} r_{j,i}}{p_i \prod_{j \in V_i} r_{j,i} + (1 - p_i) \prod_{j \in V_i} (1 - r_{j,i})} \end{aligned}$$

  Nous posons $\mathbf{c}' = (c'_1, \ldots, c'_n)$ tel que pour tout $i \in [\![1, n]\!]$, $c'_i = \begin{cases} 1 & \text{si } q_i > \frac{1}{2} \\ 0 & \text{sinon} \end{cases}$.
  Nous arrêtons l'algorithme et nous retournons le mot $\mathbf{c}'$ dès lors que ce dernier est un mot du code $\mathcal{C}$.
- **Étape verticale :** Pour tout nœud d'information i et pour tout $j \in V_i$, nous mettons à jour les probabilités $q_{i,j}$ en utilisant une nouvelle fois le lemme 6.4.2 :

$$\begin{aligned} q_{i,j} &:= \mathbb{P}\left(c_i = 0 \mid p_i, (r_{j,i})_{j \in V_i \setminus \{j\}}\right) \\ &= \frac{p_i \prod_{j \in V_i \setminus \{j\}} r_{j,i}}{p_i \prod_{j \in V_i \setminus \{j\}} r_{j,i} + (1 - p_i) \prod_{j \in V_i \setminus \{j\}} (1 - r_{j,i})} \end{aligned}$$

- Nous itérons ensuite à partir de l'**étape horizontale**.

Dans des cas fortement bruités, il se peut que l'algorithme ne converge pas vers une solution. Pour nous assurer que l'algorithme se termine, on fixe un nombre maximal d'itérations. En général, une dizaine d'itérations suffisent.

L'algorithme de décodage *sum-product* n'est pas la seule proposition de décodage à décisions souples. De nombreux autres algorithmes ont été proposés [FMI99, CF02, JVSV03, Gui04, CD05, CGW07, LdL12]. Les différences essentielles dans ces décodages par rapport au décodage *sum-product* résident dans la façon de mettre à jour les probabilités $r_{j,i}$ et $q_{i,j}$. Cependant, l'algorithme *sum-product* original reste l'un des plus performants en terme de capacité de correction.

6.5 Régularité des codes LDPC

Une matrice est dite régulière lorsque ses lignes et ses colonnes sont respectivement toutes de même poids. Un code LDPC régulier est défini par une matrice de parité creuse régulière.

Définition 6.5.1 (code LDPC régulier)**.** *Un code LDPC est dit* régulier *lorsqu'il possède une matrice de parité creuse dont les lignes et les colonnes sont respectivement de même poids ; sinon, il est* irrégulier*.*

Soit une matrice $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$ de rang $n-k$. On suppose que $\mathbf{H}$ est une matrice régulière et on note w_ℓ et w_c, les poids respectifs des lignes et des colonnes de $\mathbf{H}$. Le rendement du code LDPC régulier dont $\mathbf{H}$ est une matrice de parité vérifie :

$$\frac{k}{n} = 1 - \frac{w_c}{w_\ell} \tag{6.11}$$

Cette équation peut nous permettre de choisir les poids des lignes et des colonnes qui maximisent les performances d'un code LDPC régulier.

L'équation (6.11) peut être généralisée aux codes LDPC irréguliers. Soit $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$ une matrice de parité creuse d'un code LDPC. On suppose que $\mathbf{H}$ est de rang plein. Nous rappelons que les poids des colonnes et des lignes de $\mathbf{H}$ sont aussi respectivement les degrés des nœuds d'information et de parité du graphe de Tanner associé à $\mathbf{H}$. Il est possible de relier les distributions des degrés des nœuds d'information et de parité du graphe de Tanner associé à $\mathbf{H}$ au rendement du code LDPC. Nous introduisons pour cela les deux polynômes suivants :

$$P(X) := \sum_{i=1}^{n-k} a_i X^{i-1} \tag{6.12}$$

$$Q(Y) := \sum_{j=1}^{n} b_j Y^{j-1} \tag{6.13}$$

où a_i est la proportion d'arêtes reliées à un nœud d'information de degré i et b_j est la proportion d'arêtes reliées à un nœud de parité de degré j pour tout $(i,j) \in [\![1, n-k]\!] \times [\![1, n]\!]$.

Nous pouvons alors exprimer le rendement du code LDPC en fonction des coefficients de P et Q :

$$\frac{k}{n} = 1 - \frac{\sum_{j=1}^{n} \frac{b_j}{j}}{\sum_{i=1}^{n-k} \frac{a_i}{i}} \tag{6.14}$$

Remarque 6.5.1. En fait, l'équation 6.14 est vérifiée pour n'importe quel graphe de Tanner associé à une matrice de rang plein. Cependant, en pratique, les matrices de parité utilisées ne sont pas toujours de rang plein.

Dans la littérature, il est souvent proposé la construction de codes LDPC réguliers (ou au moins quasi-réguliers ; c'est-à-dire réguliers par bloc). Gallager donne notamment une méthode dans [Gal63] pour construire des codes LDPC réguliers. Notons toutefois que les codes LDPC irréguliers peuvent atteindre de meilleures performances.

6.6 Les codes LDPC quasi-cycliques

Dans la plupart des normes que nous avons évoquées au début de cette section, les codes LDPC ont une structure quasi-cyclique ; c'est-à-dire qu'ils possèdent une matrice de parité qui peut être partitionnée en sous-matrices carrées cycliques.

Définition 6.6.1 (matrices cycliques et quasi-cycliques)**.** *Une matrice carrée $m \times m$ est* circulante *si pour tout $i \in [\![2, m]\!]$, la $i^{ème}$ ligne est une permutation circulaire de la $(i-1)^{ème}$ ligne.*

Une matrice est dite quasi-cyclique *d'ordre m lorsqu'elle se partitionne en sous-matrices carrées $m \times m$ circulante.*

Une matrice circulante $\mathbf{M}_{\mathrm{C}}$ de taille m a donc la forme suivante :

$$\mathbf{M}_{\mathrm{C}} = \begin{bmatrix} x_1 & x_2 & x_3 & \cdots & x_m \\ x_m & x_1 & x_2 & \cdots & x_{m-1} \\ x_{m-1} & x_m & x_1 & \cdots & x_{m-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_2 & x_3 & x_4 & \cdots & x_1 \end{bmatrix} \tag{6.15}$$

Soit α et β deux entiers positifs. Une matrice quasi-cyclique $\mathbf{M}_{\mathrm{QC}}$ d'ordre m et de taille $\alpha m \times \beta m$ a la forme suivante :

$$\mathbf{M}_{\mathrm{QC}} = \left[\begin{array}{c|c|c|c} \mathbf{M}_{\mathrm{C}}^{(1,1)} & \mathbf{M}_{\mathrm{C}}^{(1,2)} & \cdots & \mathbf{M}_{\mathrm{C}}^{(1,\beta)} \\ \hline \mathbf{M}_{\mathrm{C}}^{(2,1)} & \mathbf{M}_{\mathrm{C}}^{(2,2)} & \cdots & \mathbf{M}_{\mathrm{C}}^{(2,\beta)} \\ \hline \vdots & \vdots & \vdots & \vdots \\ \hline \mathbf{M}_{\mathrm{C}}^{(\alpha,1)} & \mathbf{M}_{\mathrm{C}}^{(\alpha,2)} & \cdots & \mathbf{M}_{\mathrm{C}}^{(\alpha,\beta)} \end{array}\right] \tag{6.16}$$

où, pour tout $(i, j) \in [\![1, \alpha]\!] \times [\![1, \beta]\!]$, $\mathbf{M}_{\mathrm{C}}^{(i,j)}$ est une matrice $m \times m$ circulante.

Définition 6.6.2 (code quasi-cyclique)**.** *Un code est dit quasi-cyclique d'ordre m s'il possède une matrice de parité quasi-cyclique d'ordre m.*

Les codes LDPC quasi-cycliques présentent de nombreux avantages. Tout d'abord, il est plus facile de construire un graphe de Tanner sans cycles courts lorsque la matrice de parité qui lui est associée est quasi-cyclique et notamment lorsque les blocs circulants qui la composent sont des rotations de la matrice identité ou des blocs nuls [WYD08]. De plus, il semble que cette structure supplémentaire ne déteriore pas les performances des codes LDPC [ZZ05]. Un autre avantage non négligeable d'un code LDPC quasi-cyclique est que l'espace mémoire nécessaire pour stocker sa matrice de parité est divisé par l'ordre de cyclicité par rapport à un code LDPC aléatoire. En outre, certaines structures quasi-cycliques permettent d'accélérer le décodage et/ou le codage. Par exemple, il est souvent observé des codes LDPC quasi-cycliques dont la matrice génératrice systématique est aussi quasi-cyclique. Notons que les $n-k$ dernières colonnes de la matrice génératrice systématique d'un code LDPC (qu'il soit quasi-cyclique ou non) n'ont aucune raison d'être de poids faibles. Le codage consistant à multiplier le message par la matrice génératrice (systématique) peut alors être relativement coûteux. Toutefois, si le code a une matrice génératrice systématique quasi-cyclique alors la multiplication vecteur/matrice nécessaire au codage peut être effectuée beaucoup plus efficacement grâce à des registres à décalage [PPWW72, section 8.14].

6.7 Les codes LDPC avec une structure convolutive

Les codes convolutifs introduits par Elias en 1955 [Eli55] se sont largement imposés dans les normes de télécommunication entre les années 60 et 90 avant d'être finalement

détrônés par les turbo-codes en 1993 puis par les codes LDPC. Malgré cela, ils sont encore très utilisés aujourd'hui.

Dans [Sha48], Shannon montre que pour n'importe quel canal discret sans mémoire, il existe des codes correcteurs permettant d'approcher autant que l'on veut la capacité du canal. Pour des codes en bloc, ce résultat est asymptotique par rapport à la longueur du code. Il est donc nécessaire de considérer des codes en bloc longs. Cependant, les codes convolutifs sont une alternative aux codes en bloc. En effet, la sortie d'un codeur convolutif de longueur n et de dimension k dépend des k symboles du message à coder mais aussi d'un certain nombre de mots de codes qui l'ont précédé et que l'on a stockés dans des registres. En procédant de cette façon, des mots de taille infinie (ou *flux*) sont produits. Dans [Vit67], Viterbi donne un algorithme de décodage efficace des codes convolutifs qui permet d'atteindre de meilleures performances que les codes en bloc. Son algorithme consiste essentiellement en la recherche du plus court chemin dans un graphe qui est un problème classique en théorie des graphes. Dans [BCJR74], Bahl, Cocke, Jelinek et Raviv proposent un nouveau décodage des codes convolutifs qui est cette fois-ci un décodage itératif à décisions souples. L'algorithme de Viterbi calcule le mot de code le plus probable alors que l'algorithme BCJR calcule pour chaque bit d'information x_i du mot émis, la probabilité $\mathbb{P}(x_i = 1)$. En outre, un autre avantage des codes convolutifs est que leur codage peut être effectué efficacement grâce à l'utilisation de registres à décalage.

Par la suite, les codes convolutifs sont devenus la brique de base de nombreux codes tels que les turbo-codes. Certains codes LDPC utilisent aussi une structure convolutive qui leur permet de bénéficier de l'efficacité des codeurs convolutifs. Nous allons voir succinctement comment certaines formes particulières de matrices de parité peuvent engendrer une telle structure convolutive dans les codes LDPC et comment utiliser ces matrices de parité pour le codage.

Dans les codes LDPC binaires normés, il est souvent observé une forme particulière sur les dernières colonnes des matrices de parité ; à savoir, les coefficients non-nuls de ces colonnes forment une double diagonale telle que la diagonale supérieure commence sur la 1ère ligne et la diagonale inférieure est située m positions en dessous de la diagonale supérieure avec m un diviseur commun de n et $n-k$.

Nous décrivons ici une instance particulière de cette structure que l'on ne trouve pas dans les normes LDPC mais qui nous permet de simplifier notre discours. Nous supposons que les $n-k$ dernières colonnes de la matrice de parité $\mathbf{H}$ forment une matrice quasi-cyclique d'ordre m où les sous-matrices cycliques sont soit la matrice identité $\mathbf{Id}_m$ de taille $m \times m$, soit la matrice nulle $\mathbf{0}_m$ de même taille :

$$\mathbf{H} := \left[\begin{array}{c|ccccc} \mathbf{H}_1 & \mathbf{Id}_m & \mathbf{0}_m & \cdots & \cdots & \mathbf{0}_m \\ \mathbf{H}_2 & \mathbf{Id}_m & \ddots & \ddots & & \vdots \\ \vdots & \mathbf{0}_m & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \mathbf{0}_m \\ \mathbf{H}_t & \mathbf{0}_m & \cdots & \mathbf{0}_m & \mathbf{Id}_m & \mathbf{Id}_m \end{array}\right] \tag{6.17}$$

où $t := \frac{n-k}{m}$ et pour tout $i \in [\![1, t]\!]$, $\mathbf{H}_i$ est une matrice creuse de taille $m \times k$. Nous notons aussi $\mathbf{H}' := \left[\mathbf{H}_1^\top | \cdots | \mathbf{H}_t^\top\right]$.

Avec cette matrice de parité, un codage systématique du code peut être décrit de la façon suivante :

$$\begin{array}{rccc} \textit{codage} : & \mathbb{F}_2^k & \longrightarrow & \mathbb{F}_2^n \\ & \mathbf{x} & \longmapsto & (\mathbf{x}, \mathbf{r}) \end{array} \tag{6.18}$$

où $\mathbf{r} := (r_1, r_2, \cdots, r_{n-k}) := \left(\mathbf{x}\mathbf{H}_1^\top, \mathbf{x}\mathbf{H}_1^\top + \mathbf{x}\mathbf{H}_2^\top, \cdots, \sum_{i=1}^t \mathbf{x}\mathbf{H}_i^\top\right)$.

Nous remarquons alors que les $n-k$ dernières positions du mot de code (ou partie de redondance) sont produites par un codeur convolutif. En effet, pour tout $i \in [\![1, t]\!]$, le $i^{\text{ème}}$ bloc de m bits de la partie de redondance du mot de code est la somme des i premiers blocs de m bits du vecteur $\mathbf{xH'}$. La redondance est donc obtenue en composant la multiplication par la matrice $\mathbf{H'}$ avec un codeur convolutif. La figure 6.2 représente schématiquement le codage donné par l'équation (6.18).

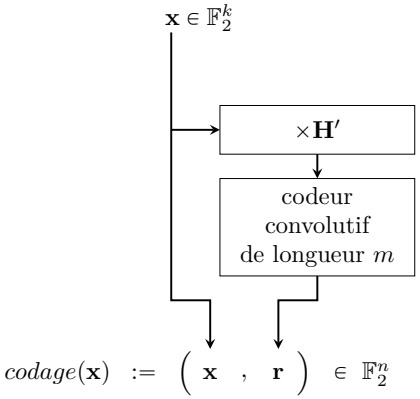


Figure 6.2 – Codage d'un code LDPC avec structure convolutive.

Notons que la matrice $\mathbf{H'}$ est creuse. Nous pouvons donc effectuer le produit matriciel qui précède le codeur convolutif de façon efficace en utilisant par exemple le graphe de Tanner de cette matrice. Un autre avantage de cette structure est qu'elle permet une parallélisation d'un facteur m des calculs. Cette propriété est vraie pour le codage et le décodage mais nous allons l'illustrer ici exclusivement pour le codage. Par la suite, on note $\mathbf{H'}_{*,i}$ la $i^{\text{ème}}$ colonne de $\mathbf{H'}$ (qui est aussi la transposée de la $(i \mod m)^{\text{ème}}$ ligne de la matrice $\mathbf{H}_j$ où $j := \lceil \frac{i}{m} \rceil$). Pour tout $i \in [\![1, m]\!]$, on note $\mathbf{r}_{[i]} := (r_i, r_{i+m}, \cdots, r_{i+tm})$ le mot obtenu en extrayant les positions de $\mathbf{r}$ qui sont congrues à i modulo m. De façon analogue, on note $\mathbf{H'}_{[i]}$ la matrice formée des colonnes de $\mathbf{H'}$ dont les indices sont congrus à i modulo m. La figure 6.3 décrit alors une façon de paralléliser le codage systématique du code LDPC défini par la matrice de parité $\mathbf{H}$ donnée par l'équation (6.17).

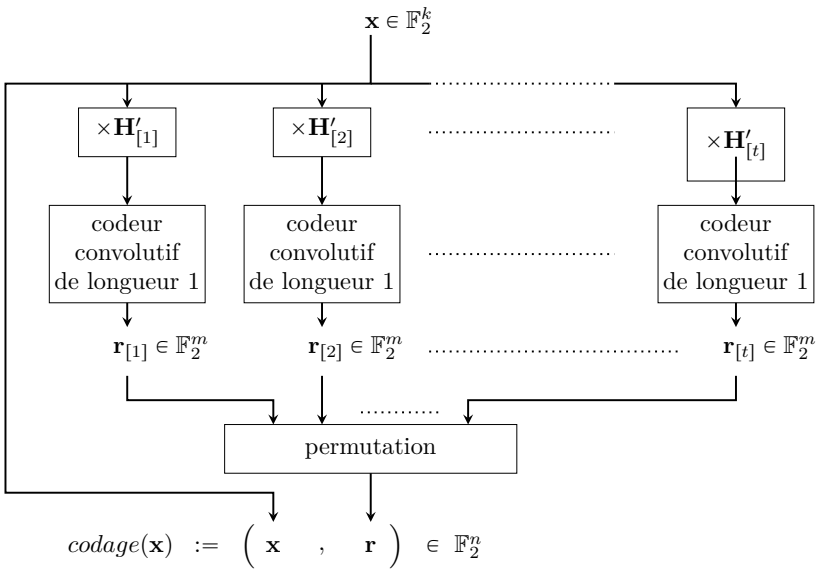


Figure 6.3 – Parallélisation du codage d'un code LDPC avec structure convolutive.

Pour que le décodage d'un code LDPC se passe bien, il faut éviter que sa matrice de parité contienne des colonnes de poids 1. C'est pourquoi les matrices de parité que l'on trouve dans la littérature des normes LDPC sont légèrement différentes de celle décrite par l'équation 6.17. Cela engendre quelques modifications sur le discours que nous venons de tenir (essentiellement des pré-calculs supplémentaires) mais le principe de base reste le même. Dans sa thèse [Tix15], Tixier décrit précisément ces différentes variations.

Pour finir, une structure convolutive peut être cumulée à une structure quasi-cyclique. Les bonnes performances des codes LDPC structurés ainsi que leurs codages et décodages particulièrement efficaces font de ces codes de bons candidats pour des systèmes temps réels.

Chapitre 7

Énumérateur de poids des équations de parité d'un code LDPC

Dans ce chapitre, nous cherchons à approcher l'énumérateur de poids des équations de parité du code $\mathcal{C}$ que nous voulons reconnaître. Nous ne pouvons pas déterminer directement cet énumérateur de poids. En revanche, nous pouvons construire l'énumérateur de poids typique des équations de parité d'une famille de codes ayant une matrice de parité avec des poids de lignes et de colonnes spécifiques. Ainsi, si nous connaissons les poids des lignes et des colonnes d'une matrice de parité de $\mathcal{C}$, alors nous pouvons déterminer l'énumérateur de poids typique des équations de parité d'une famille de codes dont $\mathcal{C}$ fait partie. Cependant, nous ne connaissons pas nécessairement les paramètres d'une matrice de parité de $\mathcal{C}$. Nous pouvons toutefois affiner leur estimation tout au long du processus de reconnaissance de code.

Ce chapitre est essentiel pour la reconnaissance de codes LDPC ou même pour la reconnaissance de n'importe quel code possédant des équations de parité creuses. En effet, comme nous avons pu le voir dans le chapitre précédent, l'énumérateur de poids des équations de parité d'un code est un outil indispensable pour optimiser le paramètre principal t de nos algorithmes de reconnaissance de codes. Les sections 7.1, 7.2 et 7.3 sont principalement des rappels sur les énumérateurs de poids. Dans ces sections, nous cherchons à être le plus général possible : nous nous plaçons notamment dans le corps fini $\mathbb{F}_q$ et non simplement dans $\mathbb{F}_2$. Dans les sections 7.4 et 7.5, nous construisons les polynômes énumérateurs de poids des codes LDPC binaires et de leurs duaux. Ces travaux permettent notamment de donner une expression asymptotique de ces énumérateurs de poids. Ce résultat a été publié dans [**?**, **?**].

7.1 Polynôme énumérateur de poids d'un code

Définition 7.1.1. *Soit $\mathcal{C}[n,k]_q$ un code en bloc linéaire. Le polynôme énumérateur de poids de $\mathcal{C}$, noté $P_{\mathcal{C}}(X)$, est le polynôme univarié à coefficients dans $\mathbb{N}$ tel que pour tout poids w, le coefficient du monôme X^w est égal au nombre de mots du code $\mathcal{C}$ dont le poids de Hamming vaut w. Par définition, le degré de $P_{\mathcal{C}}(X)$ est inférieur ou égal à n.*

Remarque 7.1.1. On définira les polynômes énumérateurs de poids sur $\mathbb{Q}[X]$. Ce qui nous permettra notamment de parler d'énumérateurs de poids moyens.

Par exemple, le polynôme énumérateur de poids du code de répétition de longueur n

et de dimension 1 est :

$$1 + (q-1)X^n$$

Proposition 7.1.2. *Soit $(\mathcal{C}_i)_{i\in[\![1,N]\!]}$, un ensemble de N codes en bloc linéaires. Soient $(P_{\mathcal{C}_i})_{i\in[\![1,N]\!]}$, leurs polynômes énumérateurs de poids.*
Le produit cartésien de codes $\mathcal{C} := \mathcal{C}_1 \times \cdots \times \mathcal{C}_N := \{(\mathbf{c}_1, \cdots, \mathbf{c}_N) \ : \ \forall i \in [\![1, N]\!], \mathbf{c}_i \in \mathcal{C}_i\}$ a pour polynôme énumérateur de poids :

$$P_{\mathcal{C}}(X) = \prod_{i=1}^{N} P_{\mathcal{C}_i}(X)$$

Démonstration de la proposition 7.1.2.

Soit $\mathcal{C}_1 \subseteq \mathbb{F}_q^{n_1}$ et $\mathcal{C}_2 \subseteq \mathbb{F}_q^{n_2}$, deux codes en bloc linéaires. On note leurs polynômes énumérateurs de poids comme suit :

$$P_{\mathcal{C}_1}(X) = \sum_{u=0}^{n_1} A_u X^u \quad \text{et} \quad P_{\mathcal{C}_2}(X) = \sum_{v=0}^{n_2} B_v X^v$$

Le produit de ces deux polynômes est :

$$P_{\mathcal{C}_1}(X) \times P_{\mathcal{C}_2}(X) = \sum_{w=0}^{n_1+n_2} \left(\sum_{u+v=w} A_u B_v \right) X^w$$

Par définition, pour tout entier positif u, A_u est le nombre de mots de poids u du code $\mathcal{C}_1$. Et pour tout entier positif v, B_v est le nombre de mots de poids v du code $\mathcal{C}_2$. Ainsi, le nombre de mots de poids w de la juxtaposition des codes $\mathcal{C}_1$ et $\mathcal{C}_2$ est bien $\sum_{u+v=w} A_u B_v$.

□

Par la suite, nous utilisons la notation suivante pour donner un coefficient particulier d'un polynôme :

Notation 7.1.3. *Soit $P \in \mathbb{Q}[X]$. On note* coef $\left(P \ ; \ X^d\right)$ *le coefficient du monôme de degré d dans P.*

De même, soit $P \in \mathbb{Q}[X, Y]$. On note coef $\left(P \ ; \ X^{d_1} Y_2^d\right)$ *le coefficient du monôme de degré d_1 en X et d_2 en Y dans P.*

7.2 Énumérateur de poids d'un code linéaire aléatoire

Le polynôme énumérateur de poids typique d'un code $\mathcal{C}$ tiré uniformément dans l'ensemble des codes en bloc linéaires aléatoires $[n, k]_q$ est :

$$P_{\mathcal{C}}(X) \simeq \sum_{w=0}^{n} \frac{\binom{n}{w}}{q^n} \times q^k \times X^w = \sum_{w=0}^{n} \frac{\binom{n}{w}}{q^{n-k}} X^w \tag{7.1}$$

En effet, le code $\mathcal{C}$ est un sous ensemble de cardinalité q^k de l'espace vectoriel $\mathbb{F}_q^n$. Or, pour tout poids $w \in [\![0, n]\!]$, la proportion de vecteurs de poids w dans $\mathbb{F}_q^n$ est $\frac{\binom{n}{w}}{q^n}$. De fait, si le code $\mathcal{C}$ est un code linéaire aléatoire, alors ses éléments sont uniformément répartis dans $\mathbb{F}_q^n$; ce qui nous permet d'obtenir l'équation (7.1). De la même façon, le polynôme énumérateur de poids typique du code dual de $\mathcal{C}$ est :

$$P_{\mathcal{C}^\perp}(X) \simeq \sum_{w=0}^{n} \frac{\binom{n}{w}}{q^k} X^w \tag{7.2}$$

La figure 7.1 représente le coefficient du monôme de degré w de $P_{\mathcal{C}}(X) = P_{\mathcal{C}^\perp}(X)$ en fonction de w avec $\mathcal{C}$, un code linéaire binaire aléatoire de longueur 200 et de rendement $\frac{1}{2}$.

Remarque 7.2.1. Si le code est de rendement $\frac{1}{2}$, alors $k = n - k$ et donc il s'en déduit que $P_{\mathcal{C}}(X) = P_{\mathcal{C}^\perp}(X)$.

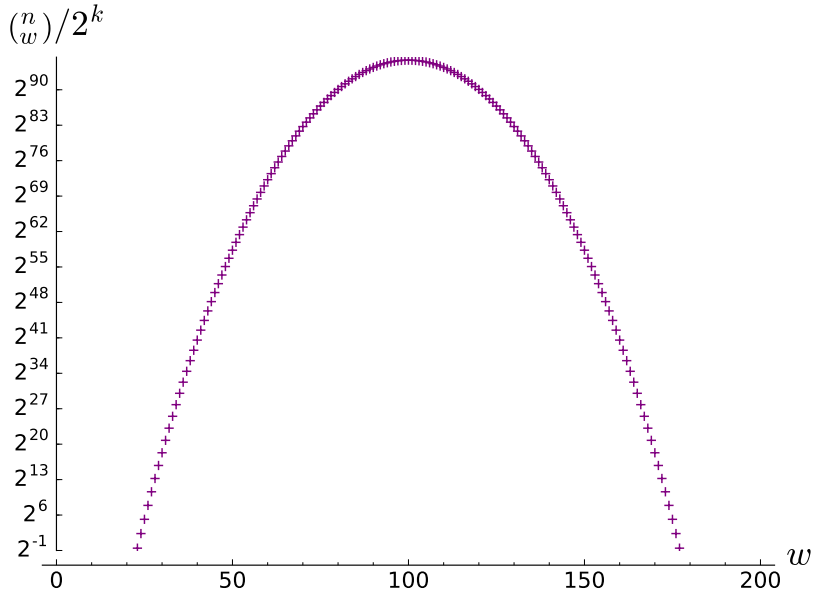


Figure 7.1 – Énumérateur de poids typique d'un code linéaire binaire aléatoire $\mathcal{C}[200, 100]_2$ ou de son dual.

Approcher l'énumérateur de poids du dual d'un code LDPC par l'énumérateur de poids du dual d'un code linéaire aléatoire est insuffisant. En effet, cette approximation sous estime la quantité d'équations de poids faibles. Par exemple, si $\mathcal{C}$ est un code LDPC régulier, il existe au moins $n - k$ mots de $\mathcal{C}^\perp$ de poids minimal ; or la formule précédente n'en trouvera généralement aucune.

Les équations de parité de poids faibles jouent un rôle important dans nos méthodes de reconnaissance de code. Il nous faut donc déterminer un énumérateur de poids des équations de parité plus précis que celui que nous venons de donner.

7.3 Théorème de MacWilliams

Le théorème de MacWilliams permet de relier le polynôme énumérateur de poids d'un code en bloc linéaire avec celui de son dual.

Théorème 7.3.1 (MacWilliams)**.** *Si $\mathcal{C}$ est un code en bloc linéaire $[n, k]_q$ alors :*

$$P_{\mathcal{C}^\perp}(X) = \frac{(1 + (q-1)X)^n}{|\mathcal{C}|} \; P_{\mathcal{C}}\left(\frac{1 - X}{1 + (q-1)X}\right)$$

Démonstration du théorème 7.3.1.

Une démonstration du théorème de MacWilliams est donnée dans le cours d'algèbre de Michel Demazure [**?**]. □

Notons $E_{n,R} \subseteq \mathbb{Q}[X]$, l'espace vectoriel constitué des polynômes énumérateurs de poids de l'ensemble des codes en bloc linéaires q-aire de longueur n et de rendement R.

Proposition 7.3.2. *L'identité de MacWilliams :*

$$\begin{array}{rccc} \varphi & : & E_{n,R} & \longrightarrow & E_{n,1-R} \\ & & P_{\mathcal{C}} & \longmapsto & P_{\mathcal{C}^\perp} \end{array} \tag{7.3}$$

donnée par le théorème 7.3.1 est une application linéaire.

La proposition 7.3.2 montre que l'identité de Mac Williams est linéairement stable ; ce qui permet notamment d'affirmer que la transformation de MacWilliams du polynôme énumérateur moyen de codes linéaires $[n, nR]$ est la moyenne des transformations de MacWilliams de chacun des codes.

L'identité de MacWilliams peut nous permettre notamment de déterminer le polynôme énumérateur de poids d'un code de parité. En effet, le code de parité $[n, n-1]$ est le code dual du code de répétition $[n, 1]$ dont le polynôme énumérateur de poids est $1 + (q-1)X^n$. Ainsi, le polynôme énumérateur de poids du code de parité est :

$$\frac{(1+(q-1)X)^n}{q^1}\left(1+(q-1)\left(\frac{1-X}{1+(q-1)X}\right)^n\right) = \frac{(1+(q-1)X)^n + (q-1)(1-X)^n}{q} \tag{7.4}$$

Ce polynôme nous sera utile par la suite pour construire le polynôme énumérateur de poids des équations de parité d'un code LDPC.

7.4 Énumérateur de poids d'un code LDPC

Dans cette section, nous nous plaçons dans le cas binaire. Nous nous inspirons des travaux de Die, Richardson et Urbanke dans [**?**] pour construire le polynôme énumérateur de poids typique d'un code LDPC binaire. Dans ce papier, les auteurs utilisent les graphes de Tanner [Tan81] que nous avons déjà évoqués dans la section **??**. Nous rappelons leur définition :

Définition 7.4.1. *Le graphe de Tanner associé à une matrice de parité* $\mathbf{H}$ *d'un code linéaire* $\mathcal{C}[n,k]_2$ *est le graphe biparti composé de* n *nœuds d'information représentant les colonnes de* $\mathbf{H}$ *et* $n-k$ *nœuds de parité représentant les équations de parité (c'est-à-dire les lignes de* $\mathbf{H}$*). Le* $i^{ème}$ *nœud d'information est relié au* $j^{ème}$ *nœud de parité si et seulement si* $\mathbf{H}_{i,j} = 1$.

Un exemple est donné dans la figure 7.2.

$$\mathbf{H}_0 = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

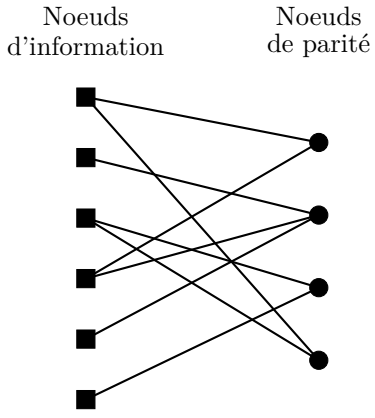


Figure 7.2 – Exemple de graphe de Tanner d'un code linéaire $[6,2]_2$.

Pour chaque nœud d'information g_i (respectivement, nœud de parité d_j), il sera noté α_i (respectivement β_j), le degré de ce nœud.

Un mot binaire $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ est un mot du code $\mathcal{C}$ si et seulement si $\mathbf{H}\mathbf{c}^T = \mathbf{0}$. Le graphe de Tanner associé à $\mathbf{H}$ permet de vérifier graphiquement cette égalité. En effet, si on affecte la valeur c_i à chaque nœud d'information g_i, alors $\mathbf{c}$ est un mot de $\mathcal{C}$ si et seulement si pour chaque nœud de parité d_j, la somme modulo 2 des valeurs associées à ses voisins est nulle.

Le graphe de Tanner de $\mathbf{H}$ peut alors être vu comme un graphe normal (graphe possédant des nœuds logiques) dont chaque nœud d'information g_i est succédé par un nœud de répétition G_i de degré $\alpha_i + 1$ et chaque nœud de parité d_j est précédé d'un nœud d'addition D_j sur $\mathbb{F}_2$ de degré $\beta_j + 1$. Les nœuds d'information et les nœuds de parité seront donc de degré 1. Un exemple de graphe normal de Tanner est donné dans la figure 7.3. Sur celle-ci, les nœuds de répétition sont représentés par le symbole $=$ et les nœuds d'addition sont représentés par le symbole $\bigoplus$.

Dans le graphe normal de Tanner, une affectation $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ des nœuds d'informations $g_1, \cdots, g_n$ représente un mot du code $\mathcal{C}$ si et seulement si tous les nœuds logiques $G_1, \cdots, G_n$ et $D_1, \cdots, D_{n-k}$ sont vérifiés lorsque l'on affecte des valeurs nulles aux nœuds de parité $d_1, \cdots, d_{n-k}$.

Dans ce nouveau graphe, nous avons aussi introduit des nœuds intermédiaires sur les liens reliant les nœuds de répétition $=$ et les nœuds d'addition $\bigoplus$. Ces nœuds intermédiaires sont représentés par des disques noirs $\bullet$ sur la figure 7.3. Pour tout $i \in [\![1, n]\!]$, on note $\{\gamma_{(i,s)}\}_{s\in[\![1,\alpha_i]\!]}$ l'ensemble des nœuds intermédiaires ajoutés au voisinage du nœud de répétition G_i et pour tout $j \in [\![1, n-k]\!]$, , on note $\{\delta_{(j,s)}\}_{s\in[\![1,\beta_j]\!]}$ l'ensemble des nœuds intermédiaires ajoutés au voisinage du nœud d'addition D_j.

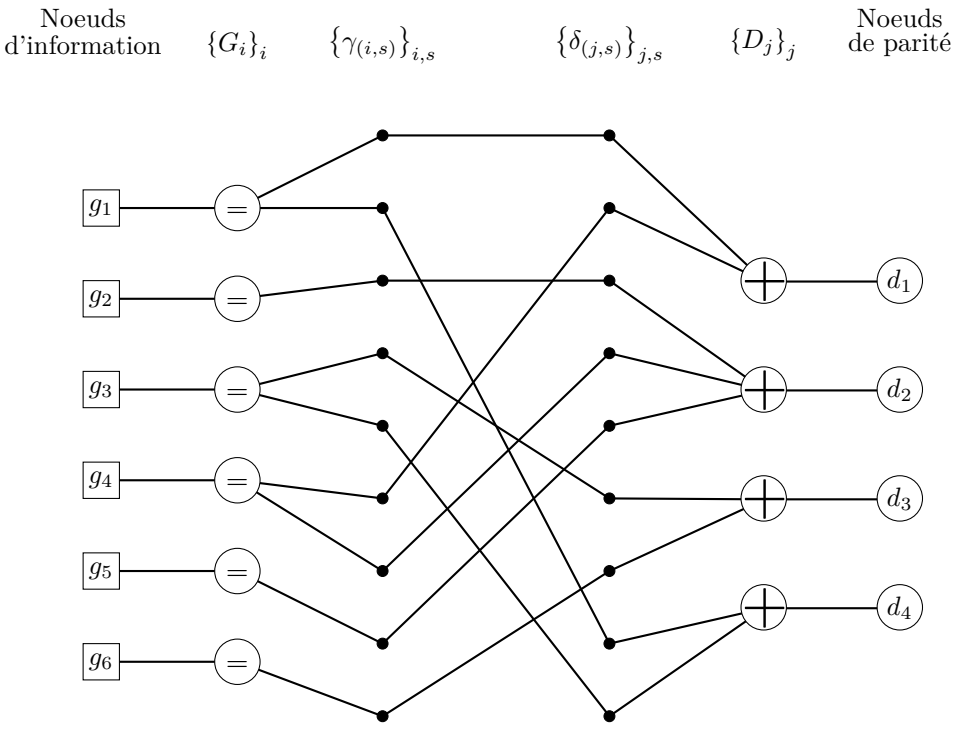
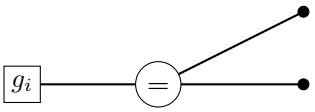


Figure 7.3 – Graphe normal de Tanner associé à la matrice de parité $\mathbf{H}_0$ de la figure 7.2.

Nous allons à présent construire pas-à-pas l'énumérateur de poids du code $\mathcal{C}$. Pour cela, nous subdivisons le graphe normal de Tanner en différents sous-graphes. Nous observons alors que chacun de ces sous-graphes représente un code particulier dont nous pouvons déterminer l'énumérateur de poids. Enfin, l'étape finale de notre construction consiste à assembler toutes nos observations pour construire l'énumérateur de poids de $\mathcal{C}$.

Observation 1. Pour tout $i \in [\![1, n]\!]$, le sous graphe composé du nœud de répétition G_i et de tous ses voisins modélise un code de répétition que l'on note $\mathcal{C}_i^{\text{rep}}$.



Un mot $(c_0, \cdots, c_{\alpha_i})$ de longueur $\alpha_i + 1$ est un mot du code $\mathcal{C}_i^{\text{rep}}$ si et seulement si le nœud logique G_i est satisfait lorsque l'on affecte les valeurs c_0, c_1, $\cdots$, c_{α_i} aux nœuds respectifs g_i, $\gamma_{(i,1)}$, $\cdots$, $\gamma_{(i,\alpha_i)}$.

Le polynôme énumérateur de poids du code $\mathcal{C}_i^{\text{rep}}$ est :

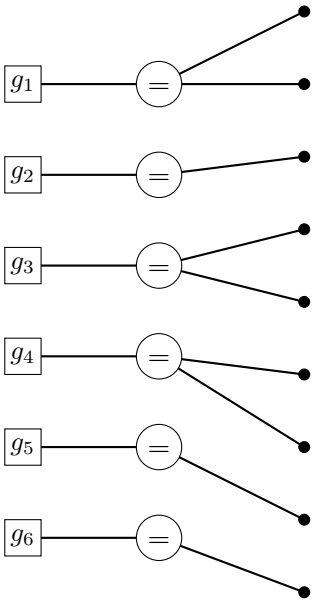
$$P_{\mathcal{C}_i^{\text{rep}}}(X) = 1 + X^{\alpha_i} \tag{7.5}$$

Toutefois, nous pouvons définir un polynôme énumérateur de poids décrivant plus précisément le poids des mots de $\mathcal{C}_i^{\text{rep}}$. En effet, l'utilisation d'un polynôme bivarié permet de distinguer le poids du bit affecté au nœud g_i du poids des bits affectés aux nœuds $\gamma_{(i,1)}$, $\cdots$, $\gamma_{(i,\alpha_i)}$:

$$P_{\mathcal{C}_i^{\text{rep}}}(X, Y) = 1 + XY^{\alpha_i} \tag{7.6}$$

Dans ce polynôme, le coefficient du monôme $X^{w_1}Y^{w_2}$ est le nombre de mots de $\mathcal{C}_i^{\text{rep}}$ de poids w_1 sur le bit associé au nœud g_i et de poids w_2 sur les bits associés aux nœuds $\gamma_{(i,1)}$, $\cdots$, $\gamma_{(i,\alpha_i)}$.

Observation 2. Le sous-graphe composé de tous les graphes associés aux codes $\{\mathcal{C}_i^{\text{rep}}\}_{i\in[\![1,n]\!]}$ modélise le produit cartésien de codes $\mathcal{C}^{\text{rep}} := \mathcal{C}_1^{\text{rep}} \times \cdots \times \mathcal{C}_n^{\text{rep}}$.



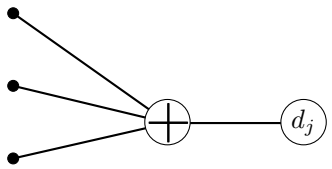
D'après la proposition 7.1.2, le polynôme énumérateur de poids du code $\mathcal{C}^{\text{rep}}$ est :

$$P_{\mathcal{C}^{\text{rep}}}(X,Y) = \prod_{i=1}^{n} P_{\mathcal{C}_i^{\text{rep}}}(X,Y) \tag{7.7}$$

$$= \prod_{i=1}^{n} (1 + XY^{\alpha_i}) \tag{7.8}$$

Dans ce polynôme, le coefficient du monôme $X^{w_1}Y^{w_2}$ est le nombre de mots de $\mathcal{C}^{\text{rep}}$ de poids w_1 sur les bits associés aux nœuds $\{g_i\}_{i\in[\![1,n]\!]}$ et de poids w_2 sur les bits associés aux nœuds $\left\{\gamma_{(i,s)}\right\}_{(i,s)\in[\![1,n]\!]\times[\![1,\alpha_i]\!]}$.

Observation 3. Pour tout $j \in [\![1, n-k]\!]$, le sous-graphe composé du nœud d'addition D_j et de tous ses voisins modélise un code de parité que l'on note $\mathcal{C}_j^{\text{par}}$.



Un mot $(c_0, \cdots, c_{\beta_j})$ de longueur $\beta_j + 1$ est un mot du code $\mathcal{C}_j^{\text{par}}$ si et seulement si le nœud logique D_j est satisfait lorsque l'on affecte les valeurs c_0, c_1, $\cdots$, c_{β_j} aux nœuds respectifs d_j, $\delta_{(j,1)}$, $\cdots$, $\delta_{(j,\beta_j)}$.

Le polynôme énumérateur de poids du code $\mathcal{C}_j^{\text{par}}$ est :

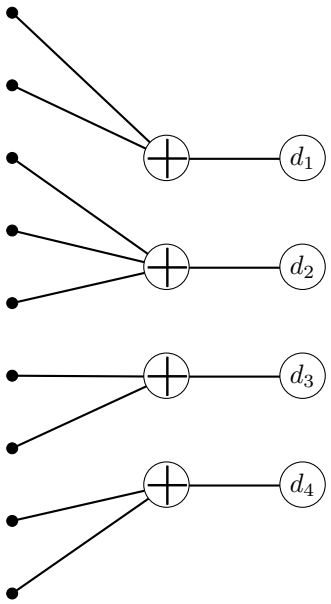
$$P_{\mathcal{C}_j^{\text{par}}}(X) = \frac{(1+X)^{\beta_j} + (1-X)^{\beta_j}}{2} \tag{7.9}$$

Nous distinguons toutefois le poids du bit affecté au nœud d_j du poids des bits affectés aux nœuds $\delta_{(j,1)}, \cdots, \delta_{(j,\beta_j)}$. Pour cela, nous définissons le polynôme énumérateur de poids bivarié suivant :

$$P_{\mathcal{C}_j^{\text{par}}}(Y,Z) = \frac{(1+Z)(1+Y)^{\beta_j} + (1-Z)(1-Y)^{\beta_j}}{2} \tag{7.10}$$

Dans ce polynôme, le coefficient du monôme $Y^{w_1}Z^{w_2}$ est le nombre de mots de $\mathcal{C}_j^{\text{par}}$ de poids w_1 sur les bits associés aux nœuds $\delta_{(j,1)}, \cdots, \delta_{(j,\alpha_j)}$ et de poids w_2 sur le bit associé au nœud d_j .

Observation 4. Le sous-graphe composé de tous les graphes associés aux codes $\left\{\mathcal{C}_j^{\text{par}}\right\}_{j\in[\![1,n-k]\!]}$ modélise le produit cartésien de codes $\mathcal{C}^{\text{par}} := \mathcal{C}_1^{\text{par}} \times \cdots \times \mathcal{C}_{n-k}^{\text{par}}$.



D'après la proposition 7.1.2, le polynôme énumérateur de poids du code $\mathcal{C}^{\text{par}}$ est :

$$P_{\mathcal{C}^{\text{par}}}(Y,Z) = \prod_{j=1}^{n-k} P_{\mathcal{C}_j^{\text{par}}}(Y,Z) \tag{7.11}$$

$$= \prod_{j=1}^{n-k} \left(\frac{(1+Z)(1+Y)^{\beta_j} + (1-Z)(1-Y)^{\beta_j}}{2} \right) \tag{7.12}$$

Dans ce polynôme, le coefficient du monôme $Y^{w_1} Z^{w_2}$ est le nombre de mots de $\mathcal{C}^{\text{par}}$ de poids w_1 sur les bits associés aux nœuds $\left\{\delta_{(j,s)}\right\}_{(j,s)\in[\![1,n-k]\!]\times[\![1,\beta_j]\!]}$ et de poids w_2 sur les bits associés aux nœuds $\{d_j\}_{j\in[\![1,n-k]\!]}$.

Conclusion. Soit $\mathbf{c} = (c_1, \cdots, c_n)$ un mot de $\mathbb{F}_2^n$ de poids w. Le mot $\mathbf{c}$ est associé à un unique mot $\mathbf{c}' \in \mathcal{C}^{\text{rep}}$ dont les bits associés aux nœuds $g_1, \cdots, g_n$ valent respectivement $c_1, \cdots, c_n$. Notons $w + \ell$ le poids du mot $\mathbf{c}'$. Supposons que les nœuds $\left\{\gamma_{(i,s)}\right\}_{(i,s)\in[\![1,n]\!]\times[\![1,\alpha_i]\!]}$ et $\left\{\delta_{(j,s)}\right\}_{(j,s)\in[\![1,n-k]\!]\times[\![1,\beta_j]\!]}$ sont reliés aléatoirement. Alors le mot $\mathbf{c}'$ est associé aléatoirement à un unique mot $\mathbf{c}'' \in \mathcal{C}^{\text{par}}$ dont le poids des bits associés aux nœuds $\left\{\delta_{(j,s)}\right\}_{(j,s)\in[\![1,n-k]\!]\times[\![1,\beta_j]\!]}$ vaut ℓ.

La probabilité que $\mathbf{c}''$ soit nul sur les bits associés aux nœuds $d_1, \cdots, d_{n-k}$ est :

$$\frac{\text{coef}\left(P_{\mathcal{C}^{\text{par}}}(Y,Z)\ ;\ Y^\ell Z^0\right)}{\binom{N}{\ell}} = \frac{\text{coef}\left(P_{\mathcal{C}^{\text{par}}}(Y,0)\ ;\ Y^\ell\right)}{\binom{N}{\ell}} \quad \text{avec} \quad N := \sum_{i=1}^{n} \alpha_i = \sum_{j=1}^{n} \beta_j \tag{7.13}$$

Finalement, l'espérance A_w du nombre de mots de poids w de $\mathcal{C}$ est :

$$A_w = \sum_{\ell=0}^{N} \frac{\text{coef}\left(P_{\mathcal{C}^{\text{rep}}}(X,Y)\ ;\ X^w Y^\ell\right) \cdot \text{coef}\left(P_{\mathcal{C}^{\text{par}}}(Y,0)\ ;\ Y^\ell\right)}{\binom{N}{\ell}} \tag{7.14}$$

Notons que si les poids $(\beta_j)_{j\in[\![1,n-k]\!]}$ des lignes de la matrice de parité $\mathbf{H}$ du code $\mathcal{C}$ sont constants égaux à β, alors le polynôme univarié $P_{\mathcal{C}^{\text{par}}}(Y,0)$ devient :

$$P_{\mathcal{C}^{\text{par}}}(Y,0) = \left(\frac{(1+Y)^\beta + (1-Y)^\beta}{2}\right)^{n-k} \tag{7.15}$$

Nous remarquons aussi que lorsque les poids $(\alpha_i)_{i\in[\![1,n]\!]}$ des colonnes de la matrice de parité $\mathbf{H}$ du code $\mathcal{C}$ sont constants égaux à α, le polynôme bivarié $P_{\mathcal{C}^{\text{rep}}}(X,Y)$ devient :

$$P_{\mathcal{C}^{\text{rep}}}(X,Y) \;=\; \sum_{w=0}^{n} \binom{n}{w} X^w Y^{\alpha w} \tag{7.16}$$

et donc l'espérance A_ω devient :

$$A_w = \frac{\binom{n}{w} \cdot \text{coef}\left(P_{\mathcal{C}^{\text{par}}}(Y,0)\ ;\ X^{\alpha w}\right)}{\binom{N}{\alpha w}} \tag{7.17}$$

Soit Ω la famille des codes linéaires possédant une matrice de parité dont les poids des colonnes et des lignes sont respectivement donnés par les α_i et les β_j. Par construction, le code $\mathcal{C}$ que nous recherchons appartient à la famille de codes Ω. Le polynôme énumérateur $\sum_{w=0}^{n} A_w X^w$ est le polynôme éumérateur de poids typique d'un code de Ω dont $\mathcal{C}$ fait partie. Nous pouvons vérifier expérimentalement que cet énumérateur de poids est très proche de l'énumérateur de poids du code particulier $\mathcal{C}$.

Pour finir, l'identité de MacWilliams étant stable par passage à la moyenne (cf. section 7.3), il est possible de déterminer le polynôme énumérateur de poids typique des équations de parité d'un code de Ω à partir du polynôme $\sum_{w=0}^{n} A_w X^w$.

Soit $\Omega[200,100,3,6]_2$ une famille de codes LDPC réguliers binaires de longueur 200 et de rendement $\frac{1}{2}$ possédant des matrices de parité dont les poids des lignes et des colonnes sont respectivement 6 et 3. La figure 7.4 montre à la fois l'énumérateur de poids typique d'un code et celui de son dual lorsque ce code est tiré uniformément dans $\Omega[200,100,3,6]_2$. L'énumérateur de poids typique des équations de parité d'un code de $\Omega[200,100,3,6]_2$ a été obtenu grâce à l'identité de MacWilliams.

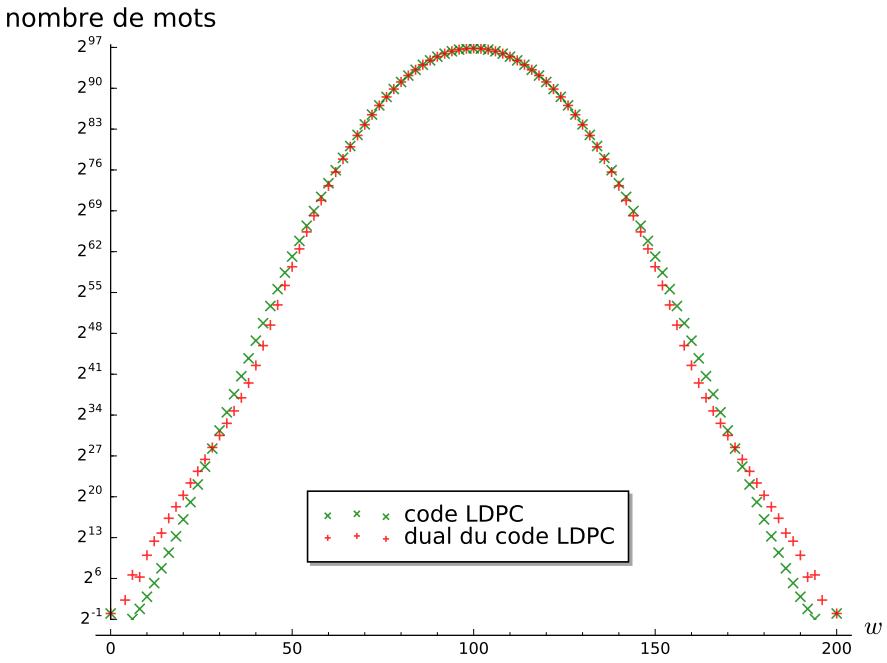


Figure 7.4 – Énumérateur de poids typique d'un code de $\Omega[200, 100, 3, 6]_2$ et de son dual.

Remarque 7.4.1. Les équations de parité d'un code de $\Omega[200, 100, 3, 6]_2$ sont nécessairement de poids pairs car elles sont générées par des équations de poids pair. C'est pourquoi sur la courbe de la figure 7.4, les valeurs sont nulles une fois sur deux.

7.5 Énumérateur de poids du dual d'un code LDPC

Dans la section précédente, nous avons construit le polynôme énumérateur de poids typique des équations de parité d'une famille Ω de codes LDPC. Pour cela, nous avons utilisé des séries génératrices ainsi que l'identité de MacWilliams. En procédant ainsi, nous n'obtenons pas de forme asymptotique de notre énumérateur de poids.

Nous pouvons réaliser le même type de démonstration que dans la section précédente sur un graphe modélisant non pas un code $\mathcal{C}$ de Ω mais plutôt son code dual. Cette façon de procéder permet d'une part d'obtenir une forme asymptotique de notre énumérateur de poids du code dual et d'autre part de pouvoir construire des énumérateurs de poids de sous-ensembles du dual définis par une répartition particulière des éléments du support.

Le graphe normal de Tanner $\mathcal{G}$ associé à une matrice de parité $\mathbf{H}$ d'un code $\mathcal{C}$ modélise le code $\mathcal{C}$; c'est-à-dire que l'on définit un mot du code directement à partir de $\mathcal{G}$. Dans [**?**], Forney montre que pour construire un graphe $\mathcal{G}^{\perp}$ modélisant l'espace dual de $\mathcal{C}$, il suffit de "dualiser" les nœuds logique du graphe $\mathcal{G}$. Par exemple, les nœuds de répétition $=$ sont remplacés par des nœuds de parité $+$ et inversement.

Les nœuds intermédiaires de degré 2 que nous avons représentés par des disques noirs • sont aussi des nœuds logiques puisque ce sont en fait des nœuds de répétition du même ordre que les nœuds représentés par le symbole $=$. Toutefois, un nœud de répétition de degré 2 se comporte exactement comme un nœud de parité de degré 2. Ainsi, "dualiser" les noeuds intermédiaires consiste simplement à les garder tels quels.

La figure 7.5 représente le graphe normal de Tanner modélisant le code généré par la

matrice $\mathbf{H}_0$ de la figure 7.2.

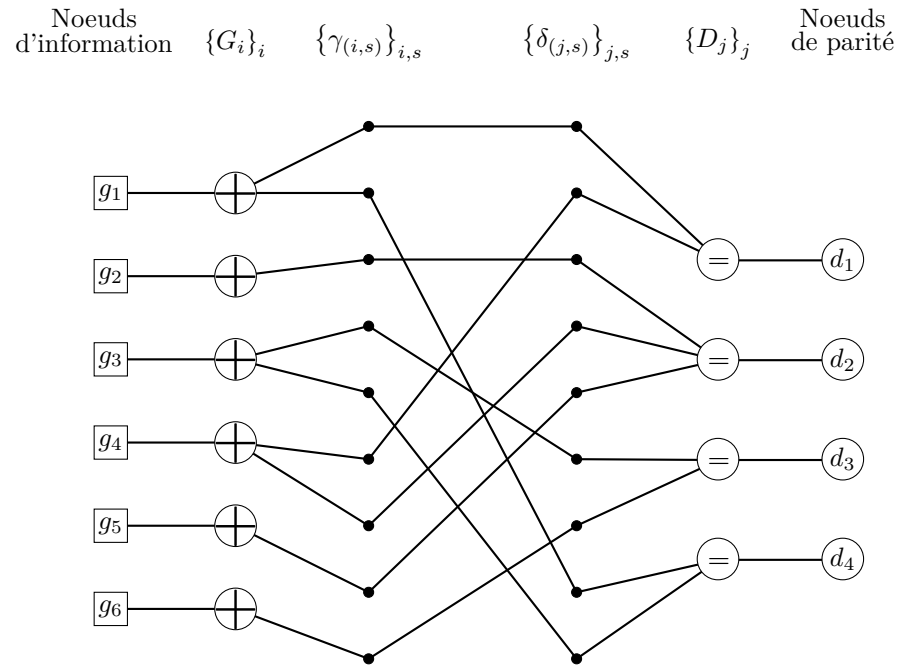
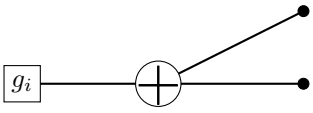


Figure 7.5 – Graphe normal de Tanner modélisant le code généré par la matrice $\mathbf{H}_0$ de la figure 7.2.

Pour simplifier la lecture, nous conservons les notations des nœuds du graphe $\mathcal{G}$ dans $\mathcal{G}^\perp$. Ainsi, les nœuds G_i sont maintenant des nœuds d'addition sur $\mathbb{F}_2$ et les nœuds D_j sont des nœuds de répétition. D'après la remarque précédente, les nœuds $\gamma_{(i,s)}$ et $\delta_{(j,s)}$ gardent leur nature d'origine.

Remarque 7.5.1. Tout comme une matrice de parité d'un code peut être vue comme une matrice génératrice de son dual, on peut voir le graphe normal de Tanner $\mathcal{G}^\perp$ comme un graphe générateur du code dual de $\mathcal{C}$. En effet, un mot $\mathbf{h} = (h_1, \cdots, h_n)$ est un mot du code dual de $\mathcal{C}$ si et seulement s'il existe une affectation des nœuds de parité $d_1, \cdots, d_{n-k}$ telle que tous les nœuds logiques de $\mathcal{G}^\perp$ soient satisfaits lorsque l'on affecte les valeurs $c_1, \cdots, c_n$ aux nœuds d'information $g_1, \cdots, g_n$. En fait, on peut même observer que l'affectation des nœuds de parité $d_1, \cdots, d_{n-k}$ correspond exactement au mot générant $\mathbf{h}$ via la matrice de parité $\mathbf{H}$.

Observation 1. Pour tout $i \in [\![1, n]\!]$, le sous-graphe de $\mathcal{G}^\perp$ composé du nœud de répétition G_i et de tous ses voisins modélise un code de parité que l'on note $\mathcal{C}_i^{\text{par}}$.



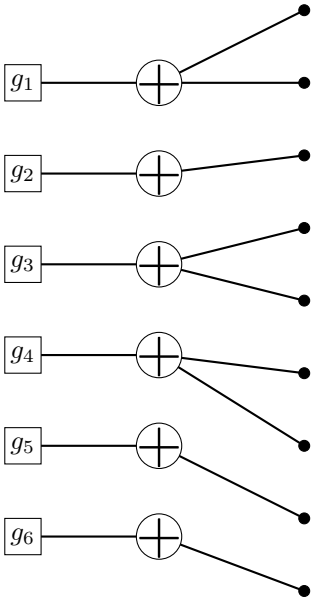
Un mot $(c_0, \cdots, c_{\alpha_i})$ de longueur $\alpha_i + 1$ est un mot du code $\mathcal{C}_i^{\text{par}}$ si et seulement si le nœud logique G_i est satisfait lorsque l'on affecte les valeurs c_0, c_1, $\cdots$, c_{α_i} aux nœuds respectifs g_i, $\gamma_{(i,1)}$, $\cdots$, $\gamma_{(i,\alpha_i)}$.

Le polynôme énumérateur de poids du code $\mathcal{C}_i^{\text{par}}$ est :

$$P_{\mathcal{C}_i^{\text{par}}}(X, Y) = \frac{(1+X)(1+Y)^{\alpha_i} + (1-X)(1-Y)^{\alpha_i}}{2} \tag{7.18}$$

Dans ce polynôme, le coefficient du monôme $X^{w_1}Y^{w_2}$ est le nombre de mots de $\mathcal{C}_i^{\text{par}}$ de poids w_1 sur le bit associé au nœud g_i et de poids w_2 sur les bits associés aux nœuds $\gamma_{(i,1)}$, $\cdots$, $\gamma_{(i,\alpha_i)}$.

Observation 2. Le sous-graphe de $\mathcal{G}^{\perp}$ composé de tous les graphes associés aux codes $\{\mathcal{C}_i^{\text{par}}\}_{i\in[\![1,n]\!]}$ modélise le produit cartésien de codes $\mathcal{C}^{\text{par}} := \mathcal{C}_1^{\text{par}} \times \cdots \times \mathcal{C}_n^{\text{par}}$.



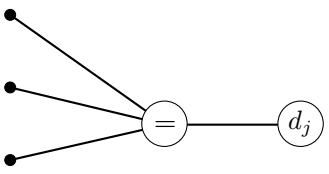
D'après la proposition 7.1.2, le polynôme énumérateur de poids du code $\mathcal{C}^{\text{par}}$ est :

$$P_{\mathcal{C}^{\text{par}}}(X,Y) = \prod_{i=1}^{n} P_{\mathcal{C}_i^{\text{par}}}(X,Y) \tag{7.19}$$

$$= \prod_{i=1}^{n} \frac{(1+X)(1+Y)^{\alpha_i} + (1-X)(1-Y)^{\alpha_i}}{2} \tag{7.20}$$

Dans ce polynôme, le coefficient du monôme $X^{w_1}Y^{w_2}$ est le nombre de mots de $\mathcal{C}^{\text{par}}$ de poids w_1 sur les bits associés aux nœuds $\{g_i\}_{i\in[\![1,n]\!]}$ et de poids w_2 sur les bits associés aux nœuds $\left\{\gamma_{(i,s)}\right\}_{(i,s)\in[\![1,n]\!]\times[\![1,\alpha_i]\!]}$.

Observation 3. Pour tout $j \in [\![1, n-k]\!]$, le sous-graphe de $\mathcal{G}^{\perp}$ composé du nœud d'addition D_j et de tous ses voisins modélise un code de répétition que l'on note $\mathcal{C}_j^{\text{rep}}$.



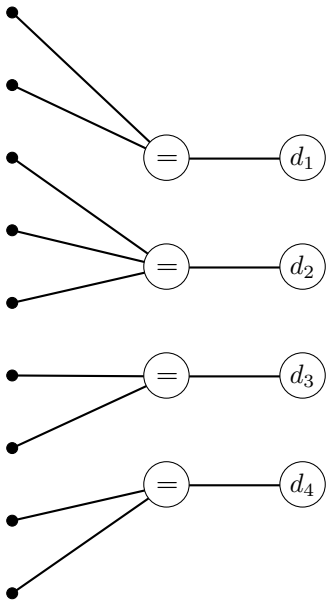
Un mot $(c_0, \cdots, c_{\beta_j})$ de longueur $\beta_j + 1$ est un mot du code $\mathcal{C}_j^{\text{rep}}$ si et seulement si le nœud logique D_j est satisfait lorsque l'on affecte les valeurs c_0, c_1, $\cdots$, c_{β_j} aux nœuds respectifs d_j, $\delta_{(j,1)}$, $\cdots$, $\delta_{(j,\beta_j)}$.

Le polynôme énumérateur de poids du code $\mathcal{C}_j^{\text{rep}}$ est :

$$P_{\mathcal{C}_j^{\text{rep}}}(Y,Z) = 1 + ZY^{\beta_j} \tag{7.21}$$

Dans ce polynôme, le coefficient du monôme $Y^{w_1}Z^{w_2}$ est le nombre de mots de $\mathcal{C}_j^{\text{rep}}$ de poids w_1 sur les bits associés aux nœuds $\delta_{(j,1)}, \cdots, \delta_{(j,\alpha_j)}$ et de poids w_2 sur le bit associé au nœud d_j .

Observation 4. Le sous-graphe de $\mathcal{G}^{\perp}$ composé de tous les graphes associés aux codes $\left\{\mathcal{C}_j^{\text{rep}}\right\}_{j\in[\![1,n-k]\!]}$ modélise le produit cartésien de codes $\mathcal{C}^{\text{rep}} := \mathcal{C}_1^{\text{rep}} \times \cdots \times \mathcal{C}_{n-k}^{\text{rep}}$.



D'après la proposition 7.1.2, le polynôme énumérateur de poids du code $\mathcal{C}^{\text{rep}}$ est :

$$\begin{aligned} P_{\mathcal{C}^{\text{rep}}}(Y,Z) &= \prod_{j=1}^{n-k} P_{\mathcal{C}_j^{\text{rep}}}(Y,Z) \\ &= \prod_{j=1}^{n-k} \left(1 + ZY^{\beta_j}\right) \end{aligned} \tag{7.22}$$

Dans ce polynôme, le coefficient du monôme $Y^{w_1}Z^{w_2}$ est le nombre de mots de $\mathcal{C}^{\text{rep}}$ de poids w_1 sur les bits associés aux nœuds $\left\{\delta_{(j,s)}\right\}_{(j,s)\in[\![1,n-k]\!]\times[\![1,\beta_j]\!]}$ et de poids w_2 sur les bits associés aux nœuds $\{d_j\}_{j\in[\![1,n-k]\!]}$.

Conclusion. Pour tout couple $(w,\ell) \in [\![1,n]\!]\times[\![1,n-k]\!]$, il y a coef $\left(P_{\mathcal{C}^{\text{par}}}(X,Y)\ ;\ X^wY^\ell\right)$ mots de $\mathcal{C}^{\text{par}}$ de poids w sur les bits associés aux nœuds d'information $\{g_i\}_i$ et de poids ℓ sur les bits associés aux nœuds $\{\gamma_{(i,s)}\}_{(i,s)}$. Pour chacun de ces mots, la probabilité pour que les bits associés aux nœuds $\{\gamma_{(i,s)}\}_{(i,s)}$ correspondent aux bits associés aux nœuds $\{\delta_{(j,s)}\}_{(j,s)}$ d'un mot du code $\mathcal{C}^{\text{rep}}$ est :

$$\frac{\text{coef}\left(P_{\mathcal{C}^{\text{rep}}}(Y,1)\ ;\ Y^\ell\right)}{\binom{N}{\ell}} \qquad \text{avec} \qquad N := \sum_{i=1}^{n}\alpha_i = \sum_{j=1}^{n}\beta_j \tag{7.23}$$

car les nœuds $\left\{\gamma_{(i,s)}\right\}_{(i,s)}$ et $\left\{\delta_{(j,s)}\right\}_{(j,s)}$ sont supposés être reliés aléatoirement. Dans l'équation (7.23), nous avons fixé la variable Z à 1. Cela permet d'ignorer le poids des bits associés aux nœuds $\{d_j\}_j$ dans les mots de $\mathcal{C}^{\text{rep}}$.

Finalement, l'espérance A_w du nombre d'équations de parité de poids w de $\mathcal{C}$ est :

$$A_w = \sum_{\ell=0}^{N} \frac{\text{coef}\left(P_{\mathcal{C}^{\text{par}}}(X,Y)\ ;\ X^wY^\ell\right)\cdot\text{coef}\left(P_{\mathcal{C}^{\text{rep}}}(Y,1)\ ;\ Y^\ell\right)}{\binom{N}{\ell}} \tag{7.24}$$

Nous remarquons que lorsque les poids $(\beta_j)_{j\in[\![1,n-k]\!]}$ des lignes de la matrice de parité $\mathbf{H}$ du code $\mathcal{C}$ sont constants égaux à β, le polynôme univarié $P_{\mathcal{C}^{rep}}(Y,1)$ devient :

$$P_{\mathcal{C}^{rep}}(Y,1) = \left(1+Y^{\beta}\right)^{n-k} \tag{7.25}$$

D'autre part, lorsque les poids $(\alpha_i)_{i\in[\![1,n]\!]}$ des colonnes de la matrice de parité $\mathbf{H}$ du code $\mathcal{C}$ sont constants égaux à α, le polynôme bivarié $P_{\mathcal{C}^{\text{par}}}(X,Y)$ devient quant à lui :

$$P_{\mathcal{C}^{\text{par}}}(X,Y) \;=\; \left(\frac{(1+X)(1+Y)^{\alpha}+(1-X)(1-Y)^{\alpha}}{2}\right)^{n} \tag{7.26}$$

Le théorème 7.5.1 résume notre résultat sur les énumérateurs de poids des équations de parité d'un code. Nous avons pu vérifier expérimentalement qu'il produit les mêmes énumérateurs de poids que la formule obtenue avec l'identité de MacWilliams.

Théorème 7.5.1. *Soit Ω la famille des codes linéaires binaires possédant une matrice de parité dont l'ensemble des poids des colonnes et des lignes sont respectivement $\{\alpha_i\}_i$ et $\{\beta_j\}_j$.*

L'espérance du nombre d'équations de parité de poids w d'un code choisi uniformément dans Ω est :

$$A_w = \sum_{\ell=0}^{N} \frac{\text{coef}\left(Q(X,Y)\;;\; X^wY^{\ell}\right)\cdot \text{coef}\left(P(Y)\;;\; Y^{\ell}\right)}{\binom{N}{\ell}} \tag{7.27}$$

avec :

$$N \;:=\; \sum_{i=1}^{n}\alpha_i = \sum_{j=1}^{n-k}\beta_j \tag{7.28}$$

$$P(Y) \;:=\; \prod_{j=1}^{n-k}\left(1+Y^{\beta_j}\right) \tag{7.29}$$

$$Q(X,Y) \;:=\; \prod_{i=1}^{n}\frac{(1+Y)^{\alpha_i}(1+X)+(1-Y)^{\alpha_i}(1-X)}{2} \tag{7.30}$$

Il est possible de déduire une forme asymptotique de l'énumérateur de poids du dual d'un code LDPC binaire à partir du théorème 7.5.1. Dans le théorème 7.5.2, nous proposons de développer les calculs dans le cas des codes LDPC réguliers. Notons toutefois que ce théorème se généralise aux cas non-réguliers.

Théorème 7.5.2. *Soit $\Omega\,[n,k,\alpha,\beta]_2$ la famille des codes LDPC binaires $[n,k]_2$ réguliers possédant une matrice de parité dont les colonnes sont toutes de poids α et les lignes de poids β. Nous supposons $k:=\lfloor Rn\rfloor$ pour une constante $R\in[0,1]$.*

Pour tout $\omega\in[0,1]$, l'espérance du nombre d'équations de parité de poids $w:=\lfloor\omega n\rfloor$ dans un code choisi uniformément dans $\Omega[n,k,\alpha,\beta]_2$ lorsque n tend vers l'infini est :

$$A_w = \tilde{O}(2^{n\rho}) \tag{7.31}$$

avec :

$$\rho := \sup_{\lambda\in[0,\alpha]}\left(\log_2\left(f(x_2,y_2)\right)+(1-R)h_2\left(\tfrac{\lambda}{(1-R)\beta}\right)-\alpha h_2\left(\tfrac{\lambda}{\alpha}\right)\right) \tag{7.32}$$

où $f(X,Y):=\dfrac{(1+Y)^{\alpha}(1+X)+(1-Y)^{\alpha}(1-X)}{2X^{\omega}Y^{\lambda}}$ et où x_2 et y_2 minimise f.

Démonstration du théorème 7.5.2.

Le nombre typique d'équations de parité de poids $w := \lfloor \omega n \rceil$ d'un code tiré uniformément dans $\Omega[n,k,\alpha,\beta]_2$ est :

$$\begin{aligned} A_w &= \sum_{\ell=0}^{\alpha n} \frac{\text{coef}\left((Q(X,Y))^n \ ; \ X^w Y^\ell\right) \cdot \text{coef}\left((P(Y))^{n-k} \ ; \ Y^\ell\right)}{\binom{\alpha n}{\ell}} \\ &= \tilde{O}\left(\sup_{\lambda \in [0,\alpha]} \frac{\text{coef}\left((Q(X,Y))^n \ ; \ X^{\lfloor \omega n \rfloor} Y^{\lfloor \lambda n \rfloor}\right) \cdot \text{coef}\left((P(Y))^{n-k} \ ; \ Y^{\left\lfloor \frac{\lambda}{1-R}(n-k) \right\rfloor}\right)}{2^{\alpha n h_2\left(\frac{\lambda}{\alpha}\right)}} \right) \end{aligned}$$

avec $P(Y) = 1 + Y^\beta$ et $Q(X,Y) = \dfrac{(1+Y)^\alpha(1+X) + (1-Y)^\alpha(1-X)}{2}$.

Les coefficients de P sont positifs. Ainsi, pour tout $y > 0$ et tout entier $s \geqslant 0$, on a :

$$(P(y))^{n-k} \geqslant \text{coef}\left((P(Y))^{n-k} \ ; \ Y^s\right) \cdot y^s$$

et donc notamment :

$$\text{coef}\left((P(Y))^{n-k} \ ; \ Y^{\left\lfloor \frac{\lambda}{1-R}(n-k) \right\rfloor}\right) \leqslant \left(\inf_{y>0} \left(\frac{P(y)}{y^{\frac{\lambda}{1-R}}} \right) \right)^{n-k}$$

Une analyse différentielle montre que la borne inférieure de $\frac{P(y)}{y^{\frac{\lambda}{1-R}}}$ est atteinte en $y_1 := \left(\frac{\lambda}{\beta(1-R)-\lambda}\right)^{\frac{1}{\beta}}$, ce qui nous donne alors :

$$\begin{aligned} \text{coef}\left((P(Y))^{n-k} \ ; \ Y^{\left\lfloor \frac{\lambda}{1-R}(n-k) \right\rfloor}\right) &= \tilde{O}\left(\left(\frac{P(y_1)}{y_1^{\frac{\lambda}{1-R}}} \right)^{n-k} \right) \\ &= \tilde{O}\left(\left(\left(\frac{\beta(1-R)}{\beta(1-R)-\lambda} \right) \cdot \left(\frac{\beta(1-R)-\lambda}{\lambda} \right)^{\frac{\lambda}{(1-R)\beta}} \right)^{n-k} \right) \\ &= \tilde{O}\left(2^{(n-k)h_2\left(\frac{\lambda}{(1-R)\beta}\right)} \right) \end{aligned}$$

Déterminons maintenant une approximation asymptotique de $\text{coef}\left((Q(X,Y))^n \ ; \ X^w Y^\ell\right)$. Les coefficients de Q sont tous positifs. Ainsi, pour tous $x, y > 0$ et pour tous entiers s et t, on a :

$$(Q(X,Y))^n \geqslant \text{coef}\left((Q(X,Y))^n \ ; \ X^s Y^t\right) \cdot X^s Y^t$$

et donc notamment :

$$\text{coef}\left((Q(X,Y))^n \ ; \ X^{\lfloor \omega n \rfloor} Y^{\lfloor \lambda n \rfloor}\right) \leqslant \left(\inf_{x,y>0} \left(\frac{Q(x,y)}{x^\omega y^\lambda} \right) \right)^n$$

Nous utilisons alors une méthode numérique pour trouver les valeurs x_2 et y_2 qui minimisent $f(x,y) := \dfrac{Q(x,y)}{x^\omega y^\lambda}$.

Les travaux de Gardy et Solé dans [**?**] et ceux de Good dans [**?**] montrent que lorsque n tend vers l'infini, la borne du théorème est fine.

□

Remarque 7.5.2. Dans le théorème 7.5.2, la fonction f est de classe $\mathscr{C}^2$. Nous pouvons donc minimiser cette fonction de façon efficace grâce à des méthodes d'optimisation numériques de descente de gradient (par exemple la méthode de Newton).

Chapitre 8

Les codes $U|U+V$ récursifs et les codes polaires

Les codes polaires ont été inventés par Arıkan il y a maintenant plus de 10 ans [Arı09]. Ils sont présents dans la $5^{\text{ème}}$ génération des standards pour la téléphonie mobile. Ils ont été choisis pour permettre d'augmenter les débits et l'efficacité spectrale des réseaux cellulaires actuels tout en continuant à corriger les erreurs de transmission.

Les codes polaires sont très appréciés pour leurs propriétés de correction : ils sont capables d'atteindre la capacité des canaux symétriques sans mémoire. De plus, ils possèdent des algorithmes de codage et de décodage peu coûteux. Nous expliciterons notamment le décodage par annulation successive (une version revisitée de [Kor09]) qui s'exécute en un temps de l'ordre de $O(n \log_2(n))$ où n est la longueur du code. Ce décodage retourne un mot de code dont la probabilité qu'il ait été émis est relativement grande. Mais ce n'est pas le mot de code émis *le plus* probable. En 2012, Tal et Vardy ont proposé un décodage en liste des codes polaires qui permet de se rapprocher un peu plus du mot de code le plus probablement émis [TV15]. Son coût est de l'ordre de $O(\ell n \log(n))$ où ℓ est la taille de la liste retournée par le décodeur.

Dans la suite de cette thèse, nous allons utiliser les codes polaires et le décodeur en liste de Tal et Vardy pour construire une fonction de hachage floue et ainsi améliorer la recherche de couples d'éléments proches dans un ensemble.

Dans cette section, nous commençons par donner une construction originale des codes polaires. Celle-ci nous amènera notamment à une généralisation de ces codes qui pourrait permettre d'en améliorer encore les performances sans impacter significativement la complexité de leur décodage. En outre, pour construire des codes polaires performants, il est courant d'utiliser des simulations et de se baser sur des statistiques. Cette façon de procéder peut être parfois coûteuse. Nous verrons que notre construction, en se basant sur des calculs de probabilités, permet de produire des codes polaires performants plus efficacement.

8.1 Définition d'un code $U|U+V$ récursif

La structure de base des codes polaires est celle des codes $U|U+V$. Nous commençons par définir ces derniers :

Définition 8.1.1 (code $U|U+V$)**.** *Soient U et V deux codes binaires linéaires de longueur n. Le code $U|U+V$ est le code de longueur $2n$ définit par :*

$$U|U+V = \{(\mathbf{u}, \mathbf{u}+\mathbf{v}) \text{ tel que } \mathbf{u} \in U \text{ et } \mathbf{v} \in V\}$$

Notons qu'un code $U|U+V$ est de rendement $R = R_U + R_V$ où R_U et R_V sont les rendements respectifs de U et V.

Les codes que nous considérons ici sont des constructions récursives de codes $U|U+V$.

Définition 8.1.2 (code $U|U+V$ récursif)**.** *Un code $U|U+V$ récursif U_ε de profondeur d et de longueur $2^d n$ est obtenu à partir d'un arbre binaire complet $\mathcal{T}$ (c'est-à-dire que tous les nœuds internes ont exactement deux fils) et un ensemble de codes $U_{\mathbf{x}}$ associés à chaque nœud $\mathbf{x}$ de l'arbre binaire. Ici, $\mathbf{x}$ est vu comme un mot binaire encodant le chemin à suivre depuis la racine ε pour atteindre le nœud qu'il désigne (aller vers un fils gauche est encodé par un 0 tandis qu'aller vers un fils droit est encodé par un 1). La longueur du mot $\mathbf{x}$ (qui est aussi la longueur du chemin reliant le nœud $\mathbf{x}$ à la racine ε) est notée t. La longueur du code $U_{\mathbf{x}}$ est donc $2^{d-t}n$.*

U_ε est défini récursivement par :

$$\begin{cases} U_\varepsilon & := & U_0|U_0+U_1 \\ U_{\mathbf{x}} & := & U_{(\mathbf{x},0)}|U_{(\mathbf{x},0)}+U_{(\mathbf{x},1)} \end{cases} \tag{8.1}$$

Les codes $U_{\mathbf{x}}$ tels que $\mathbf{x}$ est une feuille de l'arbre sont appelés les codes constituants*.*

Notation 8.1.3. *Rappelons que pour tout $\mathbf{x} \in \mathbb{F}_2^t$ et tout $b \in \mathbb{F}_2$, nous notons $(\mathbf{x}, b)$ la concaténation du vecteur $\mathbf{x}$ et du bit b. Le vecteur ainsi produit est de longueur $t+1$.*

Un code $U|U+V$ est graphiquement représenté par un nœud avec deux fils : les fils de gauche et droite représentent respectivement les codes constituants U et V. La figure 8.1 représente cet arbre (les codes constituants sont représentés en rouge).

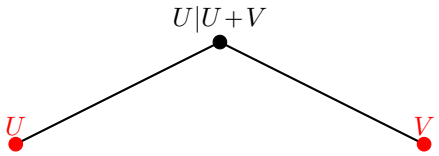


Figure 8.1 – Représentation d'un code $U|U+V$.

Un autre exemple est donné dans la figure 8.2 et représente graphiquement un code $U|U+V$ récursif U_ε de profondeur 3. Dans cet exemple, le code est composé de 6 codes constituants (en rouge sur la figure) :

$$U_{000}, U_{001}, U_{01}, U_{10}, U_{110} \text{ et } U_{111}$$

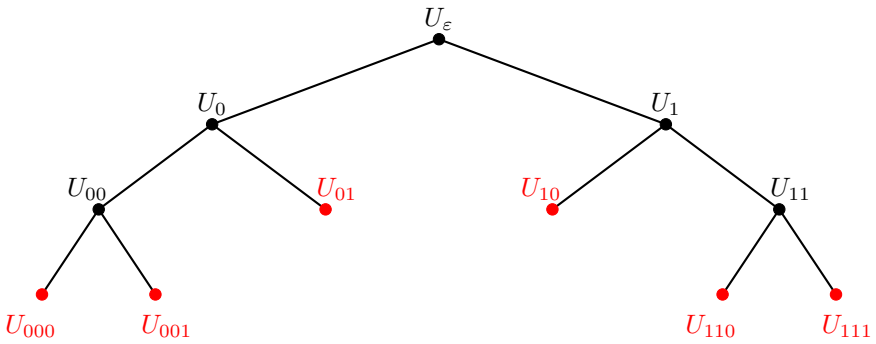


Figure 8.2 – Représentation d'un code $U|U+V$ récursif de profondeur 3.

8.2 Modélisation des canaux associés à un code $U|U+V$ récursif

Nous avons vu qu'un code $U|U+V$ récursif est défini par un arbre et un ensemble de codes constituants. Cependant, nous n'avons pas encore proposé de construction précise de cet arbre ni de définition précise des codes constituants. Autrement dit, nous ne savons pas où placer les feuilles dans l'arbre et encore moins quels codes constituants leur associer. Ce paragraphe permet de comprendre l'impact des codes $U|U+V$ récursifs sur le canal de transmission et ainsi nous aider à définir des codes constituants optimaux.

Supposons que chaque bit d'un mot d'un code $U|U+V$ est transmis via un canal binaire symétrique. Un bit transmis est alors soit un bit d'un mot du code U, soit la somme d'un bit d'un mot de U et d'un bit d'un mot de V. Par exemple, notons respectivement u et v deux bits des codes U et V et supposons que les bits u et $u+v$ sont tous les deux transmis via un canal binaire symétrique. Les bits reçus sont alors respectivement notés y_1 et y_2. Le bit $y_1 + y_2$ est alors une altération du bit v. Nous avons ainsi modélisé un canal transmettant le bit v mais celui-ci est dégradé par rapport au canal considéré à l'origine. Quel est alors l'intérêt de considérer ce canal moins performant ? La réponse est la suivante : après avoir retrouvé v, nous pouvons modéliser un canal plus performant pour la transmission du bit u. En effet, connaissant la valeur de v, il est plus facile de retrouver u car nous avons deux versions altérées indépendantes de ce bit : y_1 d'une part et $y_2 + v$ de l'autre. Cette procédure consistant à détériorer un canal pour en améliorer un autre est appelé *polarisation*. Nous allons à présent formaliser cette notion et l'appliquer aux codes $U|U+V$ récursifs.

Notation 8.2.1. *Notons* $\mathrm{BSC}(p)[r]$*, un canal binaire symétrique de probabilité d'erreur* $p \in [0,1]$*. Le réel* $r \in [0,1]$ *représente la proportion de bits empruntant ce canal.*

Si les bits d'un mot d'un code $U|U+V$ récursif sont transmis via un canal binaire symétrique (éventuellement de probabilités d'erreurs différentes pour chaque bit transmis), alors les canaux produits par polarisations successives ne sont pas des canaux binaires symétriques mais des compositions de canaux binaires symétriques que nous définissons ainsi :

Définition 8.2.2 (BSC-composition)**.** *Soit* N *un entier positif. Un ensemble* $\left\{ \mathrm{BSC}(p_i)[r_i] \right\}_{i \in [\![1,N]\!]}$ *tels que la somme des* r_i *vaut* 1 *est appelé BSC-composition.*

Un mot binaire de longueur n *qui est transporté par cette BSC-composition a en moyenne* $r_i n$ *bits qui sont transportés par le canal binaire symétrique de probabilité d'erreur* p_i*.*

La capacité moyenne de la *BSC*-composition $\left\{ \mathrm{BSC}(p_i)[r_i] \right\}_{i \in [\![1,N]\!]}$ est simplement la moyenne pondérée par les r_i de toutes les capacités des canaux binaires symétriques $\mathrm{BSC}(p_i)$:

Propriété 8.2.3. *Soit* N *un entier positif et soit* $\Upsilon := \left\{ \mathrm{BSC}(p_i)[r_i] \right\}_{i \in [\![1,N]\!]}$ *une BSC-composition. La capacité moyenne de* Υ *est :*

$$C_\Upsilon = \sum_{\mathrm{BSC}(p)[r] \in \Upsilon} r \cdot C_{\mathrm{BSC}(p)} \tag{8.2}$$

où $C_{\mathrm{BSC}(p)} := 1 - h_2(p)$ *est la capacité du canal binaire symétrique de probabilité d'erreur* p *(cf. corollaire 1.3.8).*

Nous définissons l'opérateur $\boxplus$ qui associe deux objets de type $\mathrm{BSC}(\cdot)[\cdot]$:

Définition 8.2.4. *L'opérateur $\boxplus$ associe deux BSC de la manière suivante :*

$$\mathrm{BSC}(p)[r] \ \boxplus \ \mathrm{BSC}(q)[s] \ := \ \mathrm{BSC}(p+q-2pq)[rs] \tag{8.3}$$

Proposition 8.2.5. *Soit u et $u+v$ deux bits transmis respectivement via un $\mathrm{BSC}(p)[1]$ et un $\mathrm{BSC}(q)[1]$. Les bits reçus correspondant sont notés y_1 et y_2.*

Le bit y_1+y_2 correspond au bit v transmis via le canal $\mathrm{BSC}(p)[1] \boxplus \mathrm{BSC}(q)[1]$.

Démonstration de la proposition 8.2.5.

$$\begin{array}{rcl}
\mathbb{P}(y_1+y_2 \ = \ v) & = & \mathbb{P}((y_1=u \text{ et } y_2=u+v) \text{ ou } (y_1\neq u \text{ et } y_2\neq u+v)) \\
& = & \mathbb{P}(y_1=u)\,\mathbb{P}(y_2=u+v)+\mathbb{P}(y_1\neq u)\,\mathbb{P}(y_2\neq u+v) \\
& = & (1-p)(1-q)+pq \\
& = & 1-p-q+2pq
\end{array}$$

□

Nous pouvons généraliser la définition 8.2.4 et la proposition 8.2.5 aux BSC-compositions :

Définition 8.2.6. *L'opérateur $\boxplus$ associe deux BSC-compositions de la manière suivante :*

$$\begin{array}{l}
\left\{\mathrm{BSC}(p_i)[r_i]\right\}_{i\in[\![1,N]\!]} \ \boxplus \ \left\{\mathrm{BSC}(q_j)[s_j]\right\}_{j\in[\![1,M]\!]} \\
\qquad := \left\{\mathrm{BSC}(p_i)[r_i] \ \boxplus \ \mathrm{BSC}(q_j)[s_j]\right\}_{(i,j)\in[\![1,N]\!]\times[\![1,M]\!]}
\end{array} \tag{8.4}$$

Corollaire 8.2.7. *Soit u (*resp. *v) un bit transmis via une BSC-composition Υ_1 (*resp. *Υ_2). Le bit reçu correspondant est noté $\widetilde{u}$ (*resp. *$\widetilde{v}$).*

Le bit $\widetilde{u}+\widetilde{v}$ correspond au bit $u+v$ transmis via le canal $\Upsilon_1 \boxplus \Upsilon_2$.

Nous définissons à présent un second opérateur sur les $\mathrm{BSC}(\cdot)[\cdot]$. Cet opérateur ne retourne pas un simple BSC mais un couple de BSC :

Définition 8.2.8. *L'opérateur $\boxtimes$ associe deux BSC de la manière suivante :*

$$\begin{array}{l}
\mathrm{BSC}(p)[r] \ \boxtimes \ \mathrm{BSC}(q)[s] \\
\qquad := \left\{ \begin{array}{l} \mathrm{BSC}\left(\frac{pq}{pq+(1-p)(1-q)}\right)[rs(1-p-q+2pq)] \\ \mathrm{BSC}\left(\frac{p(1-q)}{p(1-q)+q(1-p)}\right)[rs(p+q-2pq)] \end{array} \right\}
\end{array} \tag{8.5}$$

Remarque 8.2.1. Le résultat de l'opération $\mathrm{BSC}(p)[r] \boxtimes \mathrm{BSC}(q)[s]$ n'est une BSC-composition que si $r=s=1$.

Proposition 8.2.9. *Soient deux bits u et v. u est transmis via un $\mathrm{BSC}(p)[1]$ tandis que $u+v$ est transmis via un $\mathrm{BSC}(q)[1]$.*

Cette procédure équivaut à transmettre v via le canal $\mathrm{BSC}(p)[1] \boxplus \mathrm{BSC}(q)[1]$ et u via le canal $\mathrm{BSC}(p)[1] \boxtimes \mathrm{BSC}(q)[1]$.

Démonstration de la proposition 8.2.9.

Notons y_1 le bit reçu correspondant au bit u émis via le canal $\mathrm{BSC}(p)[1]$ et y_2 le bit reçu correspondant au bit $u+v$ émis via le canal $\mathrm{BSC}(q)[1]$.
Tout d'abord, d'après la proposition 8.2.5, v est transmis via le canal $\mathrm{BSC}(p)[1] \boxplus \mathrm{BSC}(q)[1]$. Le bit v est alors retrouvé en décodant $y_1 + y_2$.
D'autre part, maintenant que nous connaissons v, nous pouvons considérer que le bit u est transmis à la fois via le canal $\mathrm{BSC}(p)[1]$ et le canal $\mathrm{BSC}(q)[1]$ de façon indépendante. Les deux bits reçus correspondant sont y_1 et $y_2 + v$. Notons :

$$p'(i) := \mathbb{P}(u = 1 \mid y_1 = i) = \begin{cases} p & \text{si } i = 0 \\ 1-p & \text{si } i = 1 \end{cases} \tag{8.6}$$

et

$$q'(j) := \mathbb{P}(u = 1 \mid y_2 + v = j) = \begin{cases} q & \text{si } j = 0 \\ 1-q & \text{si } j = 1 \end{cases} \tag{8.7}$$

avec i et $j \in \mathbb{F}_2$. Nous avons alors :

$$\mathbb{P}(u = 1 \mid y_1 = i\ ,\ y_2 + v = j) = \frac{p'(i)q'(j)}{p'(i)q'(j) + (1-p'(i))(1-q'(j))} \tag{8.8}$$

En traitant les différentes valeurs possibles de i et j, nous avons :

$$\mathbb{P}(u = y_1 \mid y_1 + y_2 = v) = \frac{(1-p)(1-q)}{pq + (1-p)(1-q)} \tag{8.9}$$

$$\mathbb{P}(u \neq y_1 \mid y_1 + y_2 = v) = \frac{pq}{pq + (1-p)(1-q)} \tag{8.10}$$

$$\mathbb{P}(u = y_1 \mid y_1 + y_2 \neq v) = \frac{(1-p)q}{pq + (1-p)(1-q)} \tag{8.11}$$

$$\mathbb{P}(u \neq y_1 \mid y_1 + y_2 \neq v) = \frac{p(1-q)}{pq + (1-p)(1-q)} \tag{8.12}$$

Or comme $y_1 + y_2$ est une altération du bit v après qu'il ait été transporté par le canal $\mathrm{BSC}(p)[1] \boxplus \mathrm{BSC}(q)[1]$, nous savons d'après la proposition 8.2.5 que :

$$\mathbb{P}(y_1 + y_2 \neq v) = p + q - 2pq \tag{8.13}$$

Nous pouvons alors en déduire la proposition. □

Nous pouvons généraliser la définition 8.2.8 et la proposition 8.2.9 aux BSC-compositions :

Définition 8.2.10. *L'opérateur $\boxtimes$ associe deux BSC-compositions de la manière suivante :*

$$\begin{aligned} \left\{ \mathrm{BSC}(p_i)[r_i] \right\}_{i \in [\![1,N]\!]} \ \boxtimes\ \left\{ \mathrm{BSC}(q_j)[s_j] \right\}_{j \in [\![1,M]\!]} \\ := \left\{ \mathrm{BSC}(p_i)[r_i] \ \boxtimes\ \mathrm{BSC}(q_j)[s_j] \right\}_{(i,j) \in [\![1,N]\!] \times [\![1,M]\!]} \end{aligned} \tag{8.14}$$

Remarque 8.2.2. Soient Υ_1 et Υ_2, deux BSC-compositions. Le résultat de l'opération $\Upsilon_1 \boxtimes \Upsilon_2$ est toujours une BSC-composition.

Corollaire 8.2.11. *Soient deux bits u et v. Le bit u est transmis via une BSC-composition Υ_1 tandis que $u+v$ est transmis via une BSC-composition Υ_2.*

Cette procédure équivaut à transmettre v via la BSC-composition $\Upsilon_1 \boxplus \Upsilon_2$ et u via la BSC-composition $\Upsilon_1 \boxtimes \Upsilon_2$.

Soit U_ε un code $U|U+V$ récursif associé à un arbre binaire $\mathscr{T}$ défini comme dans la définition 8.1.2. Nous supposons que les mots du code U_ε sont altérés par un canal binaire symétrique sans mémoire de probabilité d'erreur p. Notons $\Upsilon_{\mathbf{x}}$ le canal via lequel

transitent les bits d'un mot du code $U_{\mathbf{x}}$ associé au nœud $\mathbf{x}$ de $\mathscr{T}$. Les canaux $\Upsilon_{\mathbf{x}}$ sont définis récursivement :

$$\Upsilon_{\varepsilon} = \mathrm{BSC}(p)\,[1] \tag{8.15}$$

$$\Upsilon_{(\mathbf{x},1)} = \Upsilon_{\mathbf{x}} \boxplus \Upsilon_{\mathbf{x}} \tag{8.16}$$

$$\Upsilon_{(\mathbf{x},0)} = \Upsilon_{\mathbf{x}} \boxtimes \Upsilon_{\mathbf{x}} \tag{8.17}$$

La figure 8.3 décrit alors la décomposition du canal lors du décodage du code $U|U+V$ de l'exemple de la figure 8.2.

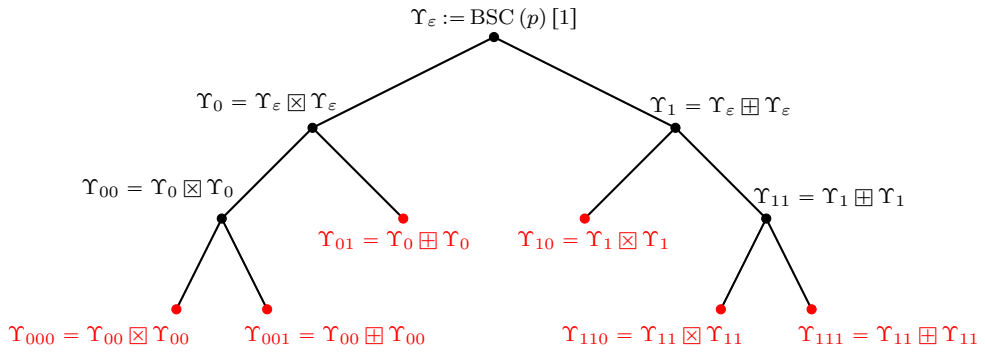


Figure 8.3 – Arbre binaire représentant les différents canaux d'un code $U|U+V$ récursif.

Détermination des canaux par simulation. Le cardinal d'une BSC-composition $\Upsilon_{\mathbf{x}}$ peut croître de façon exponentielle en fonction de la profondeur du nœud $\mathbf{x}$. C'est pourquoi en pratique, nous utilisons rarement ces formules mais estimons les canaux par simulation. Celle-ci consiste essentiellement à simuler l'envoie du mot nul dans un canal binaire symétrique de probabilité d'erreur p. Nous pouvons alors mesurer statistiquement la probabilité d'erreur des canaux virtuels associés aux feuilles de l'arbre. Si le canal associé à la feuille $\mathbf{x}$ a une probabilité d'erreur de transmission égale à $p_{\mathbf{x}}$, alors ce canal a une capacité de transmission égale à $C_{\mathbf{x}} := 1 - h_2(p_{\mathbf{x}})$. Nous choisissons alors le code constituant $U_{\mathbf{x}}$ de rendement $R_{\mathbf{x}} = C_{\mathbf{x}}$.

Cette construction est parfois coûteuse : elle demande de nombreuses simulations pour être suffisamment précise.

Détermination des canaux par des calculs de probabilités. Il est possible de procéder autrement que par simulation pour déterminer une approximation fiable des capacités des canaux associés aux nœuds de l'arbre d'un code $U|U+V$ récursif. Nous sommes en théorie capable de décrire parfaitement ces canaux comme des BSC-compositions, cependant, celles-ci sont composées de trop d'éléments et les manipuler est donc extrêmement coûteux en temps et en mémoire. Pour limiter cela, nous allons approcher les BSC-compositions par des BSC-compositions contenant un nombre limité de BSC. Concrètement, soit Υ une BSC-composition de taille N. Nous commençons par fixer un entier M qui sera la taille maximale de l'approximation de Υ. Pour tout entier $t \in [\![0, M-1]\!]$, nous isolons l'ensemble des BSC de la BSC-composition dont la probabilité d'erreur est dans l'intervalle

$\left[\frac{t}{M}, \frac{t+1}{M}\right]$:

$$\Upsilon(t) := \left\{\mathrm{BSC}\,(p)\,[r] \in \Upsilon \;:\; p \in \left[\tfrac{t}{M}, \tfrac{t+1}{M}\right]\right\} \tag{8.18}$$

Nous approchons alors $\Upsilon(t)$ par :

$$\Upsilon(t) \simeq \widetilde{\Upsilon}(t) := \mathrm{BSC}\left(\tfrac{t}{M} + \tfrac{1}{2M}\right)[R] \quad \text{où} \quad R := \sum_{\mathrm{BSC}(p)[r] \in \Upsilon(t)} r \tag{8.19}$$

Et donc par extension, nous approchons Υ par la BSC-composition suivante :

$$\Upsilon \simeq \left\{\widetilde{\Upsilon}(t)\right\}_{t \in [\![0, M-1]\!]} \tag{8.20}$$

Finalement, l'algorithme 8.2.1 permet d'estimer avec un précision de $\pm\frac{1}{2M}$ les canaux composant un code $U|U+V$ récursif U_ε dont les mots sont transportés par un canal binaire symétrique de probabilité d'erreur p. Dans cet algorithme, nous nous ramenons souvent aux notations données dans la définition 8.1.2.

Algorithme 8.2.1 : Description des canaux composant un $U|U+V$ récursif

Entrées : la profondeur d du code U_ε ;
la longueur $2^d n$ du code U_ε ;
la probabilité d'erreur $p \in \left[0, \frac{1}{2}\right]$.
Paramètre : une constante d'approximation $M \in \mathbb{N}$.
Sortie : pour chaque nœud $\mathbf{x}$, la description du canal $\Upsilon_{\mathbf{x}}$ transportant les mots du code $U_{\mathbf{x}}$.

$\Upsilon_\varepsilon \leftarrow \mathrm{BSC}\,(p)\,[1]$; `/* le symbole ε représente la chaine vide */`
pour tout $i \in [\![1, d]\!]$ **faire**
  $\mathscr{L}_i \leftarrow \varnothing$;
  pour tout $\mathbf{x} \in \mathbb{F}_2^{i-1}$ **faire**
    $\Upsilon_{(\mathbf{x},1)} \leftarrow \Upsilon_{\mathbf{x}} \boxplus \Upsilon_{\mathbf{x}}$;
    $\Upsilon_{(\mathbf{x},1)} \leftarrow$ SIMPLIFIER$(\Upsilon_{(\mathbf{x},1)}, M)$;
    $\Upsilon_{(\mathbf{x},0)} \leftarrow \Upsilon_{\mathbf{x}} \boxplus \Upsilon_{\mathbf{x}}$;
    $\Upsilon_{(\mathbf{x},0)} \leftarrow$ SIMPLIFIER$(\Upsilon_{(\mathbf{x},0)}, M)$;
  finPour
finPour
retourner tous les $\Upsilon_{\mathbf{x}}$ où $\mathbf{x} \in \bigcup_{i=0}^{d} \mathbb{F}_2^i$;

Fonction *SIMPLIFIER(Υ, M)*
  trier les éléments de Υ par ordre croissant des probabilités d'erreur ; `/* ce tri permet de parcourir les deux boucles suivantes en un temps linéaire en la taille de Υ */`
  $\widetilde{\Upsilon} \leftarrow \varnothing$;
  pour tout $t \in [\![0, M-1]\!]$ **faire**
    $R \leftarrow 0$;
    pour tout $\mathrm{BSC}\,(p)\,[r] \in \Upsilon$ *tels que* $p \in \left[\frac{t}{M}, \frac{t+1}{M}\right]$ **faire**
      $R \leftarrow R + r$;
    finPour
    $\widetilde{\Upsilon} \leftarrow \widetilde{\Upsilon} \cup \mathrm{BSC}\left(\frac{t}{M} + \frac{1}{2M}\right)[R]$;
  finPour
  retourner $\widetilde{\Upsilon}$;
finFonction

8.3 Construction d'un code $U|U+V$ récursif

Commençons par noter qu'un code $U|U+V$ récursif n'est linéaire que si ses codes constituants le sont aussi. Dans la suite, nous supposons que c'est effectivement le cas. Nous pouvons alors parler de dimension du code. La dimension d'un code $U|U+V$ récursif est la somme des dimensions de ces codes constituants. La question est donc de savoir quelles dimensions choisir pour les codes constituants. L'algorithme 8.2.1 permet de décrire les canaux via lesquels transiteront les mots des codes constituants. Grâce à la proposition 8.2.3, nous pouvons calculer leurs capacités. Ainsi, pour un code constituant $U_{\mathbf{x}}$, nous notons $C_{\mathbf{x}}$ la capacité du canal $\Upsilon_{\mathbf{x}}$ qui lui est associée.

Dans la section **??** nous avons mentionné le deuxième théorème de Shannon qui affirme que pour un canal discret sans mémoire de capacité C et pour tout $R < C$, il existe une suite de codes $\mathcal{C}_n$ de longueur n dont le rendement converge vers R et qui permet à la limite de corriger toutes les erreurs de transmission. Typiquement, un code tiré uniformément parmi tous les codes de bon rendement répondra relativement bien au second théorème de Shannon. Une première stratégie consiste donc à définir les codes constituants $U_{\mathbf{x}}$ comme des codes aléatoires de rendement égal à la capacité $C_{\mathbf{x}}$ du canal $\Upsilon_{\mathbf{x}}$. Un décodage exhaustif des codes constituants n'est alors possible que si les dimensions ou les codimensions de ces codes sont faibles. Ainsi, avec cette stratégie, les codes constituants sont les codes $U_{\mathbf{x}}$ tels que $C_{\mathbf{x}}$ soit proche de 1 ou de 0.

Par exemple, l'algorithme 8.3.1 décrit un générateur de codes $U|U+V$ récursifs de longueur 2^d dont les codes constituants sont des codes aléatoires. Dans cet exemple, les feuilles de $\mathscr{T}$ sont les nœuds $\mathbf{x}$ tels que $C_{\mathbf{x}}2^{d-t}$ ou $(1-C_{\mathbf{x}})2^{d-t}$ soit inférieure à une borne Γ (t dénotant la profondeur du nœud $\mathbf{x}$). Chaque code constituant $U_{\mathbf{x}}$ est alors défini comme un code aléatoire de longueur 2^{d-t} et de rendement $C_{\mathbf{x}}$.

Algorithme 8.3.1 : Générateur d'un code U | U + V récursif dont les codes constituants sont des codes aléatoires

Entrées : la longueur $n := 2^d$ du code à construire ;
la probabilité d'erreur p du canal de transmission.
Paramètre : un entier Γ ;
Sortie : un arbre binaire $\mathscr{T}$ et un ensemble de codes constituants $\{U_{\mathbf{x}}\}$ représentant un code U | U + V récursif où les codes constituants sont de dimension ou codimension inférieure à Γ.

```
1 T ⟵ {ε} ;                                   /* ε dénote la chaîne binaire vide */
2 E_CC ⟵ ∅ ;
3 Υ_x ⟵ {BSC(p)[1]} ;
4 GenRecursif(T, E_CC, 0, ε, Υ_ε) ;
5 retourner T et E_CC
```

```
1  Fonction GenRecursif(T, E_CC, t, x, Υ_x)
2      C_x ⟵ capacité de Υ_x ;              /* donnée par la proposition 8.2.3 */
3      si (C_x × 2^{d−t} ⩽ Γ) ou ((1 − C_x) × 2^{d−t} ⩽ Γ) alors
4          U_x ⟵ un code aléatoire de longueur 2^{d−t} et de rendement C_x ;
5          ajouter U_x dans E_CC ;
6          retourner ;
7      sinon
8          ajouter le fils gauche (x, 0) au nœud x de l'arbre T ;
9          ajouter le fils droit (x, 1) au nœud x de l'arbre T ;
10         GenRecursif(T, E_CC, t + 1, (x, 0), Υ_x ⊠ Υ_x) ;
11         GenRecursif(T, E_CC, t + 1, (x, 1), Υ_x ⊞ Υ_x) ;
12         retourner ;
13     finSi
14 finFonction
```

L'algorithme 8.3.1 décrit la construction d'un code $U|U+V$ récursif de rendement optimal pour un canal binaire symétrique de probabilité d'erreur p donnée. Nous pouvons être amenés à poser le problème de la construction d'un code $U|U+V$ récursif autrement. En effet, étant donné un rendement R, nous voulons construire le code $U|U+V$ récursif de longueur n qui optimise la capacité de correction. Pour cela, il suffit d'effectuer la même construction que précédemment mais en choisissant p comme étant la distance de Gilbert-Varshamov relative du code :

$$p := h_2^{-1}(1-R) \tag{8.21}$$

Toutefois, quitte à s'écarter légèrement des rendements donnés par l'algorithme 8.2.1, il faut faire en sorte de choisir des codes constituants tels que la somme de leurs dimensions soit $\lfloor Rn \rfloor$.

8.3.1 Le cas particulier des codes polaires

Les codes polaires construits par Arıkan dans [Arı09] sont un cas particulier des codes $U|U+V$ récursifs. En effet, un code polaire de longueur 2^d est un code $U|U+V$ récursif de profondeur d associé à un arbre binaire parfait de profondeur d. Autrement dit, c'est un code $U|U+V$ récursif dont tous les codes constituants $\{U_{\mathbf{x}}$ tel que $\mathbf{x} \in \{0,1\}^d\}$ sont des codes triviaux de longueur 1. Pour chacun des codes constituants, nous choisissons sa dimension qui sera 0 ou 1 de la même façon que pour les codes $U|U+V$ récursifs. Les codes constituants de dimension 1 sont nécessairement le code trivial composé des deux

seuls mots 1 et 0. D'autre part, si le code polaire est linéaire, alors les codes constituants de dimension 0 ne peuvent être que le code trivial composé de l'unique mot 0. On dit que la position est *gelée*. Pour une dimension k fixée, il faudra donc geler exactement $n-k$ positions : les positions associées aux $n-k$ canaux de plus faibles capacités.

Remarque 8.3.1. En gelant $n-k$ positions avec des valeurs qui ne sont pas nécessairement 0, nous pouvons produire des cosets de codes polaires de dimension k.

Notons que l'algorithme 8.3.1 paramétré avec $\Gamma = 0$ produit un code qui est très proche des codes polaires. En effet, comme pour les codes polaires, nous obtenons exclusivement des codes constituants de dimension ou codimension nulle (respectivement les positions gelées et les positions à déterminer parmi deux possibilités). Cependant, ces codes ne sont pas exactement les codes polaires car les longueurs des codes constituants ne sont pas nécessairement toutes égales à 1.

8.4 Décodage des codes constituants

Pour décoder les codes $U|U+V$ récursifs, nous aurons besoin d'un algorithme de décodage souple par maximum de vraisemblance des codes constituants. Un tel algorithme retourne le mot de code le plus probable connaissant la probabilité de chaque bit de valoir 1. La définition 8.4.1 spécifie plus formellement cette notion.

Définition 8.4.1. *Soit $\mathcal{C}$ un code de longueur n. Le décodage souple par maximum de vraisemblance d'un vecteur de probabilités $\mathbf{p} := (p_1, \cdots, p_n) \in [0,1]^n$ consiste à déterminer le mot $\mathbf{c} := (c_1, \cdots, c_n) \in \mathcal{C}$ maximisant la probabilité suivante :*

$$\prod_{i=1}^{n} {p_i}^{c_i} (1-p_i)^{(1-c_i)} \tag{8.22}$$

Soit $\mathcal{C}$ un code de longueur n et de dimension ou codimension au plus Γ. Nous souhaitons construire un algorithme de décodage souple par maximum de vraisemblance de $\mathcal{C}$ qui s'exécute en un temps de l'ordre de $O\left(2^{\Gamma}\right)$. Lorsque c'est la dimension du code qui est inférieure à Γ, une telle construction est triviale. En effet, il suffit d'énumérer tous les mots du code et de déterminer celui qui maximise la probabilité donnée par la définition 8.4.1. En revanche, la construction d'un algorithme de décodage d'un code aléatoire qui s'exécute en un temps de l'ordre de $O\left(2^{\Gamma}\right)$ n'est pas aussi simple dans le cas où c'est la codimension du code qui est inférieure à Γ.

Supposons à présent que $\mathcal{C}$ soit de dimension k telle que $n-k \leqslant \Gamma$. Soit un vecteur de probabilités $\mathbf{p} := (p_1, \cdots, p_n) \in [0,1]^n$. Notre objectif est de construire un algorithme qui énumère les mots $\mathbf{c} := (c_1, \cdots, c_n)$ de $\mathbb{F}_2^n$ dans l'ordre décroissant des

$$\prod_{i=1}^{n} {p_i}^{c_i} (1-p_i)^{(1-c_i)}$$

et choisit le premier d'entre eux qui appartient au code $\mathcal{C}$. Cette procédure est alors un décodage souple par maximum de vraisemblance répondant bien à la définition 8.4.1.

Soit le mot $\mathbf{y} := (y_1, \cdots, y_n) \in \mathbb{F}_2^n$ tel que :

$$y_i = \begin{cases} 1 & \text{si } p_i > 1/2 \\ 0 & \text{si } p_i < 1/2 \\ b & \text{si } p_i = 1/2 \end{cases}$$

avec b tiré uniformément dans $\{0,1\}$.

On pose alors le vecteur de probabilités $\mathbf{p}' := (p'_1, \cdots, p'_n)$ du motif d'erreur avec :

$$p'_i = \begin{cases} p_i & \text{si } y_i = 0 \\ 1 - p_i & \text{si } y_i = 1 \end{cases}$$

Lemme 8.4.2. *Soient* $\mathbf{e} := (e_1, \cdots, e_n) \in \mathbb{F}_2^n$ *et* $\mathbf{c} := \mathbf{y} + \mathbf{e} = (c_1, \cdots, c_n)$.

$$\forall i \in [\![1, n]\!] \ , \quad {p_i}^{c_i}(1 - p_i)^{(1-c_i)} = {p'_i}^{e_i}(1 - p'_i)^{(1-e_i)}$$

Démonstration du lemme 8.4.2.

$$\begin{aligned} {p_i}^{c_i}(1 - p_i)^{(1-c_i)} &= \left({p'_i}^{(1-y_i)}(1 - p'_i)^{y_i}\right)^{c_i} \left({p'_i}^{y_i}(1 - p'_i)^{(1-y_i)}\right)^{(1-c_i)} \\ &= {p'_i}^{((1-y_i)c_i + y_i(1-c_i))} \times (1 - p'_i)^{(y_i c_i + (1-y_i)(1-c_i))} \\ &= {p'_i}^{e_i}(1 - p'_i)^{(1-e_i)} \end{aligned}$$

□

D'après le lemme 8.4.2, énumérer les mots $\mathbf{c} := (c_1, \cdots, c_n)$ de $\mathbb{F}_2^n$ dans l'ordre décroissant des :

$$\prod_{i=1}^{n} {p_i}^{c_i}(1 - p_i)^{(1-c_i)}$$

revient à énumérer les vecteurs d'erreur $\mathbf{e} := (e_1, \cdots, e_n)$ de $\mathbb{F}_2^n$ dans l'ordre décroissant des :

$$\prod_{i=1}^{n} {p'_i}^{e_i}(1 - p'_i)^{(1-e_i)}$$

Lemme 8.4.3. *L'ordre décroissant des* $\displaystyle\prod_{i=1}^{n} {p'_i}^{e_i}(1 - p'_i)^{(1-e_i)}$ *est exactement l'ordre décroissant des* $\displaystyle\sum_{i \in S} \log_2\left(\frac{p'_i}{1 - p'_i}\right)$ *avec* S *le support de* $\mathbf{e}$.

Démonstration du lemme 8.4.3.

$$\begin{aligned} \log_2\left(\prod_{i=1}^{n} {p'_i}^{e_i}(1 - p'_i)^{(1-e_i)}\right) &= \sum_{i=1}^{n} \log_2\left({p'_i}^{e_i}(1 - p'_i)^{(1-e_i)}\right) \\ &= \sum_{i=1}^{n} \log_2\left(1 - p'_i\right) + \sum_{i \in S} \log_2\left(\frac{p'_i}{1 - p'_i}\right) \end{aligned}$$

Or la fonction $\log_2$ est une fonction croissante et donc elle ne change pas l'ordre. De plus $\displaystyle\sum_{i=1}^{n} \log_2\left(1 - p'_i\right)$ est constant quel que soit $\mathbf{e} \in \mathbb{F}_2^n$. D'où finalement le lemme. □

Nous proposons finalement l'algorithme 8.4.1 pour décoder un code aléatoire de codimension faible avec de l'information souple.

Proposition 8.4.4. *L'algorithme 8.4.1 énumère les vecteurs d'erreur* $\mathbf{e} := (e_1, \cdots, e_n) \in \mathbb{F}_2^n$ *de support* $\text{supp}(\mathbf{e})$ *dans l'ordre décroissant des :*

$$\sum_{i \in \text{supp}(\mathbf{e})} \log_2\left(\frac{p'_i}{1 - p'_i}\right) \tag{8.23}$$

Démonstration de la proposition 8.4.4.

Soit S tel que $\{T_i\}_{i\in S}$ soit le support $\mathrm{supp}\,(\mathbf{e})$ du vecteur d'erreur courant $\mathbf{e}$. Soit :

$$\pi = \sum_{i\in \mathrm{supp}(\mathbf{e})} \log_2\left(\frac{p'_i}{1-p'_i}\right)$$

Les couples (S',π') et (S'',π'') construits comme dans l'algorithme 8.4.1 sont tels que :

(a) $\pi' = \sum_{i\in \mathrm{supp}(\mathbf{e}')} \log_2\left(\frac{p'_i}{1-p'_i}\right)$ avec $\mathrm{supp}\,(\mathbf{e}') = \{T_i\}_{i\in S'}$;

(b) $\pi'' = \sum_{i\in \mathrm{supp}(\mathbf{e}'')} \log_2\left(\frac{p'_i}{1-p'_i}\right)$ avec $\mathrm{supp}\,(\mathbf{e}'') = \{T_i\}_{i\in S''}$;

(c) $\pi \geqslant \pi'$ et $\pi \geqslant \pi''$.

Ainsi, dans l'algorithme 8.4.1, chaque élément qui est tiré de la file de priorité est associé à un vecteur d'erreur prioritaire sur tous les autres vecteurs d'erreur qui n'ont pas été encore énumérés. □

Les lemmes 8.4.2 et 8.4.3, montrent que l'ordre de la proposition 8.4.4 est celui rangeant les vecteurs d'erreur du plus probable au moins probable.

En outre, il est facile de montrer que le nombre de mots à énumérer est de l'ordre de $O\big(2^{n-k}\big)$ avec $n-k$ la codimension du code. En effet, $\mathcal{C}$ possède 2^{n-k} syndromes (dont le syndrome nul) et donc chaque mot $\mathbf{c}$ énuméré a une probabilité de $\frac{1}{2^{n-k}}$ d'être un mot de code. Il faut donc énumérer $2^{n-k} \leqslant 2^{\Gamma}$ mots pour que l'espérance qu'au moins l'un d'entre eux soit un mot de code soit supérieure à 1.

Remarque 8.4.1. Dans l'algorithme 8.4.1, la taille de la file de priorité augmente linéairement avec le nombre d'itérations. En effet, à chaque itération, la taille de la file de priorité augmente d'au plus 1. Or le nombre d'itérations est de l'ordre de $O\big(2^{n-k}\big) \leqslant O\big(2^{\Gamma}\big)$. Ainsi, la taille de la file de priorité est d'au plus de l'ordre de $O\big(2^{\Gamma}\big)$.

Finalement, pour les codes aléatoires de dimension ou codimension faible, nous avons le théorème suivant :

Théorème 8.4.5. *Soit $\mathcal{C}$ un code linéaire $[n,k]_2$ tel que $\min(k,n-k) \leqslant \Gamma$. Il existe un algorithme de décodage souple par maximum de vraisemblance du code $\mathcal{C}$ qui s'exécute en un temps de l'ordre de $O\big(2^{\Gamma}\big)$.*

Par la suite, nous supposerons que Γ est constant par rapport à n. Ainsi, les décodages des codes constituants des codes $U|U+V$ récursifs produits par l'algorithme 8.3.1 peuvent être réalisés en un temps constant.

Algorithme 8.4.1 : Décodage souple d'un code aléatoire de codimension faible

Entrées : un code $\mathcal{C}$ de longueur n et de dimension k ;
un vecteur de probabilités $\mathbf{p} := (p_1, \cdots, p_n) \in [0,1]^n$.

Sortie : un mot $\mathbf{c}$ du code $\mathcal{C}$ maximisant la probabilité $\prod_{i=1}^{n} {p_i}^{c_i}(1-p_i)^{(1-c_i)}$.

Fonction *DecodeConstituant*$(\mathcal{C}, \mathbf{p})$
  pour tout $i \in [\![1, n]\!]$ **faire**
    $y_i \longleftarrow \begin{cases} 1 & \text{si } p_i > 1/2 \\ 0 & \text{si } p_i < 1/2 \\ b & \text{si } p_i = 1/2 \end{cases}$ avec b tiré uniformément dans $\{0,1\}$;
    $p'_i \longleftarrow \begin{cases} p_i & \text{si } y_i = 0 \\ 1-p_i & \text{si } y_i = 1 \end{cases}$;
  finPour
  $\mathbf{y} \longleftarrow (y_1, \cdots, y_n)$; `/* y est le mot reçu */`
  $\mathbf{p'} \longleftarrow (p'_1, \cdots, p'_n)$; `/* p' est le vecteur de probabilités du motif d'erreur */`
  $T \longleftarrow$ ranger $[\![1, n]\!]$ tel que $\left\{p'_{T_1}, \cdots, p'_{T_n}\right\}$ soient dans l'ordre décroissant ;
  Tas $\longleftarrow$ initialiser une file de priorité ;
  $\mathbf{e} \longleftarrow \mathbf{0}$; `/* e est le potentiel vecteur d'erreur */`
  $S \longleftarrow \varnothing$; `/* S est tel que` $\{T_i\}_{i\in S}$ `est le support de e */`
  $\pi \longleftarrow 1$; `/* π est la priorité du vecteur d'erreur */`
  tant que $\mathbf{y} + \mathbf{e}$ *n'est pas dans* $U_{\mathbf{x}}$ **faire**
    $t \longleftarrow |S|$;
    si $t = 0$ **alors**
      $S'' \longleftarrow \{1\}$;
      $i \longleftarrow T_1$;
      $\pi'' \longleftarrow \pi + (2e_i - 1)\log_2\left(\frac{1-p'_i}{p'_i}\right)$;
    sinon si $t < n$ **alors**
      $S' \longleftarrow S$;
      $S'_t \longleftarrow S'_t + 1$;
      $i \longleftarrow T_{S_t}$; $j \longleftarrow T_{S'_t}$;
      $\pi' \longleftarrow \pi - (2e_i - 1)\log_2\left(\frac{1-p'_i}{p'_i}\right) + (2e_j - 1)\log_2\left(\frac{1-p'_j}{p'_j}\right)$;
      ajouter S' à Tas avec priorité π' ;
      $S'' \longleftarrow S$;
      $S''_{t+1} \longleftarrow S'_t + 1$;
      $i \longleftarrow T_{S''_{t+1}}$;
      $\pi'' \longleftarrow \pi + (2e_i - 1)\log_2\left(\frac{1-p'_i}{p'_i}\right)$;
      ajouter S'' à Tas avec priorité π'' ;
    finSi
    $(S, \pi) \longleftarrow$ tirer l'élément prioritaire de Tas avec sa priorité ;
    construire $\mathbf{e}$ tel que le support de $\mathbf{e}$ soit $\{T_i\}_{i\in S}$;
  fin
  retourner $\mathbf{y} + \mathbf{e}$;
finFonction

8.5 Décodage par annulation successive

Dans cette sous-section, nous décrivons un algorithme de décodage pour les codes polaires ou les codes $U|U+V$ récursifs générés par l'algorithme 8.3.1. Ce décodage est inspiré du décodage par *annulation successive* des codes polaires. Pour généraliser la notion aux codes $U|U+V$ récursifs, un décodage par annulation successive décode les codes constituants successivement, sans revenir sur un décodage déjà effectué. Ces décodages sont effectués dans un ordre bien précis : du code constituant le plus à droite au code constituant le plus à gauche dans notre construction sous forme d'arbre. De plus, chaque

décodage d'un code constituant prend en compte les décodages des codes constituants précédents.

L'algorithme que nous présentons est une version récursive du décodage par annulation successive habituellement utilisé pour les codes polaires. Il est toutefois facile d'en donner une version itérative.

Décodage des codes $U|U+V$. Soit U et V deux codes de longueurs $\frac{n}{2}$ et de dimensions respectives k_U et k_V. Commençons par décrire le décodage du code $U|U+V$. Soit $(\mathbf{u}, \mathbf{v}) \in U \times V$. Un émetteur envoie le mot $(\mathbf{u}, \mathbf{u}+\mathbf{v})$ via un canal binaire symétrique de probabilité d'erreur p. Le mot reçu correspondant est noté $\mathbf{y} := (y_1, \cdots, y_n)$. Étant donné la nature du canal de transmission, chaque bit de $\mathbf{y}$ a une probabilité p d'être différent du bit émis. On définit alors un vecteur de probabilités $\mathbf{p} := (p_1, \cdots, p_n) \in [0,1]^n$ tel que :

$$\forall i \in [\![1, n]\!], \begin{cases} p_i = p & \text{si } y_i = 0 \\ p_i = (1-p) & \text{si } y_i = 1 \end{cases}$$

Chaque composante p_i du vecteur $\mathbf{p}$ représente la probabilité que le $i^{\text{ième}}$ bit de $(\mathbf{u}, \mathbf{u}+\mathbf{v})$ vaille 1 connaissant la valeur du bit y_i.

Nous définissons deux opérations sur les vecteurs de probabilités :

Définition 8.5.1. *Soient $\mathbf{p} := (p_1, \cdots, p_n)$ et $\mathbf{q} := (q_1, \cdots, q_n)$ deux vecteurs de probabilités. Soit $\mathbf{x} := (x_1, \cdots, x_n)$ un mot binaire de longueur n.*
Les deux opérations $\boxplus$ et $\boxtimes^{\mathbf{x}}$ sont définies comme suit :

$$\forall i \in [\![1, n]\!] \ , \ (\mathbf{p} \boxplus \mathbf{q})_i := p_i + q_i - 2(p_i q_i)$$

et :

$$\forall i \in [\![1, n]\!] \ , \ (\mathbf{p} \boxtimes^{\mathbf{x}} \mathbf{q})_i := \begin{cases} \dfrac{p_i(1-q_i)}{p_i(1-q_i) + q_i(1-p_i)} & si\ x_i = 1 \\ \dfrac{p_i q_i}{p_i q_i + (1-p_i)(1-q_i)} & si\ x_i = 0 \end{cases}$$

Dans le cadre de notre décodage du code $U|U+V$, nous notons $\mathbf{p'} := \mathbf{p}_{[\![1, \frac{n}{2}]\!]}$ les $\frac{n}{2}$ premières composantes de $\mathbf{p}$ et $\mathbf{p''} := \mathbf{p}_{[\![\frac{n}{2}+1, n]\!]}$ les $\frac{n}{2}$ suivantes. Le vecteur de probabilités $\mathbf{p'} \boxplus \mathbf{p''}$ représente un vecteur de probabilités du mot $\mathbf{v}$ et $\mathbf{p'} \boxtimes^{\mathbf{v}} \mathbf{p''}$ représente un vecteur de probabilités du mot $\mathbf{u}$. L'algorithme de décodage du code $U|U+V$ consiste alors dans un premier temps à retrouver $\mathbf{v}$ à l'aide du vecteur de probabilités $\mathbf{p'} \boxplus \mathbf{p''}$ puis, dans un second temps, à retrouver $\mathbf{u}$ à l'aide du vecteur de probabilités $\mathbf{p'} \boxtimes^{\mathbf{v}} \mathbf{p''}$.

Décodage simple par annulation successive des codes $U|U+V$ récursifs. L'algorithme de décodage d'un code $U|U+V$ récursif consiste à appliquer récursivement le décodage $U|U+V$. Le pseudo-code 8.5.1 décrit ce décodage.

Algorithme 8.5.1 : Décodage simple par annulation successive (version récursive)

Paramètres : un code $U|U+V$ récursif U_ε de longueur n associé à un arbre binaire $\mathscr{T}$ et un ensemble de codes constituants $\{U_{\mathbf{x}}\}$.
Entrées : un mot reçu $\mathbf{y} \in \mathbb{F}_2^n$;
la probabilité d'erreur p du canal binaire symétrique.
Sortie : un mot $\mathbf{c} \in U_\varepsilon$.

1 **pour tout** $i \in [\![1, n]\!]$ **faire**
2 $\quad p_i \leftarrow \begin{cases} p & \text{si } y_i = 0 \\ 1-p & \text{si } y_i = 1 \end{cases}$;
3 **finPour**
4 $\mathbf{p} \leftarrow (p_1, \cdots, p_n)$;
5 **retourner** DECODE($\mathbf{p}$, n, ε, $\mathscr{T}$) ; `/* ε dénote la chaîne binaire vide */`

---

1 **Fonction** *DECODE*($\mathbf{p}$, n, $\mathbf{x}$, $\mathscr{T}$)
2 $\quad$ **si** $\mathbf{x}$ *est une feuille de* $\mathscr{T}$ **alors**
3 $\quad\quad$ **retourner** DECODECONSTITUANT$(U_{\mathbf{x}}, \mathbf{p})$; `/* cf. algorithme 8.4.1 */`
4 $\quad$ **sinon**
5 $\quad\quad$ $\mathbf{p}' \leftarrow \mathbf{p}_{[\![1, \frac{n}{2}]\!]}$;
6 $\quad\quad$ $\mathbf{p}'' \leftarrow \mathbf{p}_{[\![\frac{n}{2}+1, n]\!]}$;
7 $\quad\quad$ $\mathbf{v} \leftarrow$ DECODE($\mathbf{p}' \boxplus \mathbf{p}''$, $\frac{n}{2}$, $(\mathbf{x}, 1)$, $\mathscr{T}$) ; `/* cf. définition 8.5.1 */`
8 $\quad\quad$ $\mathbf{u} \leftarrow$ DECODE($\mathbf{p}' \boxtimes^{\mathbf{v}} \mathbf{p}''$, $\frac{n}{2}$, $(\mathbf{x}, 0)$, $\mathscr{T}$) ; `/* cf. définition 8.5.1 */`
9 $\quad\quad$ **retourner** $(\mathbf{u}, \mathbf{u}+\mathbf{v})$;
10 $\quad$ **finSi**
11 **finFonction**

La version itérative du décodage par annulation successive. Pour de meilleures performances en pratique, il est plus judicieux d'utiliser une version itérative de l'algorithme 8.5.1. Il est assez classique de transformer une procédure récursive en une boucle itérative en algorithmique. Dans notre cas, pour décoder dans U_ε il nous faut sauvegarder dans un arbre binaire $\mathscr{T}'$ (de structure analogue à $\mathscr{T}$) tous les vecteurs de probabilités $\mathbf{p}'$ et $\mathbf{p}''$ ainsi que tous les mots de codes intermédiaires $\mathbf{u}$ et $\mathbf{v}$ calculés aux étapes 5 à 8 de l'algorithme 8.5.1. À chaque décodage d'un code constituant, l'arbre est mis à jour en effectuant les mêmes opérations que décrites dans l'algorithme. En fait, il n'est même pas nécessaire de sauvegarder tout l'arbre : en effet, si $U_{\mathbf{x}}$ est le prochain code constituant à être décodé alors seuls les nœuds affiliés à la feuille $\mathbf{x}$ ainsi que leurs frères de droite sont utiles. Finalement, le pseudo-code 8.5.2 donne la version itérative de l'algorithme 8.5.1 de décodage par annulation successive.

Algorithme 8.5.2 : Décodage simple par annulation successive (version itérative)

Paramètres : un code $U|U+V$ récursif U_ε de longueur n associé à un arbre binaire $\mathscr{T}$ de profondeur d et un ensemble de codes constituants $\{U_{\mathbf{x}}\}$.
Entrées : un mot reçu $\mathbf{y} \in \mathbb{F}_2^n$; la probabilité d'erreur p du canal binaire symétrique.
Sortie : un mot $\mathbf{c} \in U_\varepsilon$.

initialiser un tableau Tab_p de taille d ; /* Tab_p[i] contiendra un vecteur de probabilités associé à un code $U_{\mathbf{x}}$ où x est un nœud de l'étage i de $\mathscr{T}$ (donc x est de longueur i). */
initialiser un tableau bidimensionnel Tab_v de taille d ; /* Tab_v[0][i] contiendra un mot d'un code $U_{\mathbf{x}}$ où x est un nœud de l'étage i de $\mathscr{T}$ et x est un fils gauche (c'est-à-dire que le bit le plus à droite de x est 0). Analogue pour Tab_v[1][i]. */
pour tout $i \in [\![1, n]\!]$ **faire**
  $p_i \leftarrow \begin{cases} p & \text{si } y_i = 0 \\ 1-p & \text{si } y_i = 1 \end{cases}$;
finPour
$\mathbf{p} \leftarrow (p_1, \cdots, p_n)$; /* initialisation du vecteur probabilité */
$\mathbf{x} \leftarrow \varepsilon$; /* ε est le mot vide */
$i \leftarrow 0$; /* la longueur de x et donc l'étage dans lequel se situe le nœud dans $\mathscr{T}$. On pourra alors utiliser la notation $\mathbf{x} := (x_i, \cdots, x_2, x_1)$ où x_1 est toujours le bit le plus à droite. */
répéter indéfiniment
  tant que $U_{\mathbf{x}}$ *n'est pas un code constituant* **faire**
    Tab_p[i] $\leftarrow \mathbf{p}$;
    $n' \leftarrow \frac{n}{2^i}$;
    $\mathbf{p} \leftarrow \mathbf{p}_{[\![1, \frac{n'}{2}]\!]} \boxplus \mathbf{p}_{[\![\frac{n'}{2}+1, n']\!]}$; /* cf. définition 8.5.1 */
    $\mathbf{x} \leftarrow (\mathbf{x}, 1)$;
    $i \leftarrow i + 1$;
  finTantQue
  $b \leftarrow x_1$ si $i > 0$ et 1 sinon ;
  Tab_v[b][i] $\leftarrow$ DecodeConstituant $(U_{\mathbf{x}}, \mathbf{p})$;
  tant que $b = 0$ **faire**
    supprimer le bit le plus à droite de $\mathbf{x}$;
    $i \leftarrow i - 1$;
    $b \leftarrow x_1$ si $i > 0$ et 1 sinon ;
    Tab_v[b][i] $\leftarrow$ (Tab_v[0][$i+1$], Tab_v[0][$i+1$] + Tab_v[1][$i+1$]) ;
  finTantQue
  si $i = 0$ **alors**
    retourner Tab_v[1][0] ;
  finSi
  $\mathbf{p}' \leftarrow$ Tab_p[$i-1$] ;
  $\mathbf{v} \leftarrow$ Tab_v[i] ;
  $n' \leftarrow \frac{n}{2^i}$;
  $\mathbf{p} \leftarrow \mathbf{p}'_{[\![1, \frac{n'}{2}]\!]} \boxtimes^{\mathbf{v}} \mathbf{p}'_{[\![\frac{n'}{2}+1, n']\!]}$; /* cf. définition 8.5.1 */
  $\mathbf{x} \leftarrow (x_i, \cdots, x_2, 0)$;
finRépéter

8.6 Distorsion des décodages des codes $U|U+V$ récursifs

La *distorsion* d'un algorithme de décodage d'un code est la distance moyenne de correction d'un mot tiré uniformément dans l'espace ambiant.

Définition 8.6.1. *Soit $\mathcal{C}$ un code sur $\mathbb{F}_2^n$ et soit D une fonction de décodage de $\mathcal{C}$. La distorsion de $\mathcal{C}$ est :*

$$\mathbb{E}\left(\Delta\left(\mathbf{x}, D(\mathbf{x})\right)\right) := \sum_{d=0}^{n} d \cdot \mathbb{P}\left(\Delta\left(\mathbf{x}, D(\mathbf{x})\right) = d\right) \tag{8.24}$$

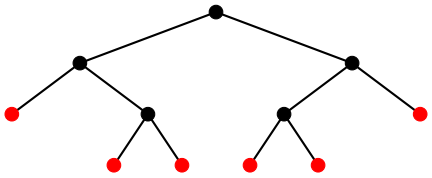
où $\mathbf{x}$ est tiré uniformément dans $\mathbb{F}_2^n$.

Un décodage est d'autant plus performant que sa distorsion est proche de la distance de Gilbert-Varshamov (cf. sous-section 2.7).

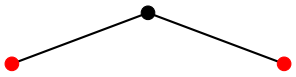
Commençons par étudier un cas simple : un code $U|U+V$ récursif de longueur 32 et de dimension 16 construit avec l'algorithme 8.3.1. Ce code, que nous notons $\mathcal{C}_\Gamma^{[32,16]}$, dépend de la dimension ou codimension maximale Γ des codes constituants. Observons alors la distorsion de $\mathcal{C}_\Gamma^{[32,16]}$ en fonction de Γ que nous avons tracé sur la figure 8.4. Sur cette figure, nous avons aussi représenté la distance de Gilbert-Varshamov (la valeur exacte et non la valeur asymptotique). Cette borne est aussi la distorsion optimale que l'on peut espérer atteindre.

Figure 8.4 – Distorsion du décodage par annulation successive de codes $U|U+V$ récursifs $[32, 16]$ en fonction de la dimension ou codimension maximale Γ des codes constituants.

Sur cette figure, nous observons des "plateaux". Ceux-ci sont dûs au fait que pour des paramètres Γ distincts, l'algorithme 8.3.1 peut produire des codes $U|U+V$ récursifs associés à une même structure d'arbre. Par exemple, l'arbre associé aux codes $U|U+V$ récursifs $[32, 16]$ produits avec $\Gamma \in \{1, 2, 3\}$ est :



ou encore, avec $\Gamma \in \{5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}$, cet arbre est :



Nous remarquons aussi sur la figure 8.4 que pour $\Gamma = k$, notre décodage atteint une distorsion égale à la distance de Gilbert-Varshamov qui est la distorsion optimale. Ce résultat s'explique par le fait que pour $\Gamma = k$, le code $U|U+V$ récursif obtenu avec l'algorithme 8.3.1 est tout simplement un code aléatoire.

Un phénomène étrange se produit pour $\Gamma = 0$. En effet, la distorsion est particulièrement bonne pour ce paramètre. Ainsi, pour construire des codes $U|U+V$ récursifs dont le décodage par annulation successive fournit la meilleure distorsion possible, il nous faut paramétrer l'algorithme 8.3.1 avec $\Gamma = 0$ ou bien avec Γ suffisamment grand. En effet, nous avons déjà remarqué que lorsque Γ tend vers la dimension k du code, la distorsion de notre décodage tend vers la distorsion optimale ; il existe donc un seuil Γ_0 tel que pour tout $\Gamma \geqslant \Gamma_0$, le code obtenu soit plus performant que le code construit avec le paramètre $\Gamma = 0$ (ou bien que les codes polaires qui en sont très proches). Cependant, en pratique, Γ_0 sera souvent trop grand ; par exemple, la figure 8.5 montre que pour des codes $U|U+V$ récursifs $[1024, 512]$, il faut choisir $\Gamma \geqslant 20$ pour atteindre au moins les performances du code polaire $[1024, 512]$.

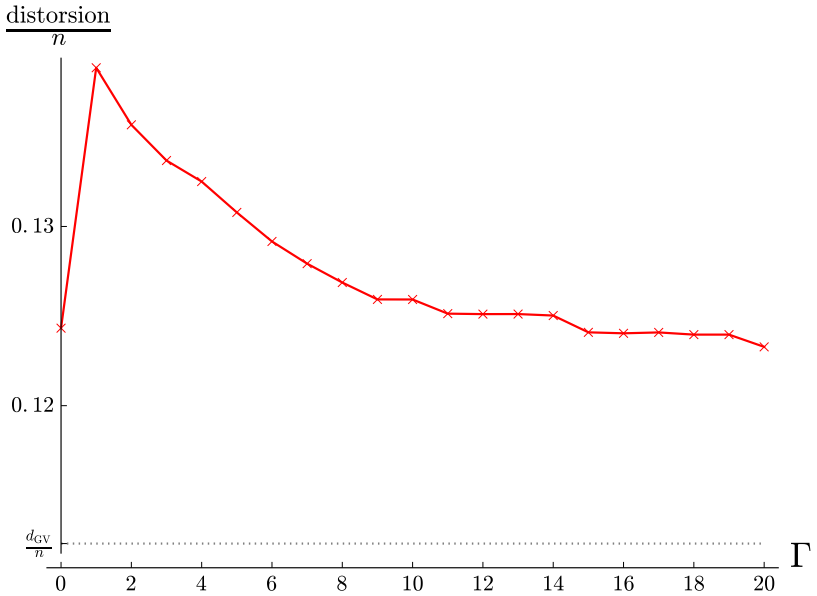


Figure 8.5 – Distorsion du décodage par annulation successive de codes $U|U+V$ récursifs $[1024, 512]$ en fonction de la dimension ou codimension maximale Γ des codes constituants.

Sur les figures 8.4 et 8.5, nous constatons que la distorsion du décodage par annulation successive est particulièrement bonne lorsque $\Gamma = 0$. Lorsque la dimension ou codimension

d'un code constituant est nulle, celui-ci est soit le code contenant uniquement le mot nul, soit le code complet. Ces codes sont optimaux dans le sens où la correction ne peut pas échouer. En revanche, lorsque la dimension ou la codimension du code constituant est faible mais non nulle, alors définir ce code aléatoirement n'est pas le meilleur choix possible. Nous pouvons alors nous demander parmi quel famille de codes nous devons choisir nos codes constituants dans l'algorithme 8.3.1 pour construire des codes $U|U+V$ récursifs les plus performants possible. Nous proposons d'utiliser des codes de Reed-Muller.

Les codes de Reed-Muller. Les codes de Reed-Muller sont très étudiés en théorie des codes correcteurs. Ils ont été découverts par Muller en 1954 et le premier algorithme de décodage a été proposé par Reed la même année.

Nous définissons les codes de Reed-Muller par leurs matrices génératrices construites de façon récursive. La définition 8.6.2 précise cette construction.

Définition 8.6.2 (code de Reed-Muller)**.** *Soient deux entiers r et m tels que $0 \leqslant r \leqslant m$. Le code de Reed-Muller* $\mathrm{RM}(r,m)$ *d'ordre r est un code linéaire de matrice génératrice* $\mathbf{G}(r,m)$ *vérifiant :*

(a) $\mathbf{G}(m,m) := \mathbf{Id}_{2^m}$ *;*

(b) $\mathbf{G}(-1,m)$ *est la matrice vide de dimension* 0×2^m *;*

(c) $\mathbf{G}(r,m) := \left[\begin{array}{c|c} \mathbf{G}(r,m-1) & \mathbf{G}(r,m-1) \\ \hline \mathbf{0} & \mathbf{G}(r-1,m-1) \end{array}\right]$ *si* $0 \leqslant r < m$.

Les codes de Reed-Muller $\mathrm{RM}(r,m)$ sont de longueur 2^m, de dimension $\sum_{i=0}^{r} \binom{m}{i}$ et de distance minimale 2^{m-r}. Parmi les codes de Reed-Muller, on retrouve certains codes triviaux : $\mathrm{RM}(-1,m)$ est le code contenant uniquement le mot nul, $\mathrm{RM}(m,m)$ est le code complet, $\mathrm{RM}(0,m)$ est le code de répétition et $\mathrm{RM}(m-1,m)$ est le code de parité.

Notons que les codes de Reed-Muller de longueur donnée ne peuvent atteindre toutes les dimensions possibles. Par exemple, pour tout $m > 1$, il n'existe pas de code de Reed-Muller de dimension 2. Or dans notre algorithme 8.3.1, nous devons pouvoir choisir des codes constituants de n'importe quelles dimensions. Pour cela, nous utilisons des codes de Reed-Muller raccourcis.

Finalement, nous remarquons expérimentalement que l'utilisation de code de Reed-Muller raccourcis pour construire nos codes $U|U+V$ récursifs permet d'obtenir des codes très légèrement plus performants que les codes polaires.

8.7 Décodage en liste

Le décodage par annulation successive des codes polaires est loin de donner le mot de code le plus probablement émis. Comme nous l'avons expliqué précédemment, ceci est dû aux mauvaises décisions qui peuvent être prises lors des décodages des codes constituants. Dans le cas des codes polaires, les codes constituants sont des codes de longueur 1. Lors du décodage par annulation successive, nous décodons les codes constituants les uns après les autres sans jamais revenir sur un décodage déjà effectué. Si le code constituant courant est de dimension 0 alors il contient uniquement le mot de code 0 (on dit que la position est gelée) et nous ne pouvons pas prendre de mauvaise décision en le décodant. En revanche, si le code constituant courant est de dimension 1 alors il contient les mots 0 et 1 et bien que nous choisissons la valeur la plus probable, il est possible que nous nous trompions... Cette erreur nous éloigne alors du mot de code le plus probablement émis.

Dans [TV15], Tal et Vardy ont proposé une amélioration du décodage (itératif) par annulation successive qui permet de prendre les décisions "dures" le plus tardivement possible. Leur décodage est en fait un décodage en liste : il retourne une liste de mots de

code dans laquelle nous recherchons ensuite exhaustivement le mot le plus probablement émis.

Le décodage de Tal et Vardy est donc une adaptation de l'algorithme 8.5.2 pour en faire un décodage en liste retournant au plus ℓ mots de codes. Dans ce décodage, à chaque fois qu'un code constituant de dimension 1 est exploré, les deux tableaux Tab_p et Tab_v sont dupliqués. L'un des duplicas correspond à choisir la valeur 1 pour le code constituant considéré tandis que l'autre duplica correspond au choix 0. Si l'on s'arrête là, alors notre algorithme ne serait pas différent d'une exploration exhaustive de tous les mots de code... Cependant, à chaque opération de duplication, seules les structures correspondant aux ℓ mots de code (partiels) les plus probables sont conservés. Lorsque le dernier code constituant est traité, nous avons une liste contenant au plus ℓ structures correspondant chacune à un mot de code. Le mot de code retourné est alors celui qui a été le plus probablement émis parmi les mots de cette liste.

Nous pouvons remarquer que les tableaux que nous dupliquons ont de nombreux éléments en commun. L'algorithme de Tal et Vardy utilise un système de pointeurs pour ne dupliquer physiquement que les parties nécessaires des structures. Avec cette astuce, le décodage de Tal et Vardy a une complexité en temps de l'ordre de $O(\ell n \log(n))$.

La figure 8.6 illustre la distorsion du décodage en liste de Tal et Vardy de codes polaires de longueur 1024 en fonction du rendement $R := \frac{k}{n}$ et pour différentes tailles ℓ de listes de décodage. Le cas $\ell = 1$ correspond au décodage simple par annulation successive. Sur cette figure, nous comparons la distorsion obtenue avec la distance de Gilbert-Varshamov que nous avons représenté avec des tirets gris. Enfin, la figure 8.7 montre la distorsion d'un code polaire $[1024, 512]$ en fonction de la taille de la liste de décodage.

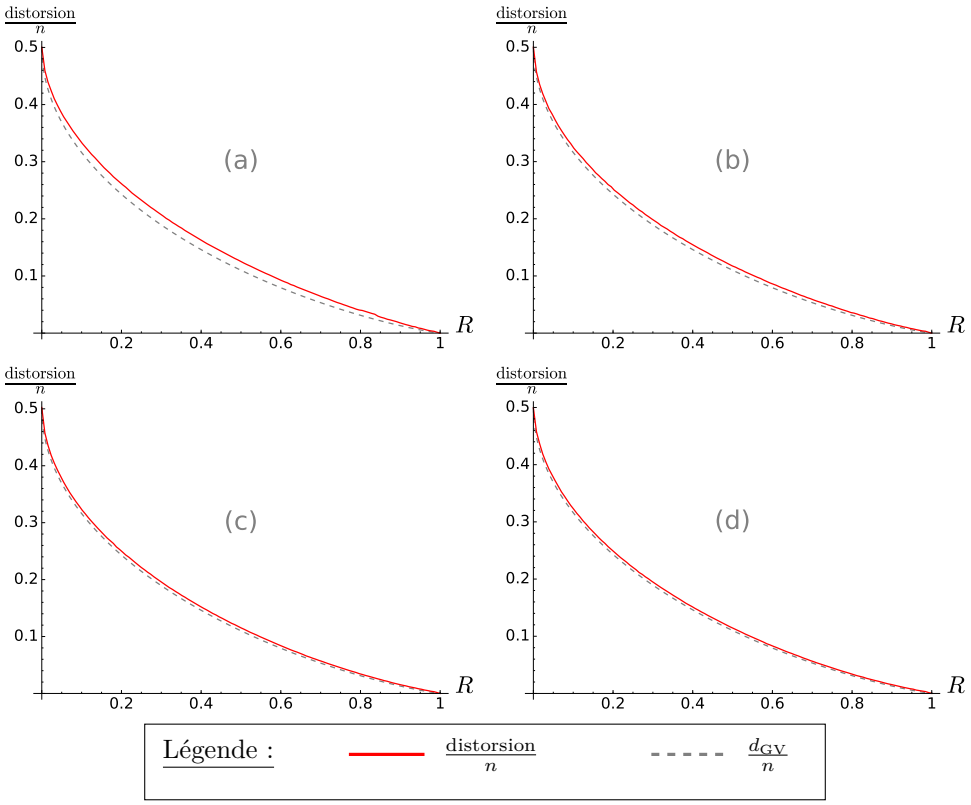


Figure 8.6 – Distorsion du décodage de Tal et Vardy pour un code polaire de longueur $n = 1024$ en fonction du rendement R. (a) $\ell = 1$, (b) $\ell = 10$, (c) $\ell = 100$, (d) $\ell = 1000$.

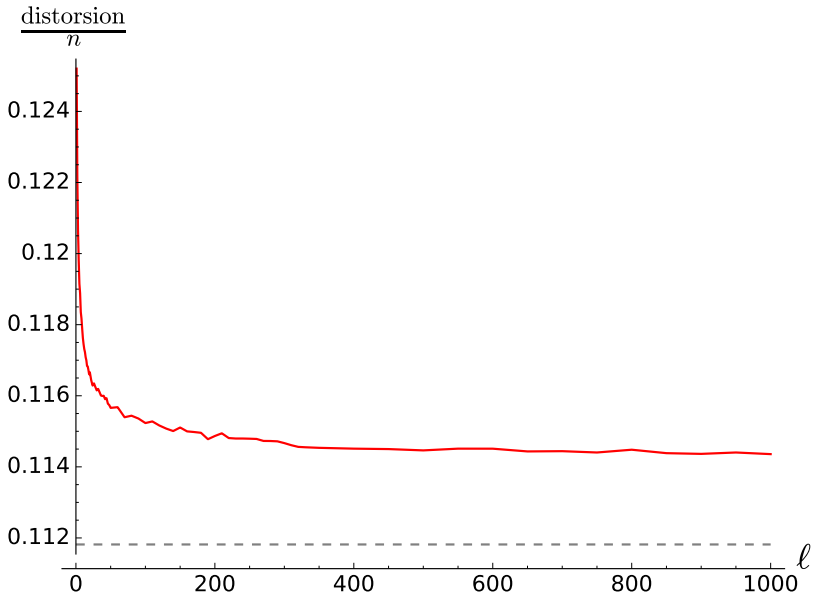


Figure 8.7 – Distorsion du décodage de Tal et Vardy pour un code polaire $[1024, 512]$ en fonction de la taille ℓ de la liste de décodage.

Remarque 8.7.1. Remarquons que le décodage des codes constituants de dimensions ou codimensions faibles que nous avons vu dans la sous-section 8.4 peut être trivialement adapté à un décodage en liste qui retourne les ℓ' mots de code les plus probables pour $\ell' > 0$ fixé. Dans le cas de la dimension faible, il suffit d'ordonner les mots de code du plus probable au moins probable et de sortir les ℓ premiers éléments. Dans le cas d'une codimension faible, il suffit de dépiler les ℓ premiers éléments du tas dans l'algorithme 8.4.1. Nous pouvons alors utiliser ces deux algorithmes dans le décodage en liste de Tal et Vardy pour généraliser celui-ci aux codes $U|U+V$ récursifs.

Chapitre 9

Cryptographie basée sur les codes

9.1 Les enjeux de la cryptographie post-quantique

Pour garantir la sécurité d'un crypto-système, on le réduit à un problème mathématique jugé calculatoirement difficile. La plupart des crypto-systèmes utilisés en pratique aujourd'hui reposent essentiellement sur des problèmes de théorie des nombres. Parmi ces problèmes, on peut citer le problème du logarithme discret sur lequel repose par exemple la sécurité de l'échange de clé de Diffie-Hellman [DH76] ou bien le problème de factorisation sur lequel repose la sécurité du chiffrement ou de la signature RSA [RSA78]. Par la suite, des versions utilisant des courbes elliptiques ont permis de diminuer significativement les tailles des clés. La mode était alors à ces structures mathématiques particulières. En 2005, la NSA (*National Security Agency*) recommandait d'ailleurs exclusivement l'utilisation de courbes elliptiques pour l'échange de clés ou les signatures. Cependant 10 ans plus tard, ceux-ci firent une annonce qui bouleversa cet équilibre :

> *"Unfortunately, the growth of elliptic curve use has bumped up against the fact of* ***continued progress in the research on quantum computing****, which has made it clear that* ***elliptic curve cryptography is not the long term solution many once hoped it would be****. [...] For those customers who are looking for mitigations to perform while the new algorithm suite is developed and implemented into products, there are several things they can do. First, it is prudent to use larger key sizes in algorithms [...] in many systems (especially, smaller scale systems). Additionally, IAD customers using layered commercial solutions to protect classified national security information with a long intelligence life* ***should begin implementing a layer of quantum resistant protection****. Such protection may be implemented today through the use of large symmetric keys and specific secure protocol standards."* [NSA15]

Par cette annonce, la NSA recommande aux entreprises de se préparer à l'arrivée des ordinateurs quantiques car une menace pèse sur le paradigme de sécurité fondé sur la théorie des nombres. En effet, un attaquant possédant un calculateur quantique disposerait d'un avantage considérable par rapport à un attaquant ne possédant qu'une machine classique. Par exemple, l'algorithme de Shor proposé en 1994 [Sho94] permet de résoudre le problème du logarithme discret ou celui de la factorisation en un temps polynomial avec un ordinateur quantique. L'algorithme de Shor n'est pas le seul algorithme quantique à menacer la cryptographie : parmi les plus connus, on peut notamment citer l'algorithme de Grover [Gro96] ou celui de Simon [Sim94].

La menace des ordinateurs quantiques a longtemps été ignorée car peu de gens croyaient réellement que de telles machines puissent exister un jour. Mais à force de persévérance et d'investissements, ils pourraient finalement voir le jour dans un avenir relativement proche. Il est estimé qu'un ordinateur quantique aurait un intérêt significatif par rapport

à un ordinateur classique dès lors qu'il possèderait un processeur quantique de 50 qubits (l'équivalent quantique du bit). Ce seuil est appelé le *seuil théorique de la suprématie quantique*. Il faut toutefois manipuler cette expression avec précaution... En effet, il est censé décrire l'instant où un ordinateur quantique pourra effectuer des taches qu'aucun ordinateur classique ne peut réaliser en un temps raisonnable. Mais les processeurs quantiques construits à ce jour sont extrêmement instables : pour construire un processeur quantique tolérant aux fautes, il faut généralement démultiplier le nombre de qubits physiques.

En 2019, Google (avec la NASA et D-WAVE) a présenté un processeur de 53 qubits. Ce processeur a même pu effectuer une opération en quelques minutes là où un ordinateur classique l'aurait effectuée en plusieurs dizaines de milliers d'années, dépassant ainsi le fameux seuil de suprématie quantique. Toutefois, ce résultat fut démenti par IBM presque aussitôt après en arguant que l'algorithme utilisé par la machine classique est trop naïf et qu'il est possible d'en trouver un autre qui effectuerait l'opération en quelques jours.

D'autres pays et grandes entreprises se sont lancés dans la course à l'ordinateur quantique avec plus ou moins de succès. Notamment en France, le CEA a récemment lancé un projet ayant pour objectif de construire un processeur quantique à 100 qubits.

Pour anticiper la menace que représente l'existence d'un ordinateur quantique, des cryptologues s'attèlent à imaginer une nouvelle cryptographie résistante à des attaques quantiques. La cryptographie dite *post-quantique* se divise actuellement en cinq grands domaines :

- la cryptographie basée sur les réseaux euclidiens ;
- la cryptographie basée sur les codes correcteurs d'erreurs ;
- la cryptographie basée sur les polynômes multivariés ;
- la cryptographie basée sur les isogénies de courbes elliptiques ;
- la cryptographie basée sur les fonctions de hachages.

En 2017, l'institut américain des standards et de la technologie (NIST ou *National Institute of Standards and Technologie*) a lancé une compétition internationale pour la création des nouveaux standards de cryptographie qui devront être à l'épreuve des calculateurs quantiques. Toutes l'actualité concernant cette compétition se trouve sur le site https://csrc.nist.gov/projects/post-quantum-cryptography. Cet appel a été entendu de part le monde et ce n'est pas moins de 69 propositions qui ont été soumises au NIST le 4 décembre 2017. Parmi ces crypto-systèmes, on retrouve l'ensemble des cinq grands domaines de la cryptographie post-quantique. Le 30 Janvier 2019, le NIST annonçait les 26 candidats retenus pour le second tour et encore une fois, on retrouve les cinq grands problèmes mathématiques.

Parmi les problèmes difficiles auxquels peuvent se réduire des crypto-systèmes post-quantiques, nous pouvons citer :

- le décodage générique de codes linéaires (binaires, q-aires ou même en métrique rang) [Pra62, Leo82, LB88, Ste88, Dum91, Bar97a, BLP11, MMT11, BJMM12, MO15, Hir16, GKH17, BM17b, BM18] ;
- le problème LPN (*Learning from Parity with Noise*) [BKW03, LF06, GJL14, ZJW16, EKM17] ;
- le problème k-liste ou presque k-liste [BM17a] ;
- le problème SVP (*Shortest Vector Problem*) dans les réseaux euclidiens [BGJ15, BDGL15, Laa15, BDGL16] ;
- le problème LWE (*Learning With Errors Problem*) ; [CN11, AFFP14, DTV15, GJS15, AGVW17] ;

Ces problèmes ont la particularité de tous faire appel à un problème de recherche de presque-collisions. Dans cette thèse, nous nous attelons à améliorer la résolution de ce problème.

9.2 Une cryptographie basée sur les codes

Le chiffrement de McEliece. Les codes correcteurs d'erreurs permettent de construire une cryptographie à clé publique. Les prémices de cette cryptographie ont vu le jour en 1978 avec le chiffrement à clé publique de McEliece [McE78]. Ce crypto-système est paramétré par trois entiers n, k et w. La clé privée est un triplet $(\mathbf{G}, \mathbf{S}, \mathbf{P})$ où :

— $\mathbf{G} \in \mathbb{F}_2^{k\times n}$ est la matrice génératrice d'un code de Goppa binaire dont un algorithme de décodage corrigeant jusqu'à w erreurs est connu ;

— $\mathbf{P} \in \mathbb{F}_2^{n\times n}$ est une matrice de permutation aléatoire ;

— $\mathbf{S} \in \mathbb{F}_2^{k\times k}$ est une matrice non-singulière aléatoire

et la clé publique est $(\mathbf{G}_{\text{pub}}, w)$ où $\mathbf{G}_{\text{pub}} := \mathbf{SGP}$. Le chiffrement d'un message $\mathbf{m} \in \mathbb{F}_2^k$ consiste alors à coder $\mathbf{m}$ dans le code permuté généré par $\mathbf{G}_{\text{pub}}$ puis à bruiter le mot de code obtenu avec une erreur aléatoire de poids w ; autrement dit, $\mathbf{c} := \mathbf{mG}_{\text{pub}} + \mathbf{e}$ où $\mathbf{e}$ est tiré uniformément dans l'ensemble des vecteurs binaires de poids w. Le déchiffrement de $\mathbf{c}$ consiste essentiellement à décoder $\mathbf{cP}^{-1}$ pour obtenir $\mathbf{mSG}$. On retrouve alors $\mathbf{m}$ en inversant $\mathbf{S}$ ainsi que k colonnes inversibles de $\mathbf{G}$.

La sécurité du chiffrement de McEliece repose essentiellement sur la difficulté de deux problèmes :

(1) distinguer $\mathbf{G}_{\text{pub}}$ d'une matrice tirée uniformément dans $\mathbb{F}_2^{k\times n}$; autrement dit, distinguer un code de Goppa binaire "masqué" d'un code aléatoire.

(2) décoder un mot d'un code aléatoire $[n, k]_2$ contenant exactement w erreurs.

Le premier problème a été résolu dans [FGO+13] pour des rendements suffisamment proches de 1. Cependant, il continue à faire ses preuves pour d'autres rendements. Dans ce chapitre, nous nous intéresserons d'avantage au second problème. Une version décisionnelle de celui-ci a été montrée NP-complète dans [BMvT78]. Ce résultat permet de montrer que le problème du décodage générique est NP-complet dans le pire cas. En moyenne sur les entrées, nous pouvons seulement dire qu'il est NP-difficile. Mais ce qui rend ce problème intéressant pour la cryptographie post-quantique est qu'il semble rester difficile même sous l'hypothèse d'un calculateur quantique. Le premier algorithme pour résoudre le problème du décodage générique date de 1962 et est dû à Prange ; c'est d'ailleurs avec cet algorithme que McEliece a mesuré ses tailles de clés. Depuis, de nombreuses méthodes ont été proposées mais celles-ci ont eu pour effet de diminuer de plus de 20% l'exposant de la complexité asymptotique du décodage de Prange. Ces décodages ne remettent donc pas en question le chiffrement de McEliece mais impliquent essentiellement une révision des tailles des clés.

Le chiffrement de Niederreiter. Une version duale du crypto-système de McEliece a été proposée par Niederreiter dans [Nie86]. Dans cette version, la clé privée est $(\mathbf{S}, \mathbf{H}, \mathbf{P})$ où :

— $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$ est une matrice de parité d'un code Goppa binaire dont un algorithme de décodage corrigeant jusqu'à w erreurs est connu ;

— $\mathbf{P} \in \mathbb{F}_2^{n\times n}$ est une matrice de permutation aléatoire ;

— $\mathbf{S} \in \mathbb{F}_2^{(n-k)\times(n-k)}$ est tel que $\mathbf{H}_{\text{pub}} := \mathbf{SHP}$ soit systématique

et la clé publique est $(\mathbf{H}_{\text{pub}}, w)$. Dans le crypto-système de Niederreiter, les messages sont des mots de $\mathbb{F}_2^n$ de poids w (des solutions relativement efficaces existent pour effectuer une bijection entre $\mathbb{F}_2^k$ et $\mathcal{S}(\mathbf{0}, w) := \{\mathbf{x} \in \mathbb{F}_2^n : |\mathbf{x}| = w\}$). Le chiffré d'un message $\mathbf{e} \in \mathcal{S}(\mathbf{0}, w)$ est le syndrome $\mathbf{s} := \mathbf{H}_{\text{pub}}\mathbf{e}^\top$. Le déchiffrement de $\mathbf{s}$ consiste essentiellement à décoder le syndrome $\mathbf{S}^{-1}\mathbf{s} := \mathbf{HPe}^\top$ pour obtenir $\mathbf{Pe}^\top$. On retrouve alors $\mathbf{e}$ en inversant $\mathbf{P}$.

Il a été montré que le chiffrement de Niederreiter est équivalent à celui de McEliece. Toutefois, cette version duale permet de réduire significativement la taille de la clé publique

car $\mathbf{H}_{\text{pub}}$ étant sous forme systématique, seules les k dernières colonnes sont nécessaires. Cependant, l'inconvénient de cette version est qu'elle nécessite une bijection entre $\mathbb{F}_2^k$ et $\mathcal{S}(\mathbf{0}, w)$ qui augmente le coût du chiffrement et du déchiffrement.

Le document [BCL+17] décrit une version du chiffrement de Niderreiter qui a été proposée à la compétition du NIST pour les futurs standards de la cryptographie post-quantique. Ce chiffrement, appelé *Classic-McEliece*, a été calibré pour résister aux décodages (classiques ou quantiques) connus à ce jour. Il utilise ainsi des codes de Goppa binaires $[6960, 5296]_2$ capables de corriger 119 erreurs. Le défaut majeur dont souffre ce crypto-système est sa taille de clé de 1.1Mo ; toutefois, cela ne l'a pas empêché de passer au second tour de la compétition du NIST.

Quelques variantes du chiffrement de McEliece. Divers codes algébriques ont été proposés pour remplacer les codes de Goppa binaires dans le crypto-système de McEliece ou de Niederreiter (ou des versions proches de ceux-ci) :

- des codes de Reed-Solomon généralisés [Nie86] (cassé dans [SS92]) ;
- des codes de Reed-Muller [Sid94] (cassé dans [MS07]) ;
- des codes géométriques [JM96] (cassé dans [FM08, CMCP14]) ;
- des sous-codes de codes de Reed-Solomon généralisés [BL05] (cassé dans [Wie09]) ;
- des sous-codes quasi-cycliques de codes BCH [Gab05] (cassé dans [OTD10]) ;
- des codes alternants quasi-cycliques [BCGO09] (partiellement cassé dans [FOPT10]) ;
- des codes de Goppa dyadiques [MB09] (partiellement cassé dans [FOPT10, FOP+14, CT19])
- des codes de Goppa "sauvages" non-binaires [BLP10] (partiellement cassé dans [COT14, FPdP14]) ;
- des codes de Srivastava généralisés [BBB+17] (chiffrement proposé à la compétition du NIST sous le nom de *DAGS*) (cassé dans [BC18, BBCO19]) ;
- d'autres codes de Goppa binaires [CBB+17] (chiffrement proposé à la compétition du NIST sous le nom de *BIG-QUAKE*) ;

L'objectif de ces constructions est essentiellement de réduire les tailles des clés. Comme nous pouvons le constater, la plupart des propositions citées plus haut ont été cassées. Ainsi, excepté *Classic-McEliece*, aucune construction utilisant des codes algébriques n'a été retenue au second tour de la compétition du NIST.

Plutôt que d'utiliser des codes algébriques, certains ont proposé des crypto-systèmes basés sur des codes probabilistes. Par exemple, dans [LJ12], il a été proposé d'utiliser des codes convolutifs mais cette solution a été cassée dans [LT13]. Les codes polaires ont donné naissance à quelques crypto-systèmes [HSA13, SK14, HSEA14, HAE15] ; cependant, aucun n'a résisté à l'attaque de [BCD+16]. Dans [MRSA00, BCGM07, BBC08, BBC12, BBC13, Bal14], il a été proposé d'utiliser des codes LDPC ou des codes LDPC quasi-cycliques dans des chiffrements de type McEliece. Les attaques [OTD08, **?**] ont montré que ces choix n'étaient pas non plus judicieux. Toutefois, le chiffrement *LEDApkc* [BBC+17b] et le schéma d'échange de clés *LEDAkem* [BBC+17a] sont deux crypto-systèmes utilisant des codes LDPC quasi-cycliques qui ont été proposés au NIST et qui ne sont pas affectés par les attaques connues sur les codes LDPC. Ces deux crypto-systèmes ont fusionné en *LEDAcrypt* [BBC+19] qui est actuellement au second tour de la compétition du NIST.

Des codes très proches des codes LDPC quasi-cycliques sont aujourd'hui très appréciés en cryptographie basée sur les codes. En effet, les codes QC-MDPC (*Quasi-Cyclic Moderate-Density Parity-Check*) ont fait l'objet de deux soumissions au NIST : *QC-MDPC-KEM* [YEK+17] et *BIKE* [AAB+17a]. Ce dernier est encore dans la course aujourd'hui. Le

tableau 9.1 donne les paramètres et la taille de clé publique de *BIKE* en fonction des différents niveaux de sécurité exigés par le NIST.

niveau de sécurité	n	k	w	poids des lignes de la matrice de parité	taille de la clé publique
128 bits	20326	10163	134	142	1.25 ko
192 bits	39706	19853	199	206	2.5 ko
256 bits	65498	32749	264	274	4.1 ko

Tableau 9.1 – Paramètres de *BIKE*.

Le but du tableau 9.1 est de nous donner une idée des tailles cryptographiques que nous cherchons à atteindre dans nos algorithmes de décodage générique.

La métrique rang. Il existe un sous-domaine de la cryptographie basée sur les codes qui utilise des codes dans une autre métrique que celle de Hamming. Ces codes sont des sous-espaces vectoriels de matrices sur $\mathbb{F}_q$ et la métrique considérée est le rang de la différence de deux éléments. Ces codes sont à l'origine de plusieurs soumissions à la compétition du NIST :

— *LAKE*, *LOCKER*, *Ouroboros-R* [ABD+17a, ABD+17b, AAB+17b] qui ont fusionné en *ROLLO* [ABD+19] au second tour ;

— *RQC* [AAB+17c] encore présent au second tour ;

— *RankSign* [AGH+17] qui a été cassé dans [DT18].

La métrique rang a deux avantages sur la métrique de Hamming. Le premier est qu'elle permet d'obtenir des tailles de clés significativement plus petites. Le second avantage est que les algorithmes de décodages génériques sont plus difficiles à construire : aujourd'hui, on ne sait pas faire beaucoup mieux que l'algorithme naïf (équivalent de Prange en métrique de Hamming). Toutefois, ces derniers temps, des attaques utilisant des bases de Gröbner ont été dévastatrices pour le paradigme de sécurité de la métrique rang [DT18, BBB+19].

Dans ce chapitre, nous ne traitons pas le décodage générique en métrique rang bien qu'il serait intéressant d'étudier le problème de presque-collisions dans cette métrique et voir l'impact que cela pourrait avoir sur le décodage.

Décodage à grande distance. Dans le domaine de la cryptographie basée sur les codes, un nouveau venu chamboule les paradigmes usuels : *WAVE* [DST19]. Avec *RankSign*, c'est une des rares signatures citée dans cette section pour la simple et bonne raison que très peu de schémas de signatures utilisant des codes ont été proposés dans la littérature. *WAVE* est en fait la seule signature encore debout aujourd'hui. Elle est aussi la seule à suivre le modèle GPV [GPV08] ; hissant ainsi les codes au niveau des réseaux euclidiens dans l'univers de la cryptographie post-quantique. L'originalité de *WAVE* réside dans le fait que, contrairement aux autres crypto-systèmes basés sur des codes, le problème de décodage générique auquel elle est associée est un décodage à grande distance. En effet, au lieu de rechercher une erreur de poids faible ($w \ll n - \frac{n}{q}$), il est recherché ici une erreur de poids élevé ($w \gg n - \frac{n}{q}$). Notons que dans les espaces binaires, le problème du décodage à faible distance et celui du décodage à grande distance sont équivalents. Cependant, sur un corps de taille strictement supérieure à 2, ils sont fondamentalement différents. Dans *WAVE*, le corps choisi est le corps ternaire. Le tableau 9.2 donne les paramètres ainsi que les tailles des clés publiques et des signatures de *WAVE* pour différents niveaux de sécurité.

niveau de sécurité	n	k	w	taille de la clé publique	taille des signatures
128 bits	8492	5605	7979	3.21Mo	0.67ko
192 bits	12738	8407	11968	7.21Mo	1.01ko
256 bits	16984	11209	15958	12.82Mo	1.34ko

Tableau 9.2 – Paramètres de *WAVE*.

Bibliographie

[AAB+17a] Carlos Aguilar Melchor, Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Shay Gueron, Tim Güneysu, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, Jean-Pierre Tillich, and Gilles Zémor. BIKE. First round submission to the NIST post-quantum cryptography call, November 2017.

[AAB+17b] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Adrien Hauteville, and Gilles Zémor. Ouroboros-R. First round submission to the NIST post-quantum cryptography call, November 2017.

[AAB+17c] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, and Gilles Zémor. Rank quasi cyclic (RQC). First round submission to the NIST post-quantum cryptography call, November 2017.

[ABD+17a] Nicolas Aragon, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Adrien Hauteville, Olivier Ruatta, Jean-Pierre Tillich, and Gilles Zémor. LAKE – Low rAnk parity check codes Key Exchange. First round submission to the NIST post-quantum cryptography call, November 2017.

[ABD+17b] Nicolas Aragon, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Adrien Hauteville, Olivier Ruatta, Jean-Pierre Tillich, and Gilles Zémor. LOCKER – LOw rank parity ChecK codes EncRyption. First round submission to the NIST post-quantum cryptography call, November 2017.

[ABD+19] Nicolas Aragon, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Adrien Hauteville, Olivier Ruatta, Jean-Pierre Tillich, Gilles Zémor, Carlos Aguilar Melchor, Slim Bettaieb, Loïc Bidoux, Magali Bardet, and Ayoub Otmani. ROLLO (merger of Rank-Ouroboros, LAKE and LOCKER). Second round submission to the NIST post-quantum cryptography call, March 2019.

[AFFP14] Martin Albrecht, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. Lazy Modulus Switching for the BKW Algorithm on LWE. In *Proceedings of the 17th International Conference on Public-Key Cryptography — PKC 2014 - Volume 8383*, pages 429–445, New York, NY, USA, 2014. Springer-Verlag New York, Inc.

[AGH+17] Nicolas Aragon, Philippe Gaborit, Adrien Hauteville, Oliver Ruatta, and Gilles Zémor. Ranksign – a signature proposal for the NIST's call. First round submission to the NIST post-quantum cryptography call, November 2017.

[AGVW17] Martin Albrecht, Florian Göpfert, Fernando Virdia, and Thomas Wunderer. Revisiting the expected cost of solving usvp and applications to lwe. pages 297–322, November 2017.

[Arı09] Erdal Arıkan. Channel polarization : a method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Trans. Inform. Theory*, 55(7) :3051–3073, 2009.

[Bal14] Marco Baldi. *QC-LDPC Code-Based Cryptography*. Springer Science & Business, 2014.

[Bar97a] Alexander Barg. Complexity issues in coding theory. *Electronic Colloquium on Computational Complexity*, October 1997.

[Bar97b] Alexander Barg. Minimum distance decoding algorithms for linear codes. In *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, 12th International Symposium, AAECC-12, Toulouse, France, June 23-27, 1997, Proceedings*, volume 1255 of *LNCS*, pages 1–14. Springer, 1997.

[BBB+17] Gustavo Banegas, Paulo S.L.M Barreto, Brice Odilon Boidje, Pierre-Louis Cayrel, Gilbert Ndollane Dione, Kris Gaj, Cheikh Thiécoumba Gueye, Richard Haeussler, Jean Belo Klamti, Ousmane N'diaye, Duc Tri Nguyen, Edoardo Persichetti, and Jefferson E. Ricardini. DAGS : Key encapsulation for dyadic GS codes. https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/submissions/DAGS.zip, November 2017. First round submission to the NIST post-quantum cryptography call.

[BBB+19] Magali Bardet, Pierre Briaud, Maxime Bros, Philippe Gaborit, Vincent Neiger, Olivier Ruatta, and Jean-Pierre Tillich. An algebraic attack on rank metric code-based cryptosystems, 2019.

[BBC08] Marco Baldi, Marco Bodrato, and Franco Chiaraluce. A new analysis of the McEliece cryptosystem based on QC-LDPC codes. In *Proceedings of the 6th international conference on Security and Cryptography for Networks*, SCN '08, pages 246–262. Springer-Verlag, 2008.

[BBC12] Marco Baldi, Marco Bianchi, and Franco Chiaraluce. Security and complexity of the McEliece cryptosystem based on QC-LDPC codes, 2012. arXiv :1109.5827.

[BBC13] Marco Baldi, Marco Bianchi, and Franco Chiaraluce. Security and complexity of the McEliece cryptosystem based on QC-LDPC codes. *IET Information Security*, 7(3) :212–220, September 2013.

[BBC+17a] Marco Baldi, Alessandro Barenghi, Franco Chiaraluce, Gerardo Pelosi, and Paolo Santini. LEDAkem. First round submission to the NIST post-quantum cryptography call, November 2017.

[BBC+17b] Marco Baldi, Alessandro Barenghi, Franco Chiaraluce, Gerardo Pelosi, and Paolo Santini. LEDApkc. First round submission to the NIST post-quantum cryptography call, November 2017.

[BBC+19] Marco Baldi, Alessandro Barenghi, Franco Chiaraluce, Gerardo Pelosi, and Paolo Santini. LEDAcrypt. Second round submission to the NIST post-quantum cryptography call, January 2019.

[BBCO19] Magali Bardet, Manon Bertin, Alain Couvreur, and Ayoub Otmani. Practical algebraic attack on DAGS. In Marco Baldi, Edoardo Persichetti, and Paolo Santini, editors, *Code-Based Cryptography - 7th International Workshop, CBC 2019, Darmstadt, Germany, May 18-19, 2019, Revised Selected Papers*, volume 11666 of *LNCS*, pages 86–101. Springer, 2019.

[BC18] Élise Barelli and Alain Couvreur. An efficient structural attack on NIST submission DAGS. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology - ASIACRYPT'18*, volume 11272 of *LNCS*, pages 93–118. Springer, December 2018.

[BCD+16] Magali Bardet, Julia Chaulet, Vlad Dragoi, Ayoub Otmani, and Jean-Pierre Tillich. Cryptanalysis of the McEliece public key cryptosystem based on polar

codes. In *Post-Quantum Cryptography 2016*, LNCS, pages 118–143, Fukuoka, Japan, February 2016.

[BCDL20] Rémi Bricout, André Chailloux, Thomas Debris-Alazard, and Matthieu Lequesne. Ternary syndrome decoding with large weights. In Kenneth G. Paterson and Douglas Stebila, editors, *Selected Areas in Cryptography – SAC 2019*, pages 437–466, Cham, January 2020. Springer International Publishing.

[BCGM07] Marco Baldi, Franco Chiaraluce, Roberto Garello, and Francesco Mininni. Quasi-cyclic low-density parity-check codes in the McEliece cryptosystem. In *Communications, 2007. ICC '07. IEEE International Conference on*, pages 951–956, June 2007.

[BCGO09] Thierry P. Berger, Pierre-Louis Cayrel, Philippe Gaborit, and Ayoub Otmani. Reducing key length of the McEliece cryptosystem. In Bart Preneel, editor, *Progress in Cryptology - AFRICACRYPT 2009*, volume 5580 of *LNCS*, pages 77–97, Gammarth, Tunisia, June 21-25 2009.

[BCJR74] Lalit Bahl, John Cocke, Frederick Jelinek, and Josef Raviv. Optimal decoding of linear codes for minimizing symbol error rate (corresp.). *IEEE Transactions on information theory*, 20(2) :284–287, 1974.

[BCL+17] Daniel J. Bernstein, Tung Chou, Tanja Lange, Ingo von Maurich, Rafael Mizoczki, Ruben Niederhagen, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, and Wang Wen. Classic McEliece : conservative code-based cryptography. https://classic.mceliece.org, November 2017. First round submission to the NIST post-quantum cryptography call.

[BDGL15] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. IACR Cryptology ePrint Archive, Report 2015/1128, 2015. http://eprint.iacr.org/2015/1128.

[BDGL16] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 10–24. SIAM, 2016.

[BGJ15] Anja Becker, Nicolas Gama, and Antoine Joux. Speeding-up lattice sieving without increasing the memory, using sub-quadratic nearest neighbor search. IACR Cryptology ePrint Archive, Report 2015/522, 2015. http://eprint.iacr.org/2015/522.

[BGT93] Claude Berrou, Alain Glavieux, and Punya Thitimajshima. Near shannon limit error-correcting coding and decoding : Turbo-codes. 1. In *Proceedings of ICC '93 - IEEE International Conference on Communications*, volume 2, pages 1064–1070 vol.2, 1993.

[BJMM12] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In *Advances in Cryptology - EUROCRYPT 2012*, LNCS. Springer, 2012.

[BKW03] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *Journal of the ACM (JACM)*, 50(4) :506–519, 2003.

[BL05] Thierry P. Berger and Pierre Loidreau. How to mask the structure of codes for a cryptographic use. *Des. Codes Cryptogr.*, 35(1) :63–79, 2005.

[BLP10] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Wild McEliece. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *LNCS*, pages 143–158, 2010.

[BLP11] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Smaller decoding exponents : ball-collision decoding. In *Advances in Cryptology - CRYPTO 2011*, volume 6841 of *LNCS*, pages 743–760, 2011.

[BM17a] Leif Both and Alexander May. The approximate k-list problem. *IACR Trans. Symmetric Cryptol.*, 2017(1) :380–397, 2017.

[BM17b] Leif Both and Alexander May. Optimizing BJMM with Nearest Neighbors : Full Decoding in $2^{2/21n}$ and McEliece Security. In *WCC Workshop on Coding and Cryptography*, September 2017.

[BM18] Leif Both and Alexander May. Decoding linear codes with high error rate and its impact for LPN security. In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography 2018*, volume 10786 of *LNCS*, pages 25–46, Fort Lauderdale, FL, USA, April 2018. Springer.

[BMvT78] Elwyn Berlekamp, Robert McEliece, and Henk van Tilborg. On the inherent intractability of certain coding problems. *IEEE Trans. Inform. Theory*, 24(3) :384–386, May 1978.

[BSFZ18] Alexander Bazarsky, Eran Sharon, Omer Fainzilber, and Ran Zamir. Adaptive bit-flipping decoder based on dynamic error information, June 2018. US Patent App. 15/896,440.

[Can16] Rodolfo Canto Torres. CaWoF, C library for computing asymptotic exponents of generic decoding work factors, 2016. https ://gforge.inria.fr/projects/cawof/.

[Can17] Rodolfo Canto Torres. Asymptotic analysis of ISD algorithms for the $q-$ary case. In *Proceedings of the Tenth International Workshop on Coding and Cryptography WCC 2017*, September 2017.

[CBB+17] Alain Couvreur, Magali Bardet, Élise Barelli, Olivier Blazy, Rodolfo Canto Torres, Phillipe Gaborit, Ayoub Otmani, Nicolas Sendrier, and Jean-Pierre Tillich. BIG QUAKE. https://bigquake.inria.fr, November 2017. NIST Round 1 submission for Post-Quantum Cryptography.

[CD05] Enver Cavus and Babak Daneshrad. A performance improvement and error floor avoidance technique for belief propagation decoding of ldpc codes. In *2005 IEEE 16th International Symposium on Personal, Indoor and Mobile Radio Communications*, volume 4, pages 2386–2390. IEEE, 2005.

[CF02] Jinghu Chen and Marc P. C. Fossorier. Density evolution for two improved BP-based decoding algorithms of LDPC codes. *IEEE Communications Letters*, 6 :208–210, May 2002.

[CGW07] Andres I. Vila Casado, Miguel Griot, and Richard D. Wesel. Informed dynamic scheduling for belief-propagation decoding of ldpc codes. In *2007 IEEE International Conference on Communications*, pages 932–937. IEEE, 2007.

[Che13] Tso-Cho Chen. Adaptive-weighted multibit-flipping decoding of lowdensity parity-check codes based on ordered statistics. *Communications, IET*, 7 :1517–1521, September 2013.

[Clu06] Mathieu Cluzeau. *Reconstruction of a communication scheme*. Theses, Ecole Polytechnique X, November 2006.

[CMCP14] Alain Couvreur, Irene Márquez-Corbella, and Ruud Pellikaan. A polynomial time attack against algebraic geometry code based public key cryptosystems.

In *Proc. IEEE Int. Symposium Inf. Theory - ISIT 2014*, pages 1446–1450, June 2014.

[CN11] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0 : Better Lattice Security Estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073, pages 1–20, Berlin, Heidelberg, December 2011. Springer.

[COT14] Alain Couvreur, Ayoub Otmani, and Jean-Pierre Tillich. New identities relating wild Goppa codes. *Finite Fields Appl.*, 29 :178–197, 2014.

[CS15] Tofar Chang and Yu Su. Dynamic Weighted Bit-Flipping Decoding Algorithms for LDPC Codes. *IEEE Transactions on Communications*, 63 :3950–3963, November 2015.

[CT19] Rodolfo Canto-Torres and Jean-Pierre Tillich. Speeding up decoding a code with a non-trivial automorphism group up to an exponential factor. In *Proc. IEEE Int. Symposium Inf. Theory - ISIT 2019*, pages 1927–1931, 2019.

[Deb19] Thomas Debris-Alazard. *Cryptographie fondée sur les codes : nouvelles approches pour constructions et preuves ; contribution en cryptanalyse*. Theses, Sorbonne Université, December 2019.

[DH76] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6) :644–654, 1976.

[DST19] Thomas Debris-Alazard, Nicolas Sendrier, and Jean-Pierre Tillich. Wave : A new family of trapdoor one-way preimage sampleable functions based on codes. Cryptology ePrint Archive, Report 2018/996, May 2019. https://eprint.iacr.org/2018/996.

[DT17] Thomas Debris-Alazard and Jean-Pierre Tillich. Statistical decoding. preprint, January 2017. arXiv :1701.07416.

[DT18] Thomas Debris-Alazard and Jean-Pierre Tillich. Two attacks on rank metric code-based schemes : Ranksign and an identity-based-encryption scheme. In *Advances in Cryptology - ASIACRYPT 2018*, volume 11272 of *LNCS*, pages 62–92, Brisbane, Australia, December 2018. Springer.

[DTV15] Alexandre Duc, Florian Tramèr, and Serge Vaudenay. Better algorithms for LWE and LWR. *IACR Cryptology ePrint Archive*, 2015 :56, 2015.

[Dum89] Il'ya Dumer. Two decoding algorithms for linear codes. *Probl. Inf. Transm.*, 25(1) :17–23, 1989.

[Dum91] Ilya Dumer. On minimum distance decoding of linear codes. In *Proc. 5th Joint Soviet-Swedish Int. Workshop Inform. Theory*, pages 50–52, Moscow, 1991.

[DWV+16] David Declercq, Chris Winstead, Bane Vasic, Fakhreddine Ghaffari, Predrag Ivanis, and Emmanuel Boutillon. Noise-aided gradient descent bit-flipping decoders approaching maximum likelihood decoding. In *2016 9th International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, pages 300–304. IEEE, 2016.

[EKM17] Andre Esser, Robert Kübler, and Alexander May. LPN decoded. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017*, volume 10402 of *LNCS*, pages 486–514, Santa Barbara, CA, USA, August 2017. Springer.

[Eli55] Peter Elias. Coding for noisy channels. *IRE Conv. Rec.*, 3 :37–46, 1955.

[FGO+13] Jean-Charles Faugère, Valérie Gauthier, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich. A distinguisher for high rate McEliece cryptosystems. *IEEE Trans. Inform. Theory*, 59(10) :6830–6844, October 2013.

[FM08] Cédric Faure and Lorenz Minder. Cryptanalysis of the McEliece cryptosystem over hyperelliptic curves. In *Proceedings of the eleventh International Workshop on Algebraic and Combinatorial Coding Theory*, pages 99–107, Pamporovo, Bulgaria, June 2008.

[FMI99] Marc P. C. Fossorier, Miodrag Mihaljevic, and Hideki Imai. Reduced complexity iterative decoding of low-density parity check codes based on belief propagation. *Communications, IEEE Transactions on*, 47 :673–680, June 1999.

[FOP+14] Jean-Charles Faugère, Ayoub Otmani, Ludovic Perret, Frédéric de Portzamparc, and Jean-Pierre Tillich. Folding alternant and Goppa codes with non-trivial automorphism groups. to appear in the Transactions on Information Theor, 2014. arXiv :1405.5101 [cs.IT].

[FOPT10] Jean-Charles Faugère, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich. Algebraic cryptanalysis of McEliece variants with compact keys. In *Advances in Cryptology - EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 279–298, 2010.

[FPdP14] Jean-Charles Faugère, Ludovic Perret, and Frédéric de Portzamparc. Algebraic attack against variants of McEliece with Goppa polynomial of a special form. In *Advances in Cryptology - ASIACRYPT 2014*, volume 8873 of *LNCS*, pages 21–41, Kaoshiung, Taiwan, R.O.C., December 2014. Springer.

[Gab05] Philippe Gaborit. Shorter keys for code based cryptography. In *Proceedings of the 2005 International Workshop on Coding and Cryptography (WCC 2005)*, pages 81–91, Bergen, Norway, March 2005.

[Gal63] Robert G. Gallager. *Low Density Parity Check Codes*. M.I.T. Press, Cambridge, Massachusetts, 1963.

[GH04] Feng Guo and Lajos Hanzo. Reliability-ratio based weighted bit-flipping decoding for low-density parity check codes. *Electronics Letters*, 40 :1356–1358, November 2004.

[GJL14] Qian Guo, Thomas Johansson, and Carl Löndahl. Solving LPN using covering codes. In *Advances in Cryptology - ASIACRYPT 2014*, volume 8873 of *LNCS*, pages 1–20. Springer, 2014.

[GJS15] Qian Guo, Thomas Johansson, and Paul Stankovski. Coded-BKW : Solving LWE using lattice codes. In *Advances in Cryptology - CRYPTO 2015*, volume 9215 of *LNCS*, pages 23–42, Santa Barbara, CA, USA, August 2015. Springer.

[GKH17] Cheikh Thiécoumba Gueye, Jean Belo Klamti, and Shoichi Hirose. Generalization of BJMM-ISD using may-ozerov nearest neighbor algorithm over an arbitrary finite field \mathbb f_q. In *Codes, Cryptology and Information Security - Second International Conference, C2SI 2017, Rabat, Morocco, April 10-12, 2017, Proceedings - In Honor of Claude Carlet*, pages 96–109, 2017.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 197–206. ACM, 2008.

[Gro96] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proc. 28th Annual ACM Symposium on the Theory of Computation*, pages 212–219, New York, NY, 1996. ACM Press, New York.

[Gui04] Frédéric Guilloud. *Architecture générique de décodeur de codes LDPC*. Theses, Télécom ParisTech, July 2004.

[HAE15] Reza Hooshmand, Mohammad Reza Aref, and Taraneh Eghlidos. Secret key cryptosystem based on non-systematic polar codes. *Wireless Personal Communications*, 84(2) :1345–1373, 2015.

[Hir16] Shoichi Hirose. May-ozerov algorithm for nearest-neighbor problem over $\mathcal{O}_q$ and its application to information set decoding. In *Innovative Security Solutions for Information Technology and Communications - 9th International Conference, SECITC 2016, Bucharest, Romania, June 9-10, 2016, Revised Selected Papers*, pages 115–126, 2016.

[HJ10] Nicholas Howgrave-Graham and Antoine Joux. New generic algorithms for hard knapsacks. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010*, volume 6110 of *LNCS*. Sringer, 2010.

[HSA13] Reza Hooshmand, M Koochak Shooshtari, and Mohammad Reza Aref. Secret key cryptosystem based on polar codes over binary erasure channel. In *2013 10th International ISC Conference on Information Security and Cryptology (ISCISC)*, pages 1–6. IEEE, 2013.

[HSEA14] R Hooshmand, M Koochak Shooshtari, T Eghlidos, and MR Aref. Reducing the key length of McEliece cryptosystem using polar codes. In *2014 11th International ISC Conference on Information Security and Cryptology (ISCISC)*, pages 104–108. IEEE, 2014.

[HU12] Ryoji Haga and Shogo Usami. Multi-bit flip type gradient descent bit flipping decoding using no thresholds. *2012 International Symposium on Information Theory and its Applications*, pages 6–10, 2012.

[Jab01] Abdulrahman Al Jabri. A statistical decoding algorithm for general linear block codes. In Bahram Honary, editor, *Cryptography and coding. Proceedings of the 8th IMA International Conference*, volume 2260 of *LNCS*, pages 1–8, Cirencester, UK, December 2001. Springer.

[JM96] Heeralal Janwa and Oscar Moreno. McEliece public key cryptosystems using algebraic-geometric codes. *Des. Codes Cryptogr.*, 8(3) :293–307, 1996.

[JP17] Jaehwan Jung and In-Cheol Park. Multi-bit flipping decoding of LDPC codes for NAND storage systems. *IEEE Communications Letters*, 21(5) :979–982, 2017.

[JVSV03] Christopher R. Jones, Esteban L. Valles, Michael Smith, and John Villasenor. Approximate-MIN constraint node updating for LDPC code decoding. pages 157–162 Vol.1, November 2003.

[JZSC05] Ming Jiang, Chunming Zhao, Zhihua Shi, and Yu Chen. An improvement on the modified weighted bit flipping decoding algorithm for LDPC codes. *Communications Letters, IEEE*, 9 :814–816, October 2005.

[Kor09] Satish Babu Korada. *Polar Codes for Channel and Source Coding*. PhD thesis, 'Ecole Polytechnique Fédérale de Lausanne (EPFL), July 2009.

[KP83] Jin H. Kim and Judea Pearl. A computational model for causal and diagnostic reasoning in inference systems. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'83, pages 190–193, San Francisco, CA, USA, 1983. Morgan Kaufmann Publishers Inc.

[KRU13] Shrinivas Kudekar, Tom Richardson, and Rüdiger L. Urbanke. Spatially coupled ensembles universally achieve capacity under belief propagation. *IEEE Transactions on Information Theory*, 59(12) :7761–7813, 2013.

[Laa15] Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In *Advances in Cryptology - CRYPTO 2015*, volume 9215 of *LNCS*, pages 3–22, Santa Barbara, CA, USA, August 2015. Springer.

[LB88] Pil J. Lee and Ernest F. Brickell. An observation on the security of McEliece's public-key cryptosystem. In *Advances in Cryptology - EUROCRYPT'88*, volume 330 of *LNCS*, pages 275–280. Springer, 1988.

[LDG+17] Khoa Le, David Declercq, Fakhreddine Ghaffari, Lounis Kessal, Oana Boncalo, and Valentin Savin. Variable-node-shift based architecture for probabilistic gradient descent bit flipping on qc-ldpc codes. *IEEE Transactions on Circuits and Systems I : Regular Papers*, 65(7) :2183–2195, 2017.

[LdL12] Jingjing Liu and Rodrigo C. de Lamare. Low-latency reweighted belief propagation decoding for ldpc codes. *IEEE Communications Letters*, 16(10) :1660–1663, 2012.

[Leo82] Jeffrey Leon. Computing automorphism groups of error-correcting codes. *IEEE Trans. Inform. Theory*, 28(3) :496–511, 1982.

[LF06] Éric Levieil and Pierre-Alain Fouque. An improved LPN algorithm. In *Proceedings of the 5th international conference on Security and Cryptography for Networks*, volume 4116 of *LNCS*, pages 348–359. Springer, 2006.

[LGK+18] Khoa Le, Fakhreddine Ghaffari, Lounis Kessal, David Declercq, Emmanuel Boutillon, Chris Winstead, and Bane Vasic. A Probabilistic Parallel Bit-Flipping Decoder for Low-Density Parity-Check Codes. *IEEE Transactions on Circuits and Systems I : Regular Papers*, pages 1–14, July 2018.

[LJ12] Carl Löndahl and Thomas Johansson. A new version of McEliece PKC based on convolutional codes. In *Information and Communications Security, ICICS*, volume 7168 of *LNCS*, pages 461–470. Springer, 2012.

[LLYS09] Guangwen Li, Dashe Li, Wang Yuling, and Wenyan Sun. Improved parallel weighted bit flipping decoding of finite geometry LDPC codes. *2009 4th International Conference on Communications and Networking in China, CHINACOM 2009*, August 2009.

[LT13] Grégory Landais and Jean-Pierre Tillich. An efficient attack of a McEliece cryptosystem variant based on convolutional codes. In P. Gaborit, editor, *Post-Quantum Cryptography'13*, volume 7932 of *LNCS*, pages 102–117. Springer, June 2013.

[Mac02] David J. C. MacKay. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, New York, NY, USA, 2002.

[Mas17] Kennedy T. F. Masunda. *Threshold based multi-bit flipping decoding of binary LDPC codes*. Theses, University of the Witwatersrand, August 2017.

[Mat93] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In *Advances in Cryptology - EUROCRYPT'93*, volume 765 of *LNCS*, pages 386–397, Lofthus, Norway, May 1993. Springer.

[MB09] Rafael Misoczki and Paulo Barreto. Compact McEliece keys from Goppa codes. In *Selected Areas in Cryptography*, Calgary, Canada, August 13-14 2009.

[McE78] Robert J. McEliece. *A Public-Key System Based on Algebraic Coding Theory*, pages 114–116. Jet Propulsion Lab, 1978. DSN Progress Report 44.

[Meu12] Alexander Meurer. *A Coding-Theoretic Approach to Cryptanalysis*. PhD thesis, Ruhr University Bochum, November 2012.

[MF05] Nenad Miladinovic and Marc P. C. Fossorier. Improved bit flipping decoding of low-density parity check codes. *IEEE Transactions on Information Theory*, 51 :1594–1606, January 2005.

[MMT11] Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in $O(2^{0.054n})$. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology - ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 107–124. Springer, 2011.

[MO15] Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, volume 9056 of *LNCS*, pages 203–228. Springer, 2015.

[Moo05] Todd K. Moon. *Error Correction Coding : Mathematical Methods and Algorithms*. Wiley-Interscience, New York, NY, USA, 2005.

[MRSA00] Chris Monico, Joachim Rosenthal, and Amin Shokrollahi A. Using low density parity check codes in the McEliece cryptosystem. In *Proc. IEEE Int. Symposium Inf. Theory - ISIT*, page 215, Sorrento, Italy, 2000.

[MS07] Lorenz Minder and Amin Shokrollahi. Cryptanalysis of the Sidelnikov cryptosystem. In *Advances in Cryptology - EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 347–360, Barcelona, Spain, 2007.

[Nie86] Harald Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Problems of Control and Information Theory*, 15(2) :159–166, 1986.

[NSA15] NSA National Security Agency. Cryptography today, August 2015. https://www.nsa.gov/ia/programs/suitebcryptography.

[OTD08] Ayoub Otmani, Jean-Pierre Tillich, and Léonard Dallot. Cryptanalysis of McEliece cryptosystem based on quasi-cyclic LDPC codes. In *Proceedings of First International Conference on Symbolic Computation and Cryptography*, pages 69–81, Beijing, China, April 2008. LMIB Beihang University.

[OTD10] Ayoub Otmani, Jean-Pierre Tillich, and Léonard Dallot. Cryptanalysis of two McEliece cryptosystems based on quasi-cyclic codes. *Special Issues of Mathematics in Computer Science*, 3(2) :129–140, January 2010.

[Ove06] Raphael Overbeck. Statistical decoding revisited. In Reihaneh Safavi-Naini Lynn Batten, editor, *Information security and privacy : 11th Australasian conference, ACISP 2006*, volume 4058 of *LNCS*, pages 283–294. Springer, 2006.

[Pea82] Judea Pearl. Reverend bayes on inference engines : A distributed hierarchical approach. In *Proceedings of the Second AAAI Conference on Artificial Intelligence*, AAAI'82, pages 133–136. AAAI Press, 1982.

[Pea88] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[PPWW72] William Wesley Peterson, Wesley Peterson, Edward J. Weldon, and Edward J. Weldon. *Error-correcting codes*. MIT press, 1972.

[Pra62] Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5) :5–9, 1962.

[RJ18] Dao Ren and Sha Jin. Improved Gradient Descent Bit Flipping Decoder for LDPC Codes on BSC Channel. *IEICE Electronics Express*, 15, April 2018.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2) :120–126, 1978.

[Sha48] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3) :379–423, 1948.

[Sho94] Peter W. Shor. Algorithms for quantum computation : Discrete logarithms and factoring. In S. Goldwasser, editor, *FOCS*, pages 124–134, 1994.

[Sid94] Vladimir Michilovich Sidelnikov. A public-key cryptosytem based on Reed-Muller codes. *Discrete Math. Appl.*, 4(3) :191–207, 1994.

[Sim94] Daniel R. Simon. On the power of quantum computation. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 116–123. IEEE Computer Society, 1994.

[SK14] Sujan Raj Shrestha and Young-Sik Kim. New McEliece cryptosystem based on polar codes as a candidate for post-quantum cryptography. In *2014 14th International Symposium on Communications and Information Technologies (ISCIT)*, pages 368–372. IEEE, 2014.

[SS92] Vladimir Michilovich Sidelnikov and S.O. Shestakov. On the insecurity of cryptosystems based on generalized Reed-Solomon codes. *Discrete Math. Appl.*, 1(4) :439–444, 1992.

[Ste88] Jacques Stern. A method for finding codewords of small weight. In G. D. Cohen and J. Wolfmann, editors, *Coding Theory and Applications*, volume 388 of *LNCS*, pages 106–113. Springer, 1988.

[SWB14] Gopalakrishnan Sundararajan, Chris Winstead, and Emmanuel Boutillon. Noisy Gradient Descent Bit-Flip Decoding for LDPC Codes. *IEEE Transactions on Communications*, 62, February 2014.

[Tan81] Robert Michael Tanner. A recursive approach to low complexity codes. *IEEE Trans. Inform. Theory*, 27(5) :533–547, 1981.

[Tix15] Audrey Tixier. *Reconnaissance de codes correcteurs. (Blind identification of error correcting codes)*. PhD thesis, Pierre and Marie Curie University, Paris, France, 2015. https://tel.archives-ouvertes.fr/tel-01238629.

[TV15] Ido Tal and Alexander Vardy. List decoding of polar codes. *IEEE Trans. Inform. Theory*, 61(5) :2213–2226, 2015.

[TWS15] Tasnuva Tithi, Chris Winstead, and Gopalakrishnan Sundararajan. Decoding LDPC codes via Noisy Gradient Descent Bit-Flipping with Re-Decoding. March 2015.

[Vit67] Andrew Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13(2) :260–269, 1967.

[Wie09] Christian Wieschebrink. Cryptanalysis of the Niederreiter public key scheme based on GRS subcodes. IACR Cryptology ePrint Archive, Report 2009/452, 2009.

[WNY+10] Tadashi Wadayama, Keisuke Nakamura, Masayuki Yagita, Yuuki Funahashi, Shogo Usami, and Ichi Takumi. Gradient Descent Bit Flipping Algorithms for Decoding LDPC Codes. *Communications, IEEE Transactions on*, 58 :1610–1614, July 2010.

[WYD08] Yige Wang, Jonathan Yedidia, and Stark Draper. Construction of high-girth QC-LDPC codes. pages 180–185, October 2008.

[WZX07] Xiaofu Wu, Chunming Zhao, and You Xiaohu. Parallel Weighted Bit-Flipping Decoding. *Communications Letters, IEEE*, 11 :671–673, September 2007.

[YEK+17] Atshudi Yamada, Edward Eaton, Kassem Kalach, Philip Lafrance, and Alex Parent. QC-MDPC KEM. First round submission to the NIST post-quantum cryptography call, November 2017.

[ZF04] Juntan Zhang and Marc P. C. Fossorier. A Modified Weighted Bit-Flipping Decoding of Low-Density Parity-Check Codes. *Communications Letters, IEEE*, 8 :165–167, April 2004.

[ZJW16] Bin Zhang, Lin Jiao, and Mingsheng Wang. Faster Algorithms for Solving LPN. In *Proceedings, Part I, of the 35th Annual International Conference on Advances in Cryptology — EUROCRYPT 2016 - Volume 9665*, pages 168–195, Berlin, Heidelberg, 2016. Springer-Verlag.

[ZYF14] Lina Zhang, Zhihui Ye, and Qi Feng. An Improved Multi-Bit Threshold Flipping LDPC Decoding Algorithm. *International Journal of Computer Theory and Engineering*, 6 :510–514, December 2014.

[ZZ05] Hao Zhong and Tong Zhang. Block-LDPC : a practical LDPC coding system design approach. *Circuits and Systems I : Regular Papers, IEEE Transactions on*, 52 :766–775, May 2005.