Recherche de Presque-Collisions pour le Décodage et la Reconnaissance de Codes Correcteurs

Kevin Carrier

Sorbonne Université

École doctorale Informatique, Télécommunications et Électronique (Paris)

Inria de Paris / Équipe-projet SECRET

Recherche de Presque-Collisions pour le Décodage et la Reconnaissance de Codes Correcteurs

Thèse de doctorat d'informatique

présentée par

Kévin CARRIER

dirigée par Nicolas Sendrier et Jean-Pierre Tillich

soutenue publiquement le 19 juin 2020

devant le jury composé de :

Jean-Pierre Tillich	Inria de Paris	Directeur
Nicolas Sendrier	Inria de Paris	Directeur
Pierre Loidreau	DGA-MI, Université de Rennes 1	Rapporteur
Gilles Zémor	Institut Mathématiques de Bordeaux	Rapporteur
Daniel Augot	Inria Saclay	Président
Annick Valibouze	LIP6, Sorbonne Université	Examinatrice
Sébastien Houcke	IMT Atlantique	Examinateur

Équipe-Projet
SECRET
Inria de Paris,
2 rue Simone IFF,
75012 Paris

Remerciements

Ces années de thèse n'auraient pas été les mêmes sans les personnes solides qui m'ont entouré, conseillé, soutenu et même supporté. Je vais tenter ici d'en dresser la liste en espérant que les personnes que je vais oublier me pardonneront cet affront.

Tout d'abord, je tiens à remercier Jean-Pierre Tillich qui m'a encadré tout au long de cette thèse. J'en sors aujourd'hui grandi, à la fois scientifiquement et humainement, et c'est sans nul doute grâce à toi. Tu as su guider mes réflexions et les résultats présentés ici n'auraient jamais vu le jour sans toi. Tu as aussi été là dans les moments de doute et tu as su me rassurer et m'encourager lorsque j'en avais le plus besoin. Jean-Pierre, je ne te remercierai jamais assez de t'être tant investi pour moi.

Je remercie également Nicolas Sendrier qui a également suivi cette thèse ; parfois d'un peu plus loin mais toujours avec beaucoup d'intérêt. Merci pour ton soutien. J'ai toujours apprécié nos échanges et regrette peut être aujourd'hui de ne pas t'avoir plus sollicité.

Je remercie aussi mes deux rapporteurs, Pierre Loidreau et Gilles Zémor, qui ont accepté la lourde tache de lire avec minutie ce manuscrit. Je remercie également Daniel Augot, Annick Valibouze et Sébastien Houcke d'avoir accepté de faire parti de mon jury.

Je souhaite également remercier Alain Couvreur et Binh-Minh Bui-Xuan d'avoir assuré le bon déroulement de ma vie de doctorant en ayant fait parti de mon comité de suivi doctoral.

J'ai eu l'immense chance d'effectuer mes années de thèse au sein de l'équipe-projet SECRET à l'Inria de Paris. J'ai passé, au sein de cette équipe, l'une des meilleures périodes de ma vie. L'effervescence intellectuelle dans laquelle j'ai baignée et les gens passionnés que j'y ai rencontrés ont été une réelle inspiration pour moi. Aujourd'hui, l'équipe a changé de nom et certains ont laissé la place à d'autres, mais je suis sûr que l'ambiance chaleureuse et rassurante qui a régné perdure encore.

J'aimerais remercier en particulier Anne Canteaut et Christelle Guiziou de m'avoir permis de participer aux conférences et autres évènements scientifiques. Je les remercie pour m'avoir aidé à ne pas me noyer dans les affres de l'administration.

Je remercie bien sûr tous les autres membres de l'équipe que j'ai eu la chance de côtoyer tout au long de mes années de thèse : Xavier Bonnetain, Christina Boura, Rémi Bricout, Rodolfo Canto-Torres, Daniel Coggia, André Chailloux, Julia Chaulet, Pascale Charpin, Sébastien Duval, Thomas Debris, Antoine Grospellier, Matthieu Lequesne, Gaëtan Leurant, Anthony Leverrier, Vivien Londe, Rocco Mora, Marìa Naya-Plasencia, Andrea Olivo, Léo Perrin, Yann Rotella, André Schrottenloher, Ferdinand Sibleyras, Valentin Vasseur.

Un merci tout particulier à Thomas, avec qui j'ai partagé le bureau mais pas que... Plus qu'une relation professionnelle, nous avons construit ensemble une belle amitié. Je ne pourrais me passer aujourd'hui de nos échanges sur des sujets aussi variés que les sciences, la politique, le foot, les amours. Tu m'as aussi fait découvrir un autre aspect de Paris. Je ne pense pas aimer un jour cette ville autant que toi, mais j'y tends dangereusement. On a partagé ensemble tellement de choses : des soirées plus loufoques les unes que les autres, un

road-trip au pays des koalas “rockeurs”, un nombre incommensurable de déménagements, des matchs de foot bière à la main (qui l'aurait cru ?...), des parties de babyfoot effrénées... Bref, mes années de thèse sans toi auraient surement été bien plus mornes !

Je souhaites également adresser un grand merci à Audrey. Il y a un peu plus de 3 ans, j'héritais de ton sujet de thèse. Je m'en suis un peu écarté avec le temps et j'espère que tu me le pardonneras. Audrey, tu m'as suivi et aidé dès le début de cette thèse et même avant. Tes remarques et tes relectures rigoureuses de mes différents travaux (notamment de ce manuscrit) ont à chaque fois permis de les sublimer. Je remercie également Jérôme, Maxime, Stanis, Marion et Florine de m'avoir suivi et soutenu tout au long de ces années. Merci de m'avoir laisser libre dans mes pérégrinations scientifiques.

La thèse n'est pas seulement un travail sur 3 ans ; c'est aussi l'aboutissement de longues années d'études qui n'auraient pas été aussi passionnantes, inspirantes et enrichissantes sans la rencontre de nombreux professeurs. Je n'aurais probablement jamais fait de thèse sans le soutien de certains d'entre eux. C'est pourquoi je voudrais remercier ici Sylvain Gervais, Xavier Saint-Raymond, Pascal Molin, Jean-François Mestre et enfin, Mireille Fouquet. Mireille, merci d'avoir cru en moi. Je suis arrivé à l'université Paris Diderot avec beaucoup de doutes et tu as su me redonner confiance en moi. Pascal, je te remercie de m'avoir donné l'opportunité d'enseigner. Tu savais à quel point cela me manquait et tu ne m'as pas oublié lorsqu'un poste s'est libéré. Je remercie aussi Françoise Levy-dit-Vehel, j'ai eu grand plaisir à dispenser les TDs de ton cours à l'ENSTA.

Je remercie également tous mes amis de Paris Diderot et du Master MIC : Mathias Ramparison, Jules Serra, Pierre-Léo Bégay, Maxence Guillemain d'Echon, Ismaïl Baaj, Kahina Kerchouche et toute la ‘team MIC”. Je ne me doutais pas en reprenant mes études que je ferais de si belles rencontres. Mathias, nous avons vécu nos années de thèse en parallèle. Merci de m'avoir écouté me plaindre de si nombreuses fois. Merci aussi de m'avoir permis d'être logé si près de l'Inria, cette aide est loin d'avoir été négligeable pour le bon déroulement de ma thèse.

Je tiens à remercier chaleureusement tous mes amis. Ils sont un soutien indéfectible en toute situation. Solène, Théotime, Samuel, Jérémie, Cécile, Morgane, Lise, Maxime, Sandrine, Mathieu, Caroline, Mikael, Jérémy, Gaylord, Valérian, Médéric merci à vous de m'avoir supporté pendant cette thèse. J'espère que vous me pardonnez tous les moments que je n'ai pas passés avec vous ces dernières années pour cause de “deadline” ou autres...

Solène, nos destins se suivent depuis la maternelle. Merci d'avoir simplement été là pendant ces années de thèse mais aussi avant. J'aime à penser que nous continuerons encore et toujours à partager toutes nos aventures de vie.

Valérian et Médéric, j'espère être un meilleur colocataire lorsque je ne suis pas en train de rédiger mon manuscrit. Merci à vous deux d'avoir redonner un souffle à ma vie parisienne.

Anthony, tu partages ma vie depuis la fin de cette thèse. Merci de ton soutien ainsi que de ton aide dans la relecture de certaines parties de ce manuscrit. Merci aussi de m'écouter parler de maths pendant des heures sans broncher. Nous ne nous attendions pas à vivre un confinement ensemble mais je n'aurais pas voulu le vivre autrement. J'espère que nous partagerons encore un long chemin ensemble.

J'aimerais terminer ces remerciements avec ceux qui comptent le plus à mes yeux : ma famille. Je remercie tout d'abord mes frères et sœurs, si nombreux sont-ils. Callyanna, Myriam, Joachim, Hermann, Florentina, Stan, Yohan, Sébastien, nos liens sont inaltérables. Votre soutien pendant cette thèse m'a été indispensable. Je remercie également mes oncles

et mes tantes, notamment Bernard et Alain qui ont toujours été là pour nous. En outre, merci à tous les deux pour vos conseils concernant la thèse et plus généralement ma carrière.

Pour finir, je remercie infiniment ma mère. Maman, merci d'avoir une confiance aveugle dans les choix que je fais. Je sais que tu es fière de moi et de ce que je suis devenu et rien ne saurait me rendre plus heureux. Pour toi, je ne cesserais jamais d'essayer d'être quelqu'un de meilleur. Merci aussi de m'écouter encore et toujours. Je ne t'en tiendrai pas rigueur si j'apprends demain que tu posais le téléphone sur le comptoir et me laissais déblatérer dans le vide pendant des heures à propos de mes illuminations mathématiques.

Table des matières

Liste des figures

Liste des tableaux

Liste des algorithmes

Notations

Ensembles, espaces vectoriels, espaces métriques

— $[\![a, b]\!]$ est l'ensemble des entiers compris entre a et b.

— $\{x_i\}_{i\in I}$ est l'ensemble des éléments x_i avec $i \in I$. Lorsque I est implicite, on note parfois $\{x_i\}_{i\in I} = \{x_i\}_i$.

— $\#E$ est le cardinal de l'ensemble E.

— $E \cup F$ est l'union des ensembles E et F.

— $E \cap F$ est l'intersection des ensembles E et F.

— $E\backslash F$ est l'ensemble E privé de l'ensemble F.

— $E \times F$ est l'ensemble des couples (x, y) tels que $x \in E$ et $y \in F$.

— $(\mathscr{E}, \Delta)$ est un espace métrique où Δ est la fonction de distance.

— Les distances relatives sont notées avec des lettres grecques. Par exemple, si w et d sont des distances, alors on note respectivement $\omega := \frac{w}{d_{\max}}$ et $\delta := \frac{d}{d_{\max}}$ où $d_{\max}$ est la distance maximale entre deux éléments de $\mathscr{E}$.

— $\mathbb{F}_2$ est le corps fini à deux éléments.

— $\mathbb{F}_q$ est le corps fini à q éléments.

— $\mathbb{F}_q^n$ est l'espace vectoriel de dimension n sur le corps $\mathbb{F}_q$.

— $\mathbb{F}_q^{m\times n}$ est l'ensemble des matrices à coefficients dans $\mathbb{F}_q$ et ayant m lignes et n colonnes.

— $\mathcal{S}_1^n := \{\mathbf{x} \in \mathbb{R}^n \ : \ \Delta(\mathbf{x}, \mathbf{c}) = r\}$ est la sphère unitaire sur l'espace $\mathbb{R}^n$ de dimension n muni de la distance euclidienne Δ.

— $\mathcal{B}_{\mathscr{E}}(\mathbf{c}, r) := \{\mathbf{x} \in \mathscr{E} \ : \ \Delta(\mathbf{x}, \mathbf{c}) \leqslant r\}$ est la boule fermée de centre $\mathbf{c}$ et de rayon r sur l'espace métrique $\mathscr{E}$. Lorsque $\mathscr{E}$ est implicite, on note cette boule $\mathcal{B}(\mathbf{c}, r)$.

— $\overline{\mathcal{B}}_{\mathscr{E}}(\mathbf{c}, r) := \{\mathbf{x} \in \mathscr{E} \ : \ \Delta(\mathbf{x}, \mathbf{c}) \geqslant r\}$ est la fermeture de $\mathscr{E}\backslash\mathcal{B}(\mathbf{c}, r)$. Lorsque $\mathscr{E}$ est implicite, on note cet ensemble $\overline{\mathcal{B}}(\mathbf{c}, r)$.

— $\mathcal{S}_{\mathscr{E}}(\mathbf{c}, r) := \{\mathbf{x} \in \mathscr{E} \ : \ \Delta(\mathbf{x}, \mathbf{c}) = r\}$ est la sphère de centre $\mathbf{c}$ et de rayon r sur l'espace métrique $\mathscr{E}$. Lorsque $\mathscr{E}$ est implicite, on note cette sphère $\mathcal{S}(\mathbf{c}, r)$.

Vecteurs, matrices

— Les vecteurs sont notés avec des lettres grasses minuscules ; par exemple, $\mathbf{x}$, $\mathbf{y}$, $\mathbf{c}$, $\mathbf{e}$...

— Les matrices sont notés avec des lettres grasses majuscules ; par exemple, $\mathbf{M}$, $\mathbf{H}$, $\mathbf{G}$...

— $\mathbf{0}$ dénote le vecteur nul ou la matrice nulle. Si la taille n'est pas implicite, alors nous l'indiquons en indice.

— $\mathbf{Id}_n$ est la matrice identité de taille $n \times n$. Lorsque la taille n est implicite, nous écrivons simplement $\mathbf{Id}$.

Soit $\mathbf{x}, \mathbf{y} \in \mathbb{F}_q^n$ et soit $\mathbf{M} \in \mathbb{F}_q^{m \times n}$:

— Pour tout $i \in [\![1, n]\!]$, x_i est la $i^{\text{ème}}$ composante du vecteur $\mathbf{x}$. On a donc $\mathbf{x} := (x_1, x_2, \cdots, x_n)$.

— $(\mathbf{x}, \mathbf{y})$ est la concaténation des vecteurs $\mathbf{x}$ et $\mathbf{y}$.

— Si $I := \{i_1, i_2, \cdots, i_s\} \subseteq [\![1, n]\!]$ est une liste ordonnée d'indices, alors $\mathbf{x}_I := (x_{i_1}, x_{i_2}, \cdots, x_{i_s})$.

— $\mathbf{x}^\top$ est la transposée du vecteur $\mathbf{x}$.

— $\Delta(\mathbf{x}, \mathbf{y}) := \#\{i \in [\![1, n]\!] \ : \ x_i \neq y_i\}$ est la distance de Hamming entre les vecteurs $\mathbf{x}$ et $\mathbf{y}$.

— $|\mathbf{x}| := \#\{i \in [\![1, n]\!] \ : \ x_i \neq 0\}$ est le poids de Hamming de $\mathbf{x}$.

— $\mathbf{x} + \mathbf{y}$ est la somme composante par composante sur $\mathbb{F}_q$. Notamment, si $q = 2$, c'est le *ou exclusif* (XOR) des deux vecteurs.

— $< \mathbf{x}, \mathbf{y} > := \sum_{i=1}^{n} x_i y_i$ dénote le produit scalaire sur $\mathbb{F}_q^n$.

— $\text{supp}(\mathbf{x}) := \#\{i \in [\![1, n]\!] \ : \ x_i \neq 0\}$ est le support de $\mathbf{x}$.

— Pour tout $(i, j) \in [\![1, n]\!] \times [\![1, m]\!]$, $\mathbf{M}_{i,j}$ est la composante de coordonnée (i, j) de la matrice $\mathbf{M}$; c'est-à-dire le coefficient de la $i^{\text{ème}}$ ligne et de la $j^{\text{ème}}$ colonne.

— Pour tout $i \in [\![1, n]\!]$, $\mathbf{M}_{i,*}$ est la $i^{\text{ème}}$ ligne de la matrice $\mathbf{M}$.

— Pour tout $j \in [\![1, m]\!]$, $\mathbf{M}_{*,j}$ est la $j^{\text{ème}}$ colonne de la matrice $\mathbf{M}$.

— Soit $\mathbf{M}' \in \mathbb{F}_q^{m \times n'}$. La matrice $[\mathbf{M}|\mathbf{M}']$ est la concaténation horizontale des matrices $\mathbf{M}$ et $\mathbf{M}'$.

— Soit $\mathbf{M}' \in \mathbb{F}_q^{m' \times n}$. La matrice $\left[\frac{\mathbf{M}}{\mathbf{M}'}\right]$ est la concaténation verticale des matrices $\mathbf{M}$ et $\mathbf{M}'$.

— Si $I := \{i_1, i_2, \cdots, i_s\} \subseteq [\![1, n]\!]$ est une liste ordonnée d'indices, alors $\mathbf{M}_I := \left[\mathbf{M}_{*,i_1}|\mathbf{M}_{*,i_2}|\cdots|\mathbf{M}_{*,i_s}\right]$ est la matrice formée des colonnes de $\mathbf{M}$ indexées par I.

— $\mathbf{M}^\top$ est la transposée de la matrice $\mathbf{M}$.

— $\text{rang}(\mathbf{M})$ est le rang de la matrice $\mathbf{M}$.

Theorie de l'information, codes correcteurs d'erreurs

— $\begin{array}{rccc} h_q : & [0,1] & \to & [0,1] \\ & p & \mapsto & p\log_q(q-1) - p\log_q(p) - (1-p)\log_q(1-p) \end{array}$ est la fonction entropie q-aire de Shannon.

— g_q^- ou h_q^{-1} est la fonction réciproque de h_q^- où h_q^- est la fonction h_q restreinte à $\left[0, 1 - \frac{1}{q}\right]$.

— g_q^+ est la fonction réciproque de h_q^+ où h_q^+ est la fonction h_q restreinte à $\left[1 - \frac{1}{q}, 1\right]$.

— $\text{BSC}(p)[1]$ ou $\text{BSC}(p)$ est le canal binaire symétrique de probabilité d'erreur p.

— Si $\sum_{i=1}^{s} r_i = 1$, alors $\{\text{BSC}(p_i)[r_i]\}_{i \in [\![1,s]\!]}$ est un canal de transmission. Un bit qui transite via ce canal a une probabilité r_i de passer par le canal $\text{BSC}(p_i)$

— $\mathcal{C}$ est un $[n, k]_q$ code linéaire signifie que $\mathcal{C}$ est un code linéaire de longueur n et de dimension k défini sur $\mathbb{F}_q$. Autrement dit, c'est un sous-espace vectoriel de $\mathbb{F}_q^n$ de dimension k. Lorsque la valeur de q est implicite, on dit plus simplement que $\mathcal{C}$ est un $[n, k]$ code linéaire.

— $\mathbf{G} \in \mathbb{F}_q^{k \times n}$ est une matrice génératrice de $\mathcal{C}[n, k]_q$ si et seulement si les lignes de $\mathbf{G}$ forment une base de $\mathcal{C}$:

$$\mathcal{C} = \left\{\mathbf{x}\mathbf{G} \in \mathbb{F}_q^n \ : \ \mathbf{x} \in \mathbb{F}_q^k\right\}$$

— $\mathcal{C}^{\perp} := \left\{\mathbf{h} \in \mathbb{F}_q^n \ : \ \forall \mathbf{c} \in \mathcal{C}, < \mathbf{c}, \mathbf{h} >= 0\right\}$ est le code dual de $\mathcal{C}$. Les éléments de $\mathcal{C}^{\perp}$ sont appelés équations de parité de $\mathcal{C}$.

— $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$ est une matrice de parité de $\mathcal{C}[n,k]_q$ si et seulement si les lignes de $\mathbf{H}$ forment une base de $\mathcal{C}^{\perp}$:

$$\mathcal{C}^{\perp} = \left\{\mathbf{x}\mathbf{H} \in \mathbb{F}_q^n \ : \ \mathbf{x} \in \mathbb{F}_q^{n-k}\right\}$$

ou bien :

$$\mathcal{C} = \left\{\mathbf{c} \in \mathbb{F}_q^n \ : \ \mathbf{H}\mathbf{c}^{\top} = \mathbf{0}\right\}$$

— $d_{\mathcal{C}}$ dénote la distance minimale du code $\mathcal{C}$.

— Soit un espace métrique $(\mathscr{E}, \Delta)$ probabilisé avec la probabilité uniforme. On définit la distance de Gilbert-Varshamov *inférieure* associée aux codes (linéaires ou non) de taille S sur $\mathscr{E}$ par :

$$d_{\mathrm{GV}}^{-}(S) := \begin{cases} \sup\{d \ : \ S \cdot \mathbb{P}\left(\Delta\left(\mathbf{u}, \mathbf{x}\right) \leqslant d\right) \leqslant 1\} & \text{si } \mathbb{P}\left(\Delta\left(\mathbf{u}, \mathbf{x}\right) \leqslant 0\right) > \frac{1}{S} \\ 0 & \text{sinon} \end{cases}$$

où $\mathbf{u}$ est tiré uniformément dans $\mathscr{E}$ et $\mathbf{x} \in \mathscr{E}$ (nous supposons que la probabilité $\mathbb{P}\left(\Delta\left(\mathbf{u}, \mathbf{x}\right) \leqslant d\right)$ ne dépend pas de $\mathbf{x}$).
Si $\mathscr{E} = \mathbb{F}_q^n$, alors pour un rendement $R = \frac{k}{n}$ constant et lorsque n tend vers l'infini, on a :

$$d_{\mathrm{GV}}^{-}(q^k) := d_{\mathrm{GV}}(n,k) := d_{\mathrm{GV}}^{-}(n,k) = nh_q^{-1}\left(1-R\right) = ng_q^{-}(1-R)(1+o(1))$$

— On définit de façon analogue la distance de Gilbert-Varshamov *supérieure*. Soit $(\mathscr{E}, \Delta)$ un espace métrique probabilisé avec la probabilité uniforme.

$$d_{\mathrm{GV}}^{+}(S) := \begin{cases} \inf\{d \ : \ S \cdot \mathbb{P}\left(\Delta\left(\mathbf{u}, \mathbf{x}\right) \geqslant d\right) \leqslant 1\} & \text{si } \mathbb{P}\left(\Delta\left(\mathbf{u}, \mathbf{x}\right) \geqslant d_{\max}\right) > \frac{1}{S} \\ d_{\max} & \text{sinon} \end{cases}$$

où $d_{\max}$ est la distance maximale entre deux éléments de $\mathscr{E}$, $\mathbf{u}$ est tiré uniformément dans $\mathscr{E}$ et $\mathbf{x} \in \mathscr{E}$ (nous supposons que la probabilité $\mathbb{P}\left(\Delta\left(\mathbf{u}, \mathbf{x}\right) \geqslant d\right)$ ne dépend pas de $\mathbf{x}$).
Si $\mathscr{E} = \mathbb{F}_q^n$, alors pour un rendement $R = \frac{k}{n}$ constant et lorsque n tend vers l'infini, on a :

$$d_{\mathrm{GV}}^{+}(q^k) := d_{\mathrm{GV}}^{+}(n,k) = ng_q^{+}(1-R)(1+o(1))$$

Comparaisons asymptotiques, notations de Landau

Soient f et g des fonctions de $\mathbb{N}$ dans $\mathbb{R}$:

— $f(n) \in o(g(n))$ signifie que f est négligeable devant g ; autrement dit, $\lim\limits_{n\to+\infty} \dfrac{f(n)}{g(n)} = 0$. Par abus de notation, nous écrivons généralement $f(n) = o(g(n))$.

— $f(n) \in O(g(n))$ signifie que $|f(n)|$ est majorée par $|g(n)|$ à un facteur constant près ; autrement dit, $\exists M > 0, \forall n \in \mathbb{N}, |f(n)| \leqslant M \cdot |g(n)|$. Par abus de notation, nous écrivons généralement $f(n) = O(g(n))$.

— $f(n) \in \tilde{O}(g(n))$ signifie que $\exists k \in \mathbb{N}, f(n) = \log\left(|g(n)|\right)^k \cdot O(g(n))$. Par abus de notation, nous écrivons généralement $f(n) = \tilde{O}(g(n))$.

— $f(n) \in \Omega\left(g(n)\right)$ signifie que $|f(n)|$ est minorée par $|g(n)|$ à un facteur constant près ; autrement dit, $\exists m > 0, \forall n \in \mathbb{N}, |f(n)| \geqslant m \cdot |g(n)|$. Par abus de notation, nous écrivons généralement $f(n) = \Omega\left(g(n)\right)$.

— $f(n) \in \tilde{\Omega}(g(n))$ signifie que $\exists k \in \mathbb{N}, f(n) = \log(|g(n)|)^k \cdot \Omega(g(n))$. Par abus de notation, nous écrivons généralement $f(n) = \tilde{\Omega}(g(n))$.

— $f(n) \in \Theta(g(n))$ signifie que $f(n) \in O(g(n))$ et $f(n) \in \Omega(g(n))$. Par abus de notation, nous écrivons généralement $f(n) = \Theta(g(n))$.

— $f(n) \in \tilde{\Theta}(g(n))$ signifie que $\exists k \in \mathbb{N}, f(n) = \log(|g(n)|)^k \cdot \Theta(g(n))$. Par abus de notation, nous écrivons généralement $f(n) = \tilde{\Theta}(g(n))$.

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 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.

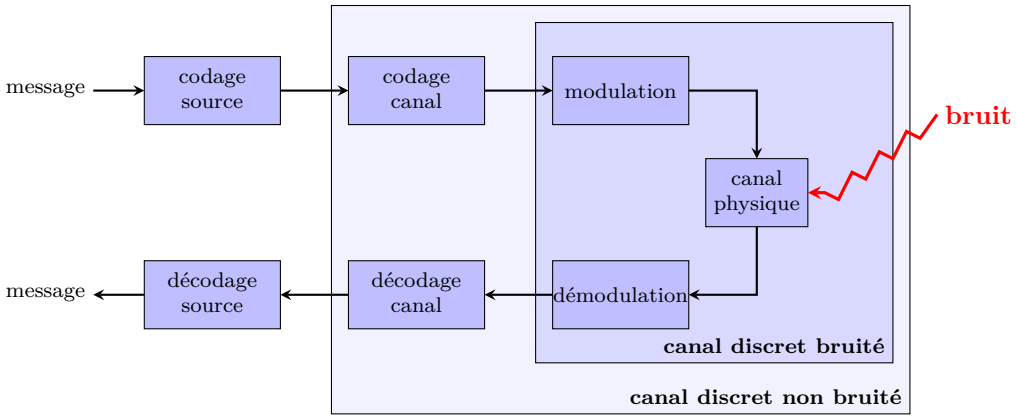


Figure 1 – Schéma de transmission numérique.

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 canal 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 qu'une opération cryptographique. Cette dernière a pour but de protéger l'information en garantissant par exemple :

— la confidentialité, qui assure que l'information n'est pas accessible à n'importe qui ;
— l'authentification, qui certifie qu'une personne est l'auteur d'un message ;
— la non-répudiation, qui empêche la remise en cause d'un contrat ;
— l'intégrité, qui assure que l'information n'a pas été manipulée/altérée volontairement ou non.

Au codage source, succède le codage canal. Le rôle 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.

Avant de transiter sur le canal physique, le message codé est modulé ; c'est-à-dire que la suite binaire que forme ce message est transformée en un signal 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. Toutefois, dans un milieu non coopératif, les paramètres de modulation, de codage canal et de codage source ne sont pas nécessairement connus ; il s'agit alors de les retrouver.

Dans ce manuscrit, nous nous intéressons aux codes correcteurs d'erreurs à travers deux problématiques : la reconnaissance d'un codage canal dans un cadre non coopératif et la mesure de sécurité de systèmes cryptographiques utilisant des codes correcteurs d'erreurs.

La reconnaissance de codes correcteurs d'erreurs

Un premier problème qui a motivé cette thèse est la reconstruction de codes correcteurs d'erreurs. Ce problème se pose lorsque que nous voulons retrouver un code inconnu à partir de la seule connaissance de mots de code bruités ; le but étant de corriger ces mots mais aussi d'en corriger d'autres qui seront potentiellement émis ultérieurement. Si nous nous référons au schéma de transmission numérique de la figure 1, nous nous plaçons en sortie du démodulateur. Nous supposons donc que la démodulation a été correctement effectuée. Dans le cas où nous ne connaissons pas non plus les paramètres de modulation, il suffit généralement de tester exhaustivement toutes les techniques de démodulation connues jusqu'à trouver une information corrélée.

En pratique, les codes rencontrés sont toujours des codes linéaires définis sur un corps fini ; c'est-à-dire qu'ils ont une structure d'espace vectoriel sur $\mathbb{F}_q$ où q est la taille du corps fini. Un code linéaire de longueur n et de dimension k sur $\mathbb{F}_q$ est donc un sous-espace vectoriel de dimension k de $\mathbb{F}_q^n$. Reconnaître de tels codes a été montré NP-complet par Valembois en 2001 [Val01], même sous l'hypothèse que la longueur et la dimension du code sont connues. Il sera donc nécessaire d'émettre des hypothèses supplémentaires, notamment sur la famille à laquelle appartient le code recherché. Pour de nombreuses familles de codes possédant une structure mathématique forte telle que les codes BCH, les codes de Reed-Solomon ou les codes de Reed-Muller, le problème de reconnaissance est essentiellement résolu. Cependant, pour d'autres familles, les solutions existantes sont insatisfaisantes.

Nous définissons les équations de parité d'un code linéaire comme étant les mots du code dual (c'est-à-dire les vecteurs de l'espace vectoriel dual). D'autre part, le nombre de

positions non-nulles d'un vecteur est appelé poids de Hamming. Dans ce manuscrit, nous nous concentrons particulièrement sur la reconstruction de codes possédant des équations de parité de poids faible (borné par rapport à n). Parmi eux, nous avons les codes LDPC (*Low-Density Parity-Check*) mais aussi les codes convolutifs, les turbos-codes, les codes raptors, les codes fontaines ou encore les codes polaires.

Dans nos travaux, nous avons développé une nouvelle technique de recherche d'équations de parité de petit poids dans un code. Celle-ci interpole et améliore les solutions existantes. Notre méthode exploite essentiellement deux idées. La première consiste à limiter la propagation des erreurs dans les mots bruités en réduisant le nombre d'opérations que nous effectuons sur ceux-ci. Pour cela, nous utilisons une élimination gaussienne partielle ou même l'algorithme BKW [BKW03]. Ce dernier permet de produire un grand nombre de zéros dans une matrice avec un nombre restreint d'opérations. La seconde idée est l'utilisation de nouvelles techniques pour résoudre le problème de recherche de presque-collisions. Ce problème consiste à rechercher tous les couples proches dans une liste d'éléments d'un espace métrique. Nous parlerons plus en détail de nos travaux sur ce problème un peu plus loin dans cette introduction.

En outre, pour analyser notre algorithme de reconnaissance de codes, nous avons construit un outil permettant de dénombrer les équations de parité de poids donné dans un code. Pour n tendant vers l'infini, nous donnons également une expression asymptotique de nos énumérateurs de poids.

Finalement, tous ces travaux ont été présentés lors du 10$^{\text{ème}}$ *International Workshop on Coding and Cryptography* en 2017 à St Petersbourg [CT17]. Une version longue de ce papier a été publiée dans le journal *Designs, Codes and Cryptography* en 2019 [CT19b].

Le décodage générique

Le problème de reconnaissance de codes LDPC s'avère être très proche de celui du décodage générique. Il consiste à effectuer une décodage avec la seule connaissance d'une base de l'espace vectoriel que forme le code. Dans la littérature, nous traitons généralement le problème dual équivalent : étant donnés une matrice $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$, un vecteur $\mathbf{s} \in \mathbb{F}_q^{n-k}$ et un entier $w \in [\![0, n]\!]$, trouver un vecteur $\mathbf{e} \in \mathbb{F}_q^n$ qui contient au plus w positions non nulles et tel que $\mathbf{He}^\top = \mathbf{s}$. Ce problème est aussi appelé décodage par syndrome. La sécurité de nombreux crypto-systèmes repose sur la difficulté de ce problème ; on parle alors de cryptographie fondée sur les codes correcteurs. Un des avantages de cette cryptographie est qu'elle produit de potentiels candidats pour protéger les systèmes d'information contre la menace des ordinateurs quantiques.

Depuis les années 60, de nombreuses méthodes ont été proposées pour résoudre le décodage générique. Dans les régimes difficiles, il demande un temps de calcul exponentiel en la longueur du code. Les différents algorithmes proposés depuis celui de Prange en 1962 [Pra62] n'ont pas changé la nature de la difficulté du problème. Toutefois, des avancées majeures ont permis de réduire significativement l'exposant de la complexité asymptotique. Parmi elles, nous pouvons citer les méthodes de Stern [Ste88] et de Dumer [Dum91] qui utilisent la recherche de collisions ou encore les méthodes MMT [MMT11] et BJMM [BJMM12] qui utilisent la technique des représentations de Howgrave-Graham et Joux [HJ10]. Les dernières avancées dans le domaine du décodage générique font toutes appel à des recherches de presque-collisions [MO15, BM17b, BM18].

Nous avons également contribué à améliorer le décodage générique. Dans le cas des espaces binaires de Hamming, nous avons réutilisé l'algorithme de Both et May [BM18] mais en remplaçant les étapes de recherche de presque-collisions par nos propres méthodes pour résoudre ce problème. Actuellement, la complexité record pour décoder génériquement un code binaire linéaire de longueur n dans le pire régime – c'est-à-dire pour la distance

de décodage la plus difficile à atteindre et pour le pire rendement – est $2^{0.08845n(1+o(1))}$ où $2^{n\,o(1)}$ est sous-exponentiel en n mais super-polynomial. Nos travaux nous ont permis de diminuer l'exposant asymptotique de cette complexité : notre méthode permet en effet de résoudre la même instance du problème de décodage générique en un temps de l'ordre de $2^{0.08821n(1+o(1))}$. De plus, le surcoût $2^{n\,o(1)}$ est ici inférieur à celui de Both et May ; nous soupçonnons même qu'il soit polynomial.

Pour finir, nous avons généralisé l'algorithme Both-May aux espaces non-binaires. En instanciant cet algorithme avec nos méthodes de recherche de presque-collisions sur $\mathbb{F}_q$, nous avons ainsi pu étendre nos améliorations sur le décodage générique à n'importe quels corps finis.

La recherche de presque-collisions

Par deux fois dans cette introduction, nous avons évoqué le problème de recherche de presque-collisions. Nous l'énonçons plus formellement comme suit : étant données deux listes $\mathscr{L}_1$ et $\mathscr{L}_2$ (éventuellement égales) d'éléments d'un espace métrique et une distance w, trouver les couples de $\mathscr{L}_1 \times \mathscr{L}_2$ à distance au plus w. Formulé ainsi, ce problème est très générique : nous ne précisons ni l'espace métrique, ni la façon dont les listes ont été générées. De plus, il en existe des variantes. Par exemple, l'une des listes peut être donnée sous forme de flux ; il s'agit alors de trouver dans l'autre liste les éléments proches de la requête courante fournie par ce flux. D'autres variantes consistent à rechercher non pas des couples proches mais *le* couple *le plus* proche.

Pour traiter ces problèmes de proximité, l'approche la plus connue est celle consistant à utiliser des fonctions de hachage localement sensibles ; c'est-à-dire des fonctions qui associent des mots proches à une même clé de hache avec une bonne probabilité [IM98]. De nombreuses solutions ont été proposées pour construire de telles fonctions. Certaines d'entre elles utilisent des décodages de codes correcteurs [GMO10, Dub10] : ce sont ces méthodes qui retiendront notre attention.

Dans le cadre du décodage générique dans les espaces binaires, la méthode de May et Ozerov [MO15] pour la recherche de presque-collision a été une avancée importante. Elle a été adaptée aux espaces non-binaires par Hirose dans [Hir16].

En généralisant le problème de presque-collisions, nous touchons de nombreux domaines d'informatique qui vont au delà du problème de reconnaissance de codes ou de celui du décodage générique. Il permet notamment de traiter des problèmes majeurs de la cryptographie fondée sur les réseaux euclidiens. Cette dernière est une autre alternative envisageable pour la cryptographie résistante aux ordinateurs quantiques.

Enfin, un problème analogue à la recherche de presque-collisions est celui qui consiste à rechercher des lointaines-collisions ; c'est-à-dire des couples d'éléments non pas proches mais le plus éloignés possible. Cette recherche peut être utilisée pour effectuer des décodages en gros poids. De tels décodages ont pour but de trouver le mot de code le plus éloigné d'un mot donné de l'espace ambiant. Ce problème a récemment trouvé une application dans une des premières signatures en cryptologie fondée sur les codes : WAVE [DST19].

Une grande partie de ce manuscrit est consacrée à la recherche de presque-collisions. Nous proposons notamment de construire des fonctions de hachage floues à l'aide de codes correcteurs d'erreurs. Notre méthode est optimale avec un décodage en liste idéal de codes aléatoires qui retourne tous les mots de code inclus dans une boule centrée sur le mot à décoder en un temps de l'ordre de la taille de la liste retournée. Ne connaissant pas de tels algorithmes de décodage, nous proposons de remplacer les codes aléatoires par des codes polaires dont les performances sont très proches. Nous avons alors analysé le décodage en liste de Tal et Vardy de ces codes [TV15] dans le cadre de la recherche de presque-collisions. Nous avons ainsi conclu que celui-ci permet d'atteindre la complexité optimale de notre méthode avec un facteur sous-exponentiel que nous soupçonnons polynomial.

Au cours de nos travaux, nous avons étudié en profondeur les codes polaires et avons proposé une nouvelle façon de les construire. Nous avons également proposé différentes pistes pour éventuellement améliorer leurs performances.

Enfin, nous avons étendu nos travaux à la recherche de lointaines-collisions. Dans ce nouveau problème, nous ne recherchons pas des couples d'éléments proches mais éloignés. Nous avons adapté notre méthode de recherche de presque-collisions à ce problème en introduisant la notion d'anti-décodeur en liste qui retourne tous les mots de code à une distance supérieure à un certain rayon de décodage du mot à décoder.

Organisation du manuscrit

Ce manuscrit s'organise en neuf chapitres. Nous résumons les thèmes abordés dans chacun d'eux :

ch. 1. Dans le premier chapitre, nous commençons par énoncer quelques outils de la théorie de l'information de Shannon qui nous serons indispensables tout au long de ce manuscrit. Nous introduisons aussi la notion de code correcteur d'erreurs et présentons quelques généralités à leur sujet. Nous détaillons ensuite deux grandes familles de codes : les codes LDPC que nous chercherons à reconnaître et les codes polaires que nous généralisons aux codes $U|U+V$ récursifs.

ch. 2. Dans le chapitre 2, nous abordons le problème du décodage générique. Nous commençons par lui donner un contexte en parlant de cryptographie dite post-quantique et de la nécessité d'inventer de nouveaux paradigmes de sécurité autres que ceux fondés sur la théorie des nombres et qui sont aujourd'hui menacés par les ordinateurs quantiques. Nous introduisons alors la cryptographie fondée sur les codes en présentant le chiffrement de McEliece qui est l'un des premiers à reposer sa sécurité sur le problème du décodage générique. Nous décrivons aussi d'autres crypto-systèmes plus récents et utilisant également des codes correcteurs.

Après avoir contextualisé le problème du décodage générique, nous présentons une série de solutions pour le résoudre. Dans la littérature, les algorithmes que nous décrivons sont généralement proposés pour le décodage de codes binaires. Dans ce chapitre, nous généralisons ces méthodes aux cas non-binaires. Pour presque toutes les méthodes que nous décrivons, cette généralisation existe déjà, sauf pour la méthode de Both et May qui n'existe que dans une version binaire.

L'algorithme de Both et May est le plus récent algorithme de décodage générique. Il permet d'obtenir les meilleures complexités asymptotiques. Il repose essentiellement sur des recherches de presque-collisions qui seront l'objet de chapitres ultérieurs.

ch. 3. Dans le chapitre 3, nous traitons le problème de reconnaissance de code. Dans un premier temps nous présentons les travaux de Tixier et Tillich sur la reconstruction de codes LDPC. Ceux-ci s'inspirent notamment de la méthode de décodage générique de Dumer. Dans un second temps, nous présentons deux méthodes relativement triviales pour résoudre ce problème. Nous nous inspirons ensuite de celles-ci pour présenter un nouvel algorithme qui interpole les solutions existantes. Ce travail a été publié dans [CT17, CT19b].

Enfin, dans la dernière partie de ce chapitre, nous montrons comment les techniques de résolution d'un autre problème connu – à savoir le problème LPN (*Learnig from Parity with Noise*) – peuvent aussi être utilisées pour la reconnaissance de codes.

ch. 4. Le chapitre 4 est une extension du chapitre 3. Il présente un outil permettant de compter le nombre d'équations de parité de poids donné dans un code LDPC (le

poids d'un mot est le nombre de positions non nulles). Nous proposons une expression asymptotique de cet énumérateur de poids qui nous permettra d'optimiser efficacement nos méthodes de reconnaissance de code. Ce travail est aussi présenté dans [CT17, CT19b].

ch. 5. Dans le chapitre 5, nous introduisons différents problèmes de proximité dont le problème de recherche de presque-collisions. Ce chapitre est essentiellement un état de l'art du sujet. Nous y décrivons notamment différentes méthodes pour résoudre le problème des presque-collisions. En particulier, nous présentons l'approche LSH (*Locality Sensitive Hashing*) qui utilise la notion de fonction de hachage floue.

Dans ce chapitre, nous présentons aussi la méthode de May et Ozerov qui est à ce jour la méthode la plus efficace pour trouver des presque-collisions, du moins dans le modèle afférant au décodage générique.

Enfin, nous terminons ce chapitre en détaillant la méthode de Gordon, Miller et Ostapenko qui utilise des codes pour construire des fonctions de hachage floues. Leur méthode est sensiblement la même que celle de Dubiner dans [Dub10] mais leur analyse diffère.

ch. 6-7. Dans les chapitres 6 et 7, nous présentons une nouvelle méthode de résolution du problème de recherche de presque ou lointaines collisions. Notre méthode s'appuie sur l'utilisation de décodage en liste de codes correcteurs.

Dans le chapitre 6, nous analysons notre méthode de façon la plus générique possible. Dans le chapitre 7, nous l'appliquons à différents espaces métriques tels que la sphère euclidienne ou les espaces de Hamming binaires et non-binaires.

ch. 8. Dans le chapitre 8, nous proposons différentes modifications de notre algorithme de recherche de presque-collisions décrit dans les chapitres 6 et 7. L'une d'entre elles permet d'améliorer significativement notre approche initiale.

ch. 9. Dans le chapitre 9, nous étudions l'utilisation de familles de codes particuliers dans notre méthode de recherche de presque-collisions. Nous proposons d'abord d'utiliser le produit cartésien de codes aléatoires. Finalement, nous adoptons plutôt les codes polaires qui sont plus difficiles à analyser mais qui nous permettent d'obtenir des résultats pratiques plus probants.

Chapitre 1

Quelques éléments de théorie de l'information

Dans ce chapitre, nous commençons par introduire des outils de théorie de l'information de Shannon que nous utiliserons tout au long de ce document. Nous introduisons aussi brièvement les codes correcteurs d'erreurs qui sont au cœur du sujet de cette thèse. Enfin, dans les sections 1.3 à 1.4, nous détaillons un peu plus certaines familles de codes :

— les codes LDPC qui seront au centre du chapitre sur la reconnaissance de codes ;

— les codes $U|U+V$ récursifs qui sont une généralisation des codes polaires.

Le travail sur les codes $U|U+V$ récursifs va au delà d'un simple état de l'art et fait parti des contributions de cette thèse.

1.1 Les canaux de communication et les modèles d'erreur

Au départ, 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.

1.1.1 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 $x\in\mathscr{A}$:

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

1.1.2 Les canaux discrets sans mémoire

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

Définition 1.1.1 (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 1.1.2 (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 :*

$$\begin{aligned} C_{CDSM}(\mathscr{A},\mathscr{B},\mathbf{\Pi}) &:= \max_{\mathscr{P}}\Big(H_q(Y)-H_q(Y|X)\Big) \quad symb_q/\Delta t && (1.4)\\ &= \log_2(q)\cdot\max_{\mathscr{P}}\Big(H_q(Y)-H_q(Y|X)\Big) \quad bits/\Delta t && (1.5)\\ &= \max_{\mathscr{P}}\Big(H_2(Y)-H_2(Y|X)\Big) \quad bits/\Delta t && (1.6)\end{aligned}$$

où :

– *$\mathscr{P}$ est l'ensemble des distributions de probabilité d'émission possibles :*

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

– *X est la variable aléatoire à valeurs dans $\mathscr{A}$ modélisant l'entrée du canal ;*
– *Y est la variable aléatoire à valeurs dans $\mathscr{B}$ modélisant la sortie du canal.*

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.

1.1.3 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 $(i,j) \in [\![1,q]\!]^2$ tel que $i \neq j$, tous les $\mathbf{\Pi}_{i,j}$ 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 1.1.3 (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 1.1.4.

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

Démonstration du théorème 1.1.4.

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

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

À titre d'exemple, la figure 1.1 donne une représentation schématisée du canal $\mathrm{SC}_3(p)$ lorsque la distribution de probabilité d'émission est uniforme :

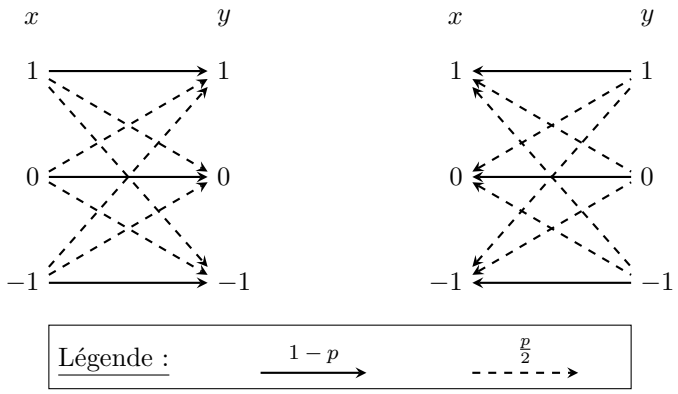


Figure 1.1 – Canal symétrique de probabilité d'erreur p sur un alphabet à 3 symboles.

Remarque 1.1.1. 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.

1.1.4 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 $\mathrm{BSC}(p)$ 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 1.1.5 :

Corollaire 1.1.6. *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 1.2 donne une représentation schématisée du canal $\mathrm{BSC}(p)$ lorsque la distribution de probabilité d'émission est uniforme :

Figure 1.2 – Canal binaire symétrique de probabilité d'erreur p.

1.1.5 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. 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 1.1.7. *La capacité du canal $EC_q(p)$ est :*

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

La figure 1.3 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 $\mathrm{BEC}(p)$.

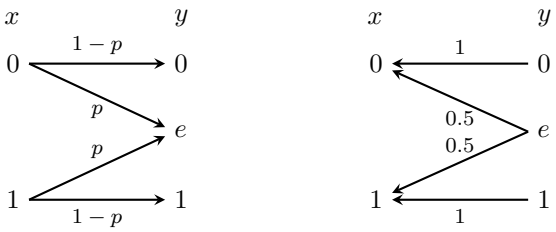


Figure 1.3 – Canal binaire à effacement de probabilité d'effacement p.

1.1.6 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.

Les formes d'erreur décrites jusqu'ici modélisent des situations qui peuvent se produire en pratique. Nous allons nous intéresser à présent 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, il est recherché un vecteur d'erreur de poids de Hamming exactement w. 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 loin.

1.2 Généralités sur les codes correcteurs d'erreurs

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.

1.2.1 Les codes en bloc linéaires

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 1.2.1. *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 $\Delta(\mathbf{x}, \mathbf{y})$:*

$$\Delta(\mathbf{x}, \mathbf{y}) := \#\{i \in [\![1, n]\!] : x_i \neq y_i\} \tag{1.17}$$

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 $\mathbf{x}$:

$$|\mathbf{x}| := \#\{i \in [\![1, n]\!] : x_i \neq 0\} = \Delta(\mathbf{x}, \mathbf{0}) \tag{1.18}$$

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.

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

Définition 1.2.2 (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 1.2.3 (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 $\mathbf{H}\mathbf{c}^\top = \mathbf{0}$.*

L'espace vectoriel orthogonal de $\mathcal{C}$ généré par la matrice $\mathbf{H}$ est appelé code dual de $\mathcal{C}$ et est noté $\mathcal{C}^\perp$. La matrice $\mathbf{G}$ est une matrice de parité de $\mathcal{C}^\perp$ et la matrice $\mathbf{H}$ en est une matrice génératrice. Les éléments du code dual $\mathcal{C}^\perp$ sont aussi appelés équations de parité du code $\mathcal{C}$.

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 :

$$\mathbf{x} := \mathbf{m}\mathbf{G} = (\mathbf{m}, \mathbf{m}\mathbf{A}) \tag{1.19}$$

Un tel codage est lui aussi dit systématique. Donner une matrice génératrice sous sa forme systématique permet de construire facilement une matrice de parité du code grâce à la propriété suivante :

Proposition 1.2.4. *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 $\mathcal{C}$.*

1.2.2 Les codes MDS

Définition 1.2.5 (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 $\mathbf{y} = \mathbf{x} + \mathbf{e}$. 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.

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

Théorème 1.2.6 (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 :*

$$d_\mathcal{C} \leqslant n - k + 1 \tag{1.20}$$

Définition 1.2.7 (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 1.2.8. *Soit* $\mathcal{C}$ *un code en bloc linéaire* $[n,k]_q$ *défini par une matrice génératrice* $\mathbf{G}$*. Le code* $\mathcal{C}$ *est MDS si et seulement si 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*).*

1.2.3 Les 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 1.2.9 (borne de Hamming)**.** *Soit* $\mathcal{C}$ *un code en bloc linéaire* $[n,k]_q$*. Le nombre* t *d'erreurs corrigibles respecte l'inégalité suivante :*

$$q^k \leqslant \frac{q^n}{\sum_{i=0}^{t} \binom{n}{i}(q-1)^i} \tag{1.21}$$

Définition 1.2.10 (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.

1.2.4 La borne de Gilbert-Varshamov

Une autre borne importante de la théorie des codes est la borne de Gilbert-Varshamov. C'est la plus grande distance minimale possible pour un code contenant un nombre fixé de mots. La borne de Gilbert-Varshamov n'est pas donc pas associée à un code particulier mais à une taille de code (son cardinal). Remarquons que le code n'est pas forcément linéaire.

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

Définition 1.2.11 (distance de Gilbert-Varshamov)**.** *Nous considérons des codes définis $\mathbb{F}_q^n$. Soit k un entier positif. La borne de Gilbert-Varshamov $d_{GV}(n,k)$ associée aux codes de taille q^k est :*

$$d_{GV}(n,k) := \sup\left\{r \geqslant 0 \ : \ q^k \cdot \mathbb{P}\left(\mathbf{u} \in \mathcal{B}_r\right) \leqslant 1\right\} \tag{1.22}$$

$$= \sup\left\{r \geqslant 0 \ : \ \sum_{i=0}^{r} \binom{n}{i}(q-1)^i \leqslant q^{n-k}\right\} \tag{1.23}$$

où $\mathcal{B}_r := \left\{\mathbf{x} \in \mathbb{F}_q^n \ : \ |\mathbf{x}| \leqslant r\right\}$ 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 1.2.12. *Pour un rendement $R = \frac{k}{n}$ constant et lorsque n tend vers l'infini, la distance relative de Gilbert-Varshamov est :*

$$\frac{d_{GV}(n,k)}{n} = h_q^{-1}\left(1 - \tfrac{k}{n}\right) + o(1) \tag{1.24}$$

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.

Par la suite, nous nous intéresserons à la plus petite distance de décodage $d_1(n,k)$ pour laquelle le décodage d'un mot quelconque de l'espace ambiant retourne nécessairement au moins un mot de code à distance au plus $d_1(n,k)$. Pour un code de taille q^k sur $\mathbb{F}_q^n$, cette distance est :

$$d_1(n,k) := \inf\left\{r \geqslant 0 \ : \ q^k \cdot \mathbb{P}\left(\mathbf{u} \in \mathcal{B}_r\right) \geqslant 1\right\} \tag{1.25}$$

$$= \inf\left\{r \geqslant 0 \ : \ \sum_{i=0}^{r} \binom{n}{i}(q-1)^i \geqslant q^{n-k}\right\} \tag{1.26}$$

De plus, en cryptographie, le modèle poids constant nous incitera à considérer la plus petite distance de décodage $d_2(n,k)$ telle que le décodage retourne au moins un mot de code à distance exactement $d_2(n,k)$. Nous avons donc pour le même code que précédemment :

$$d_2(n,k) := \inf\left\{r \geqslant 0 \ : \ q^k \cdot \mathbb{P}\left(\mathbf{u} \in \mathcal{S}_r\right) \geqslant 1\right\} \tag{1.27}$$

$$= \inf\left\{r \geqslant 0 \ : \ \binom{n}{r}(q-1)^r \geqslant q^{n-k}\right\} \tag{1.28}$$

Pour un rendement $R = \frac{k}{n}$ constant, bien que les quantités $d_{\mathrm{GV}}(n,k)$, $d_1(n,k)$ et $d_2(n,k)$ soient différentes, elles sont asymptotiquement équivalentes lorsque n tend vers l'infini :

$$\frac{d_{\mathrm{GV}}(n,k)}{n} = \frac{d_1(n,k)}{n} + o(1) = \frac{d_2(n,k)}{n} + o(1) \tag{1.29}$$

C'est pourquoi par la suite, nous confondrons souvent ces trois quantités.

1.3 Les codes LDPC

1.3.1 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 1.3.1 (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).

1.3.2 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 1.3.2 (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 1.4 donne un exemple de graphe de Tanner d'un code en bloc binaire linéaire $[6, 2]$.

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

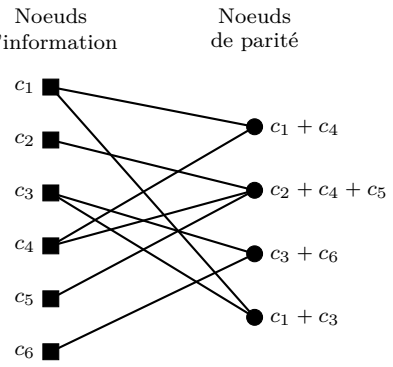


Figure 1.4 – Exemple de graphe de Tanner d'un code binaire linéaire $[6, 2]$.

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.

1.3.3 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 :

$$f_t(\mathbf{y}, i) := \begin{cases} \text{oui} & \text{si } \left|\mathbf{H}(\mathbf{y} + \mathbf{e}_i)^\top\right| = \inf\limits_{j \in [\![1,n]\!]} \left(\left|\mathbf{H}(\mathbf{y} + \mathbf{e}_j)^\top\right|\right) \neq \left|\mathbf{H}\mathbf{y}^\top\right| \\ \text{non} & \text{sinon} \end{cases} \tag{1.30}$$

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

1.3.1.

Algorithme 1.3.1 : 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 $\mathbf{H}\mathbf{c}^\top = \mathbf{0}$ et $\Delta(\mathbf{y}, \mathbf{c})$ est minimal.

```
t ← 0 ;
tant que Hy^⊤ ≠ 0 faire
    y_tmp ← y ;
    pour tout i ∈ ⟦1, n⟧ tel que f_t(y, i) = oui faire
        inverser le i^ème bit de y_tmp ;          /* nous manipulons un vecteur
         temporaire pour que la condition de la boucle soit
         indépendante de cette opération */
    finPour
    y ← y_tmp ;
finTantQue
retourner 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 $\{\Delta_i\}_{i\in[\![1,n]\!]}$:

$$f_t(\mathbf{y}, i) := \begin{cases} \text{oui} & \text{si } \Delta_i(\mathbf{y}) = \inf\limits_{j\in[\![1,n]\!]} \{\Delta_j(\mathbf{y})\} \\ \text{non} & \text{sinon} \end{cases} \tag{1.31}$$

  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 :

$$\Delta_i(\mathbf{y}) := \left|\mathbf{H}(\mathbf{y} + \mathbf{e}_i)^\top\right| \tag{1.32}$$

  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 $\mathbf{y}$) :

$$\Delta_i(\mathbf{y}) := \sum_{s=1}^{n-k} \Phi\left(\mathbf{H}_{s,*}\right) \cdot \mathbf{H}_{s,*} \cdot (\mathbf{y} + \mathbf{e}_i)^\top \tag{1.33}$$

  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 :

$$\Delta_i(\mathbf{y}) := \Psi(\mathbf{y} + \mathbf{e}_i) + \sum_{s=1}^{n-k} \Phi\left(\mathbf{H}_{s,*}\right) \cdot \mathbf{H}_{s,*} \cdot (\mathbf{y} + \mathbf{e}_i)^\top \tag{1.34}$$

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 :
$$f_t(\mathbf{y}, i) := \begin{cases} \text{oui} & \text{si } \Delta_i(\mathbf{y}) = \theta \\ \text{non} & \text{sinon} \end{cases} \tag{1.35}$$
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 :
$$f_t(\mathbf{y}, i) := \begin{cases} \text{oui} & \text{si } \Delta_i(\mathbf{y}) = \theta_t(i) \\ \text{non} & \text{sinon} \end{cases} \tag{1.36}$$
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 1.3.1, la ligne 4 est alors remplacée par :
$$4 \quad | \quad \texttt{Pour tout } i \in I' \texttt{ faire}$$
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 1.3.1, on peut par exemple remplacer la ligne 5 par :
$$\begin{array}{rll} 5 & | & \texttt{Si } \text{Bernoulli}(p) = 1 \texttt{ alors} \\ 5.1 & | & \quad | \text{ inverser le } i^{\text{ème}} \text{ bit de } \mathbf{y}_{\text{tmp}} \text{ ;} \\ 5.2 & | & \texttt{FinSi} \end{array}$$
où $p \in [0, 1]$ est un paramètre à définir et $\text{Bernoulli}(p) \in \{0, 1\}$ est tiré selon une loi de Bernoulli de paramètre p.

1.3.4 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 1.3.1. 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 1.3.2. 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 :

$$p_i := \mathbb{P}(c_i = 1 | y_i) = y_i(1 - p) + (1 - y_i)p \tag{1.37}$$

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 1.3.3. *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 $p_i := \mathbb{P}(b_i = 1)$.*

$$\mathbb{P}\left(\sum_{i=1}^{n} b_i \equiv 1 \mod 2\right) = \frac{1 - \prod_{i=1}^{n} (1 - 2p_i)}{2} \tag{1.38}$$

Lemme 1.3.4. *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 $\mathbb{P}(E_1, \ldots, E_n | b) = \mathbb{P}(E_1|b) \ldots \mathbb{P}(E_n|b)$.*

$$\mathbb{P}(b = 1 | E_1, \cdots, E_n) = \frac{\prod_{i=1}^{n} \mathbb{P}(b = 1 | E_i)}{\prod_{i=1}^{n} \mathbb{P}(b = 1 | E_i) + \prod_{i=1}^{n} (1 - \mathbb{P}(b = 1 | E_i))} \tag{1.39}$$

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 :

$$\begin{aligned} r_{j,i} &:= \mathbb{P}\left(c_i = 1 \mid (q_{s,j})_{s \in W_j \setminus \{i\}}\right) \\ &= \mathbb{P}\left(\sum_{s \in W_j \setminus \{i\}} c_s = 1 \mod 2 \;\middle|\; (q_{s,j})_{s \in W_j \setminus \{i\}}\right) \\ &= \frac{1 - \prod_{k \in W_j \setminus \{i\}} (1 - 2q_{k,j})}{2} \qquad \text{d'après le lemme 1.3.3} \end{aligned}$$

- **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 1.3.4 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 1.3.4 :

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

- 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.

1.3.5 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 1.3.5 (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 :

$$\frac{k}{n} = 1 - \frac{w_c}{w_\ell} \tag{1.40}$$

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 (1.40) 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 :

$$P(X) := \sum_{i=1}^{n-k} a_i X^{i-1} \tag{1.41}$$

$$Q(Y) := \sum_{j=1}^{n} b_j Y^{j-1} \tag{1.42}$$

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 :

$$\frac{k}{n} = 1 - \frac{\sum_{j=1}^{n} \frac{b_j}{j}}{\sum_{i=1}^{n-k} \frac{a_i}{i}} \tag{1.43}$$

Remarque 1.3.3. En fait, l'équation 1.43 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.

1.3.6 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 1.3.6 (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 :

$$\mathbf{M}_{\mathrm{C}} = \begin{bmatrix} x_1 & x_2 & x_3 & \cdots & x_m \\ x_m & x_1 & x_2 & \cdots & x_{m-1} \\ x_{m-1} & x_m & x_1 & \cdots & x_{m-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_2 & x_3 & x_4 & \cdots & x_1 \end{bmatrix} \tag{1.44}$$

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 :

$$\mathbf{M}_{\mathrm{QC}} = \left[\begin{array}{c|c|c|c} \mathbf{M}_{\mathrm{C}}^{(1,1)} & \mathbf{M}_{\mathrm{C}}^{(1,2)} & \cdots & \mathbf{M}_{\mathrm{C}}^{(1,\beta)} \\ \hline \mathbf{M}_{\mathrm{C}}^{(2,1)} & \mathbf{M}_{\mathrm{C}}^{(2,2)} & \cdots & \mathbf{M}_{\mathrm{C}}^{(2,\beta)} \\ \hline \vdots & \vdots & \vdots & \vdots \\ \hline \mathbf{M}_{\mathrm{C}}^{(\alpha,1)} & \mathbf{M}_{\mathrm{C}}^{(\alpha,2)} & \cdots & \mathbf{M}_{\mathrm{C}}^{(\alpha,\beta)} \end{array}\right] \tag{1.45}$$

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 1.3.7 (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].

1.3.7 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 1$^{\text{ère}}$ 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 :

$$\mathbf{H} := \left[\begin{array}{c|ccccc} \mathbf{H}_1 & \mathbf{Id}_m & \mathbf{0}_m & \cdots & \cdots & \mathbf{0}_m \\ \mathbf{H}_2 & \mathbf{Id}_m & \ddots & \ddots & & \vdots \\ \vdots & \mathbf{0}_m & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \mathbf{0}_m \\ \mathbf{H}_t & \mathbf{0}_m & \cdots & \mathbf{0}_m & \mathbf{Id}_m & \mathbf{Id}_m \end{array}\right] \tag{1.46}$$

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 :

$$\begin{array}{rccc} \textit{codage} : & \mathbb{F}_2^k & \longrightarrow & \mathbb{F}_2^n \\ & \mathbf{x} & \longmapsto & (\mathbf{x}, \mathbf{r}) \end{array} \tag{1.47}$$

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 1.5 représente schématiquement le codage donné par l'équation (1.47).

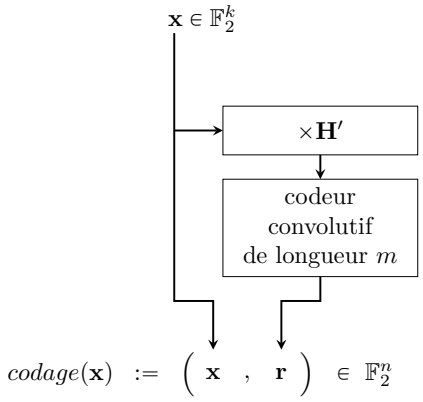


Figure 1.5 – Codage d'un code LDPC avec structure convolutive.

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 1.6 décrit alors une façon de paralléliser le codage systématique du code LDPC défini par la matrice de parité $\mathbf{H}$ donnée par l'équation (1.46).

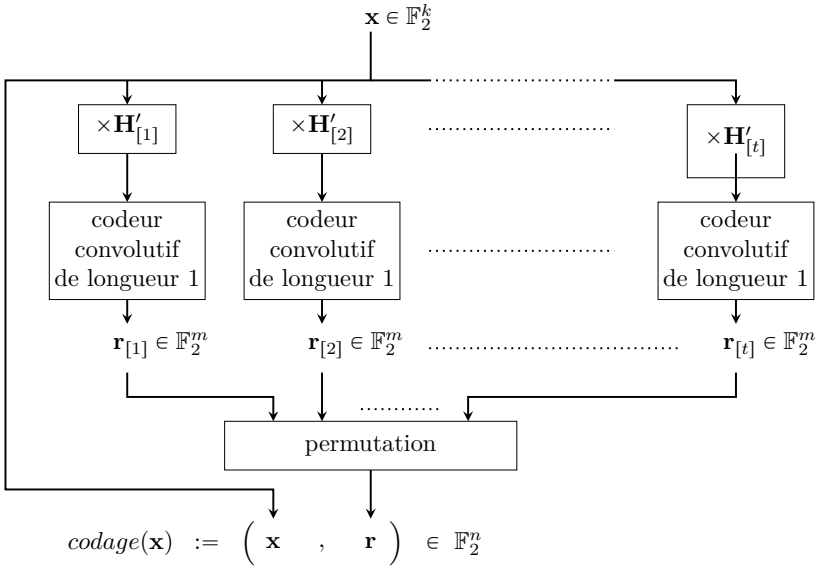


Figure 1.6 – Parallélisation du codage d'un code LDPC avec structure convolutive.

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 1.46. 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.

1.4 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.

1.4.1 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 1.4.1 (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 1.4.2 (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 :

$$\begin{cases} U_\varepsilon & := & U_0|U_0+U_1 \\ U_{\mathbf{x}} & := & U_{(\mathbf{x},0)}|U_{(\mathbf{x},0)}+U_{(\mathbf{x},1)} \end{cases} \tag{1.48}$$

Les codes $U_{\mathbf{x}}$ tels que $\mathbf{x}$ est une feuille de l'arbre sont appelés les codes constituants.

Notation 1.4.3. *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 1.7 représente cet arbre (les codes constituants sont représentés en rouge).

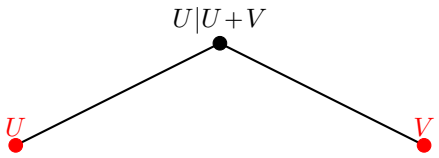


Figure 1.7 – Représentation d'un code $U|U+V$.

Un autre exemple est donné dans la figure 1.8 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) :

$$U_{000}, U_{001}, U_{01}, U_{10}, U_{110} \text{ et } U_{111}$$

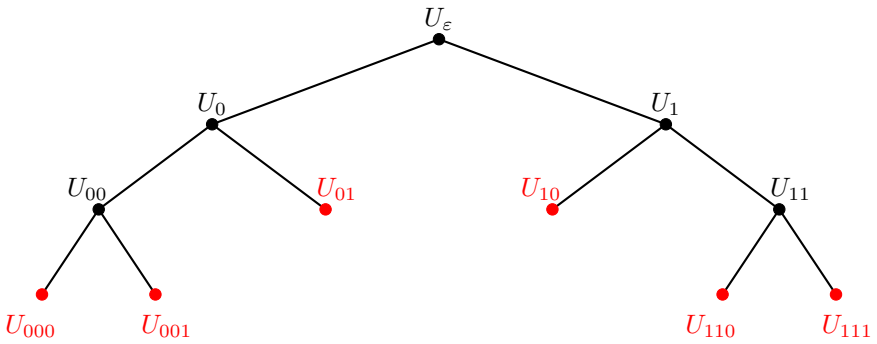


Figure 1.8 – Représentation d'un code $U|U+V$ récursif de profondeur 3.

1.4.2 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 1.4.4. *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 1.4.5 (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 1.4.6. *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 :*

$$C_\Upsilon = \sum_{\mathrm{BSC}(p)[r]\in\Upsilon} r \cdot C_{\mathrm{BSC}(p)} \tag{1.49}$$

où $C_{\mathrm{BSC}(p)} := 1 - h_2(p)$ *est la capacité du canal binaire symétrique de probabilité d'erreur* p *(cf. corollaire 1.1.6).*

Nous définissons l'opérateur $\boxplus$ qui associe deux objets de type $\mathrm{BSC}(\cdot)[\cdot]$:

Définition 1.4.7. *L'opérateur* $\boxplus$ *associe deux BSC de la manière suivante :*

$$\mathrm{BSC}(p)[r] \;\boxplus\; \mathrm{BSC}(q)[s] \;:=\; \mathrm{BSC}(p+q-2pq)[rs] \tag{1.50}$$

Proposition 1.4.8. *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 1.4.8.

$$\begin{aligned}
\mathbb{P}(y_1+y_2 = v) &= \mathbb{P}((y_1=u \text{ et } y_2=u+v) \text{ ou } (y_1\neq u \text{ et } y_2\neq u+v)) \\
&= \mathbb{P}(y_1=u)\,\mathbb{P}(y_2=u+v) + \mathbb{P}(y_1\neq u)\,\mathbb{P}(y_2\neq u+v) \\
&= (1-p)(1-q)+pq \\
&= 1-p-q+2pq
\end{aligned}$$

□

Nous pouvons généraliser la définition 1.4.7 et la proposition 1.4.8 aux BSC-compositions :

Définition 1.4.9. *L'opérateur $\boxplus$ associe deux BSC-compositions de la manière suivante :*

$$\begin{aligned}
&\left\{\mathrm{BSC}(p_i)[r_i]\right\}_{i\in[\![1,N]\!]} \boxplus \left\{\mathrm{BSC}(q_j)[s_j]\right\}_{j\in[\![1,M]\!]} \\
&\qquad := \left\{\mathrm{BSC}(p_i)[r_i] \boxplus \mathrm{BSC}(q_j)[s_j]\right\}_{(i,j)\in[\![1,N]\!]\times[\![1,M]\!]}
\end{aligned} \tag{1.51}$$

Corollaire 1.4.10. *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 1.4.11. *L'opérateur $\boxtimes$ associe deux BSC de la manière suivante :*

$$\begin{aligned}
&\mathrm{BSC}(p)[r] \boxtimes \mathrm{BSC}(q)[s] \\
&\qquad := \left\{ \begin{array}{l} \mathrm{BSC}\left(\frac{pq}{pq+(1-p)(1-q)}\right)[rs(1-p-q+2pq)] \\ \mathrm{BSC}\left(\frac{p(1-q)}{p(1-q)+q(1-p)}\right)[rs(p+q-2pq)] \end{array} \right\}
\end{aligned} \tag{1.52}$$

Remarque 1.4.1. 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 1.4.12. *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 1.4.12.

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 1.4.8, 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 :

$$p'(i) := \mathbb{P}(u=1 \mid y_1=i) = \left\{ \begin{array}{ll} p & \text{si } i=0 \\ 1-p & \text{si } i=1 \end{array} \right. \tag{1.53}$$

et

$$q'(j) := \mathbb{P}\left(u = 1 \mid y_2 + v = j\right) = \begin{cases} q & \text{si } j = 0 \\ 1-q & \text{si } j = 1 \end{cases} \tag{1.54}$$

avec i et $j \in \mathbb{F}_2$. Nous avons alors :

$$\mathbb{P}\left(u = 1 \mid y_1 = i \ , \ y_2 + v = j\right) = \frac{p'(i)q'(j)}{p'(i)q'(j) + (1-p'(i))(1-q'(j))} \tag{1.55}$$

En traitant les différentes valeurs possibles de i et j, nous avons :

$$\mathbb{P}\left(u = y_1 \mid y_1 + y_2 = v\right) = \frac{(1-p)(1-q)}{pq + (1-p)(1-q)} \tag{1.56}$$

$$\mathbb{P}\left(u \neq y_1 \mid y_1 + y_2 = v\right) = \frac{pq}{pq + (1-p)(1-q)} \tag{1.57}$$

$$\mathbb{P}\left(u = y_1 \mid y_1 + y_2 \neq v\right) = \frac{(1-p)q}{pq + (1-p)(1-q)} \tag{1.58}$$

$$\mathbb{P}\left(u \neq y_1 \mid y_1 + y_2 \neq v\right) = \frac{p(1-q)}{pq + (1-p)(1-q)} \tag{1.59}$$

Or comme $y_1 + y_2$ est une altération du bit v après qu'il ait été transporté par le canal $\mathrm{BSC}\left(p\right)\left[1\right] \boxplus \mathrm{BSC}\left(q\right)\left[1\right]$, nous savons d'après la proposition 1.4.8 que :

$$\mathbb{P}\left(y_1 + y_2 \neq v\right) = p + q - 2pq \tag{1.60}$$

Nous pouvons alors en déduire la proposition. □

Nous pouvons généraliser la définition 1.4.11 et la proposition 1.4.12 aux *BSC*-compositions :

Définition 1.4.13. *L'opérateur $\boxtimes$ associe deux BSC-compositions de la manière suivante :*

$$\begin{aligned} \left\{ \mathrm{BSC}\left(p_i\right)\left[r_i\right] \right\}_{i \in [\![1,N]\!]} \ \boxtimes \ \left\{ \mathrm{BSC}\left(q_j\right)\left[s_j\right] \right\}_{j \in [\![1,M]\!]} \\ := \ \left\{ \mathrm{BSC}\left(p_i\right)\left[r_i\right] \ \boxtimes \ \mathrm{BSC}\left(q_j\right)\left[s_j\right] \right\}_{(i,j) \in [\![1,N]\!] \times [\![1,M]\!]} \end{aligned} \tag{1.61}$$

Remarque 1.4.2. 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 1.4.14. *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 1.4.2. 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 :

$$\Upsilon_\varepsilon = \mathrm{BSC}\left(p\right)\left[1\right] \tag{1.62}$$

$$\Upsilon_{(\mathbf{x},1)} = \Upsilon_{\mathbf{x}} \boxplus \Upsilon_{\mathbf{x}} \tag{1.63}$$

$$\Upsilon_{(\mathbf{x},0)} = \Upsilon_{\mathbf{x}} \boxtimes \Upsilon_{\mathbf{x}} \tag{1.64}$$

La figure 1.9 décrit alors la décomposition du canal lors du décodage du code $U|U+V$ de l'exemple de la figure 1.8.

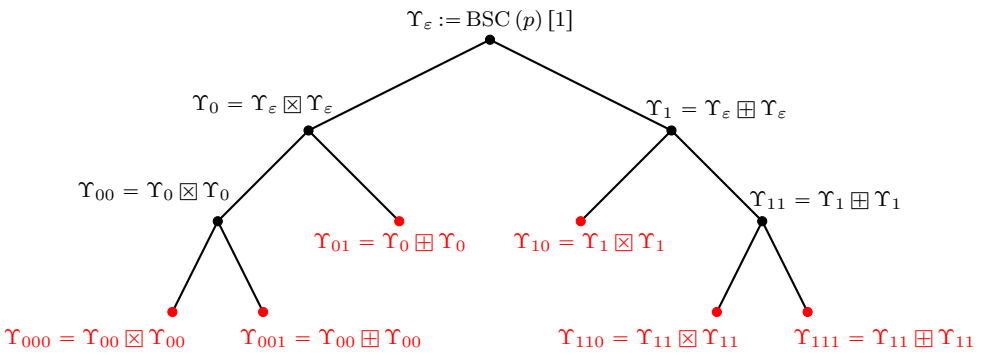


Figure 1.9 – Arbre binaire représentant les différents canaux d'un code $U|U+V$ récursif.

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 $\left[\frac{t}{M}, \frac{t+1}{M}\right]$:

$$\Upsilon(t) := \left\{\mathrm{BSC}\,(p)\,[r] \in \Upsilon \;:\; p \in \left[\tfrac{t}{M}, \tfrac{t+1}{M}\right]\right\} \tag{1.65}$$

Nous approchons alors $\Upsilon(t)$ par :

$$\Upsilon(t) \simeq \widetilde{\Upsilon}(t) := \mathrm{BSC}\left(\tfrac{t}{M} + \tfrac{1}{2M}\right)[R] \quad \text{où} \quad R := \sum_{\mathrm{BSC}(p)[r] \in \Upsilon(t)} r \tag{1.66}$$

Et donc par extension, nous approchons Υ par la BSC-composition suivante :

$$\Upsilon \simeq \left\{\widetilde{\Upsilon}(t)\right\}_{t \in [\![0, M-1]\!]} \tag{1.67}$$

Finalement, l'algorithme 1.4.1 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 1.4.2.

Algorithme 1.4.1 : 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**
    $\Upsilon_{(\mathbf{x},1)} \leftarrow \Upsilon_{\mathbf{x}} \boxplus \Upsilon_{\mathbf{x}}$;
    $\Upsilon_{(\mathbf{x},1)} \leftarrow \textsc{Simplifier}(\Upsilon_{(\mathbf{x},1)}, M)$;
    $\Upsilon_{(\mathbf{x},0)} \leftarrow \Upsilon_{\mathbf{x}} \boxplus \Upsilon_{\mathbf{x}}$;
    $\Upsilon_{(\mathbf{x},0)} \leftarrow \textsc{Simplifier}(\Upsilon_{(\mathbf{x},0)}, M)$;
  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

1.4.3 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 1.4.1 permet de décrire les canaux via lesquels transiteront les mots des codes constituants. Grâce à la proposition 1.4.6, 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 1.2 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 1.4.2 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 1.4.2 : 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 1.4.6 */
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 1.4.2 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 :

$$p := h_2^{-1}(1-R) \tag{1.68}$$

Toutefois, quitte à s'écarter légèrement des rendements donnés par l'algorithme 1.4.1, il faut faire en sorte de choisir des codes constituants tels que la somme de leurs dimensions soit $\lfloor Rn \rfloor$.

1.4.4 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 $\left\{U_{\mathbf{x}} \text{ tel que } \mathbf{x} \in \{0,1\}^d\right\}$ 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 1.4.3. 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 1.4.2 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.

1.4.5 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 1.4.15 spécifie plus formellement cette notion.

Définition 1.4.15. *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 :*

$$\prod_{i=1}^{n} {p_i}^{c_i} (1-p_i)^{(1-c_i)} \tag{1.69}$$

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 1.4.15. 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 1.4.15.

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 1.4.16. *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 1.4.16.

$$\begin{aligned} {p_i}^{c_i} (1-p_i)^{(1-c_i)} &= \left({p'_i}^{(1-y_i)} (1-p'_i)^{y_i} \right)^{c_i} \left({p'_i}^{y_i} (1-p'_i)^{(1-y_i)} \right)^{(1-c_i)} \\ &= {p'_i}^{((1-y_i)c_i + y_i(1-c_i))} \times (1-p'_i)^{(y_i c_i + (1-y_i)(1-c_i))} \\ &= {p'_i}^{e_i} (1-p'_i)^{(1-e_i)} \end{aligned}$$

□

D'après le lemme 1.4.16, 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 1.4.17. *L'ordre décroissant des* $\prod_{i=1}^{n} {p'_i}^{e_i} (1-p'_i)^{(1-e_i)}$ *est exactement l'ordre décroissant des* $\sum_{i \in S} \log_2 \left(\frac{p'_i}{1-p'_i} \right)$ *avec* S *le support de* $\mathbf{e}$.

Démonstration du lemme 1.4.17.

$$\begin{aligned} \log_2 \left(\prod_{i=1}^{n} {p'_i}^{e_i} (1-p'_i)^{(1-e_i)} \right) &= \sum_{i=1}^{n} \log_2 \left({p'_i}^{e_i} (1-p'_i)^{(1-e_i)} \right) \\ &= \sum_{i=1}^{n} \log_2 \left(1-p'_i \right) + \sum_{i \in S} \log_2 \left(\frac{p'_i}{1-p'_i} \right) \end{aligned}$$

Or la fonction $\log_2$ est une fonction croissante et donc elle ne change pas l'ordre. De plus $\sum_{i=1}^{n} \log_2 \left(1-p'_i \right)$ est constant quel que soit $\mathbf{e} \in \mathbb{F}_2^n$. D'où finalement le lemme. □

Nous proposons finalement l'algorithme 1.4.3 pour décoder un code aléatoire de codimension faible avec de l'information souple.

Proposition 1.4.18. *L'algorithme 1.4.3 énumère les vecteurs d'erreur* $\mathbf{e} := (e_1, \cdots, e_n) \in \mathbb{F}_2^n$ *de support* $\mathrm{supp}(\mathbf{e})$ *dans l'ordre décroissant des :*

$$\sum_{i \in \mathrm{supp}(\mathbf{e})} \log_2 \left(\frac{p'_i}{1 - p'_i} \right) \tag{1.70}$$

Démonstration de la proposition 1.4.18.

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 1.4.3 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 1.4.3, 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 1.4.16 et 1.4.17, montrent que l'ordre de la proposition 1.4.18 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 $O\left(2^{n-k}\right)$ 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 1.4.4. Dans l'algorithme 1.4.3, 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 $O\left(2^{n-k}\right) \leqslant O\left(2^{\Gamma}\right)$. Ainsi, la taille de la file de priorité est d'au plus de l'ordre de $O\left(2^{\Gamma}\right)$.

Finalement, pour les codes aléatoires de dimension ou codimension faible, nous avons le théorème suivant :

Théorème 1.4.19. *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* $O\left(2^{\Gamma}\right)$.

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 1.4.2 peuvent

être réalisés en un temps constant.

Algorithme 1.4.3 : 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

1.4.6 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 1.4.2. 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 1.4.20. *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 :

$$\forall i \in [\![1, n]\!] \ , \ (\mathbf{p} \boxtimes^{\mathbf{x}} \mathbf{q})_i := \begin{cases} \dfrac{p_i(1-q_i)}{p_i(1-q_i) + q_i(1-p_i)} & \text{si } x_i = 1 \\ \dfrac{p_i q_i}{p_i q_i + (1-p_i)(1-q_i)} & \text{si } x_i = 0 \end{cases}$$

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 1.4.4 décrit ce décodage.

Algorithme 1.4.4 : 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 1.4.3 */`
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 1.4.20 */`
8 $\quad\quad$ $\mathbf{u} \leftarrow$ DECODE($\mathbf{p}' \boxtimes^{\mathbf{v}} \mathbf{p}''$, $\frac{n}{2}$, $(\mathbf{x}, 0)$, $\mathscr{T}$) ; `/* cf. définition 1.4.20 */`
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 1.4.4. 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 1.4.4. 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 1.4.5 donne la version itérative de l'algorithme 1.4.4 de décodage par annulation successive.

Algorithme 1.4.5 : 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` $\mathbf{x}$ `est un nœud de l'étage` i `de` $\mathscr{T}$ `(donc` $\mathbf{x}$ `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` $\mathbf{x}$ `est un nœud de l'étage` i `de` $\mathscr{T}$ `et` $\mathbf{x}$ `est un fils gauche (c'est-à-dire que le bit le plus à droite de` $\mathbf{x}$ `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` $\mathbf{x}$ `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 1.4.20 */`
    $\mathbf{x} \leftarrow (\mathbf{x}, 1)$;
    $i \leftarrow i + 1$;
  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 1.4.20 */`
  $\mathbf{x} \leftarrow (x_i, \cdots, x_2, 0)$;
finRépéter

1.4.7 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 1.4.21. *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 :*

$$\mathbb{E}\left(\Delta\left(\mathbf{x}, D(\mathbf{x})\right)\right) := \sum_{d=0}^{n} d \cdot \mathbb{P}\left(\Delta\left(\mathbf{x}, D(\mathbf{x})\right) = d\right) \tag{1.71}$$

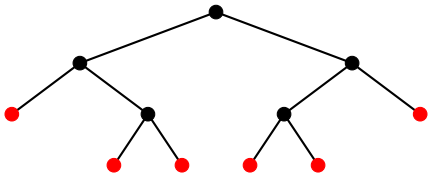
où $\mathbf{x}$ est tiré uniformément dans $\mathbb{F}_2^n$.

Un décodage est d'autant plus performant que sa distorsion est proche de la distance de Gilbert-Varshamov (cf. sous-section 1.2.4).

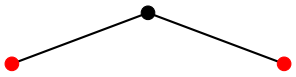
Commençons par étudier un cas simple : un code $U|U+V$ récursif de longueur 32 et de dimension 16 construit avec l'algorithme 1.4.2. Ce code, que nous notons $\mathcal{C}_\Gamma^{[32,16]}$, dépend de la dimension ou codimension maximale Γ des codes constituants. Observons alors la distorsion de $\mathcal{C}_\Gamma^{[32,16]}$ en fonction de Γ que nous avons tracé sur la figure 1.10. 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 1.10 – 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 1.4.2 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 1.10 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 1.4.2 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 1.4.2 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 1.11 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 $[1024, 512]$.

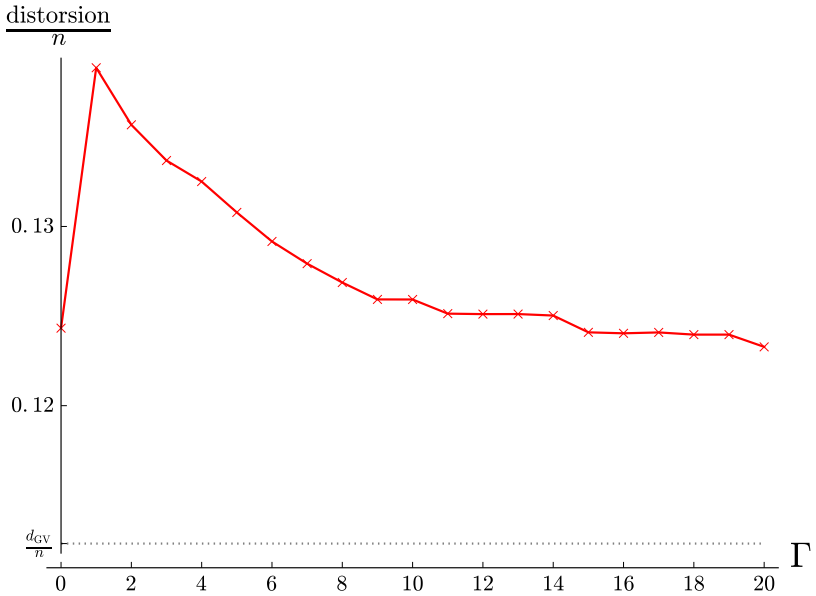


Figure 1.11 – Distorsion du décodage par annulation successive de codes $U|U+V$ récursifs $[1024, 512]$ en fonction de la dimension ou codimension maximale Γ des codes constituants.

Sur les figures 1.10 et 1.11, 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 1.4.2 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 1.4.22 précise cette construction.

Définition 1.4.22 (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 1.4.2, 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.

1.4.8 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 1.4.5 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 1.12 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 1.13 montre la distorsion d'un code polaire $[1024, 512]$ en fonction de la taille de la liste de décodage.

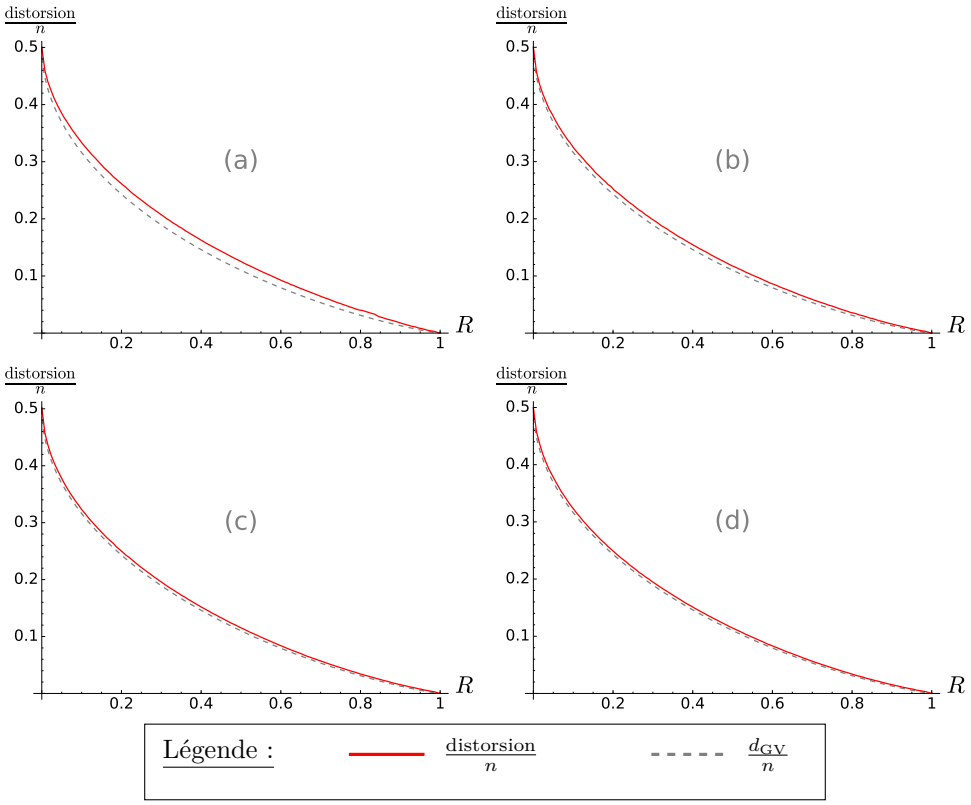


Figure 1.12 – Distorsion du décodage de Tal et Vardy pour un code polaire de longueur $n = 1024$ en fonction du rendement R. (a) $\ell = 1$, (b) $\ell = 10$, (c) $\ell = 100$, (d) $\ell = 1000$.

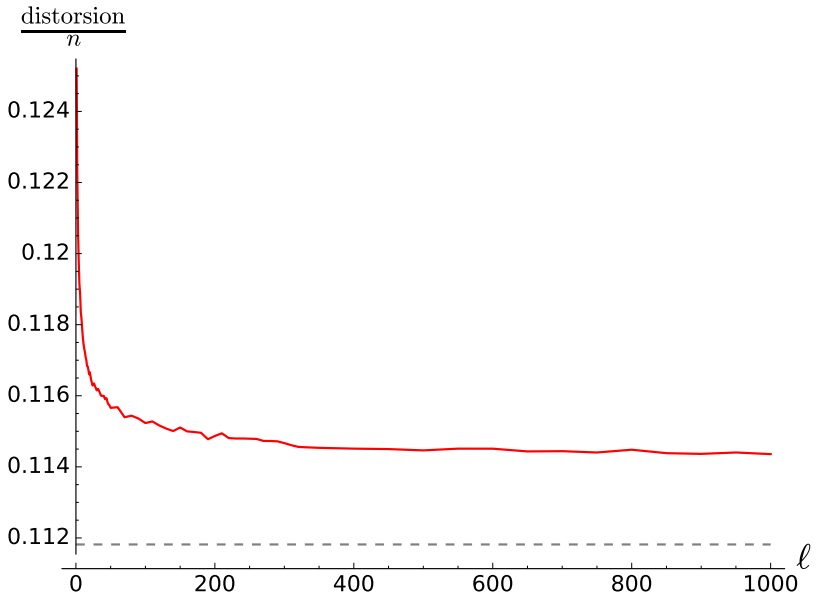


Figure 1.13 – Distorsion du décodage de Tal et Vardy pour un code polaire $[1024, 512]$ en fonction de la taille ℓ de la liste de décodage.

Remarque 1.4.5. Remarquons que le décodage des codes constituants de dimensions ou codimensions faibles que nous avons vu dans la sous-section 1.4.5 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 1.4.3. 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 2

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 2.2, nous présentons quelques crypto-systèmes basés sur le problème du décodage générique. La section 2.3 définit formellement le problème du décodage générique et reformule celui-ci dans une version dual. Enfin, les sections 2.4 à 2.7 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 2.8, 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].

2.1 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.

2.2 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, CT19a])
- des codes de Goppa "sauvages" non-binaires [BLP10] (partiellement cassé dans [COT14, FPdP14]) ;
- des codes de Srivastava généralisés [BBB+17a] (chiffrement proposé à la compétition du NIST sous le nom de *DAGS*) (cassé dans [BC18, BBCO19]) ;
- d'autres codes de Goppa binaires [BBB+17b] (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 [MRAS00, 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, FHS+17] 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 2.1 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 2.1 – Paramètres de *BIKE*.

Le but du tableau 2.1 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 2.2 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 2.2 – Paramètres de *WAVE*.

2.3 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 2.3.1 (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 2.3.1, nous nous intéresserons au problème équivalent du décodage par syndrome (en anglais, *syndrome decoding problem*) :

Problème 2.3.2 (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 2.3.2 est tel que $\mathbf{y} - \mathbf{e}$ est un mot du code $\mathcal{C}$ puisque par construction, $\mathbf{H}\left(\mathbf{y} - \mathbf{e}\right)^\top = \mathbf{H}\mathbf{y}^\top - \mathbf{H}\mathbf{e}^\top = \mathbf{s} - \mathbf{s} = \mathbf{0}$.

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 :

$$2^{(\alpha + o(1))n} \tag{2.1}$$

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 2.1, nous avons représenté l'exposant α de l'équation (2.1) 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*).

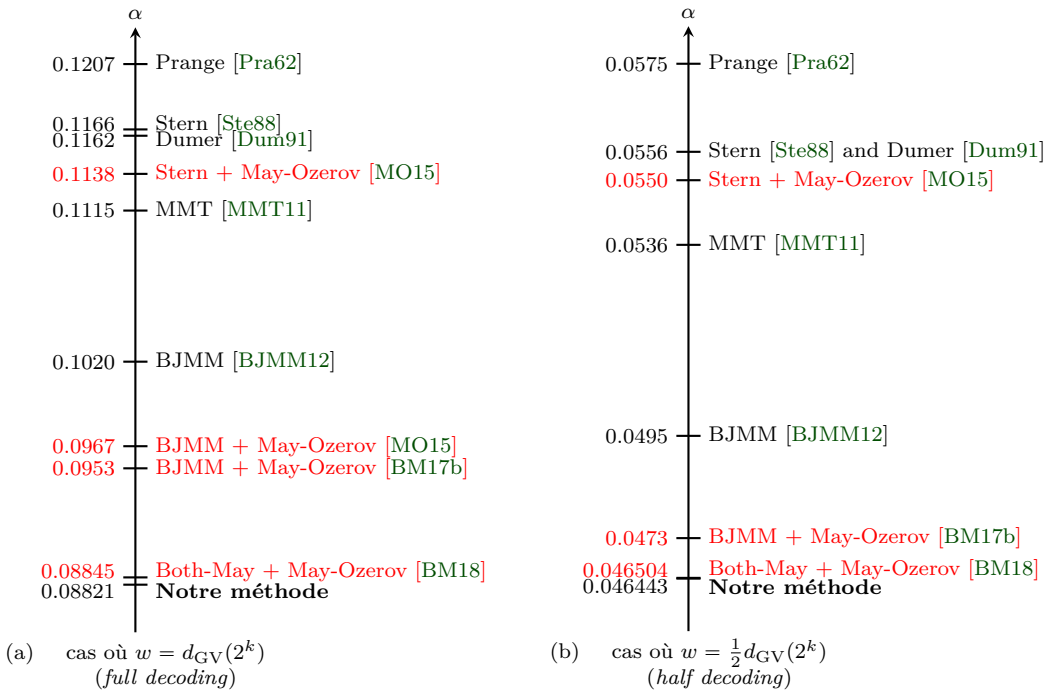


Figure 2.1 – Exposants des complexités asymptotiques des algorithmes de décodage par ensemble d'information dans le cas binaire.

Sur la figure 2.1, 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 2.8 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].

2.4 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 2.4.1. *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 :*

$$\mathbf{x}_I := (x_{i_1}, \cdots, x_{i_t}) \in \mathbb{F}_q^t \tag{2.2}$$

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 2.4.2. *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 2.4.1, 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 2.4.1 : 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 $\mathbf{H}\mathbf{e}^\top = \mathbf{s}^\top$ 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 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 :

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

Remarque 2.4.1. En choisissant un ensemble de k indices aléatoirement lors d'une itération de l'algorithme 2.4.1, 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 2.4.3. *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 :*

$$T_{Prange} = O\left(\frac{\min\left(\binom{n}{w}(q-1)^w, q^{n-k}\right)}{\binom{n-k}{w}(q-1)^w}\right) \tag{2.3}$$

$$= q^{n\left(\min(1-R, h_q(\omega)) - (1-R)h_q\left(\frac{\omega}{1-R}\right)\right)(1+o(1))} \tag{2.4}$$

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

Remarque 2.4.2. 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 $\omega \in \left[\left(1-\frac{1}{q}\right)(1-R), R + \left(1-\frac{1}{q}\right)(1-R)\right]$.

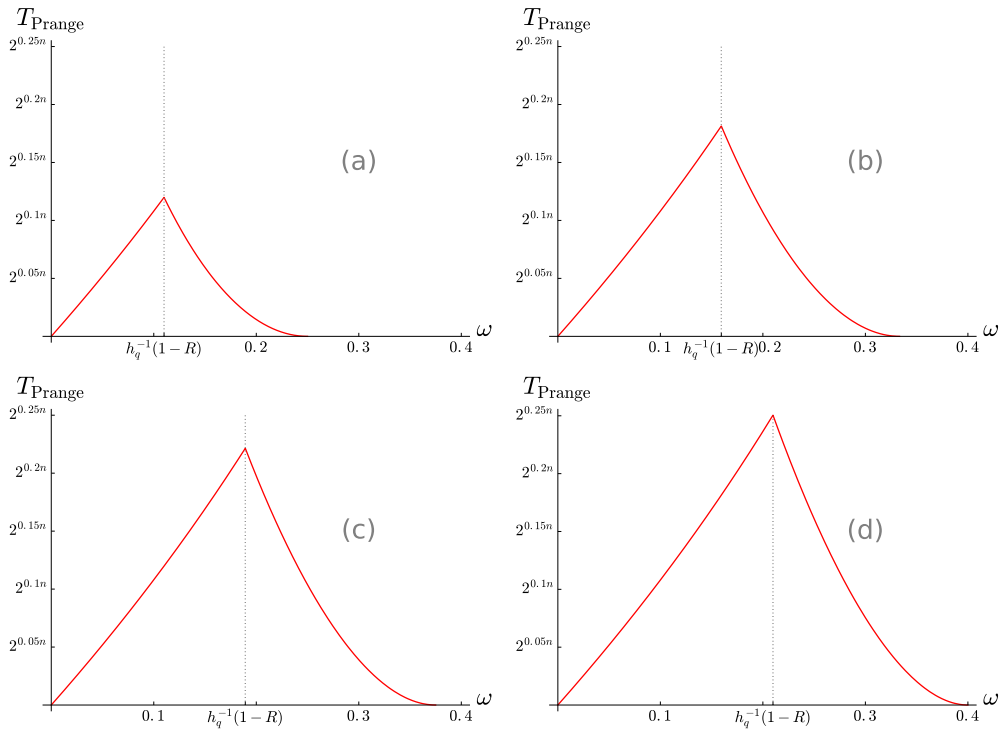


Figure 2.2 – Complexité asymptotique de l'algorithme de Prange pour $R = \frac{1}{2}$. (a) $q = 2$, (b) $q = 3$, (c) $q = 4$, (d) $q = 5$.

La figure 2.2 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 2.3 donne ce même exposant en fonction de R pour un décodage à la distance de Gilbert-Varshamov.

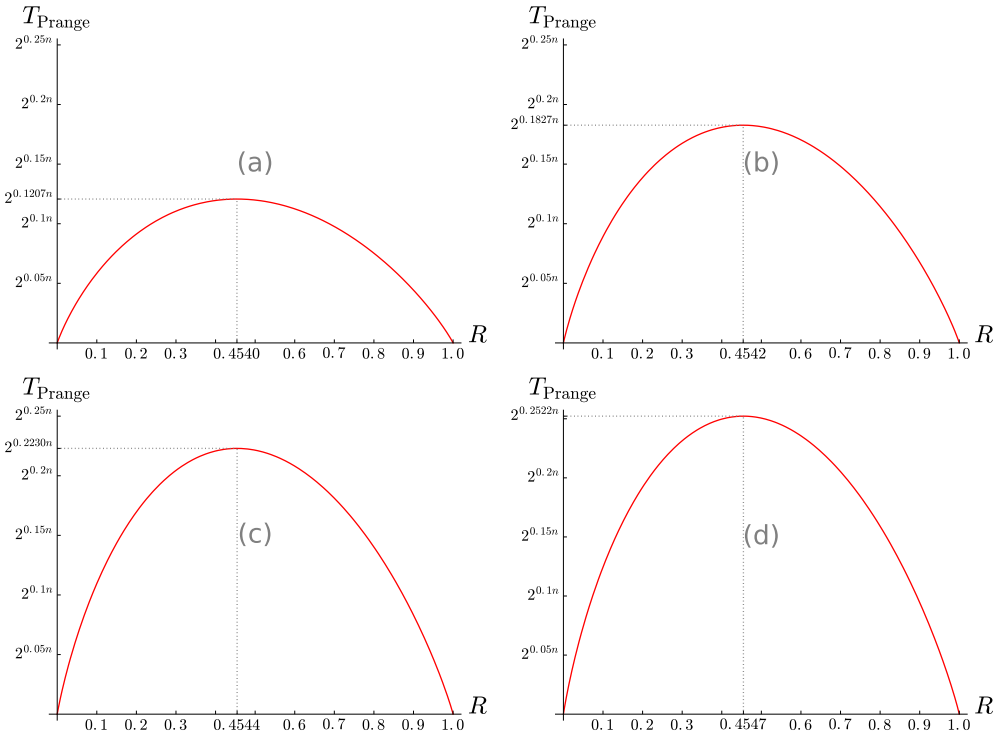


Figure 2.3 – Complexité asymptotique de l'algorithme de Prange pour $\omega = \frac{d^-_{\text{GV}}(n,k)}{n} := h_q^{-1}(1-R)$. (a) $q = 2$, (b) $q = 3$, (c) $q = 4$, (d) $q = 5$.

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 :

$$T_{\text{Prange}} = O\left(\frac{\binom{n}{w}}{\binom{n-k}{w}}\right) \tag{2.5}$$

$$= \tilde{O}\left(q^{n\left(h_q(\omega)-(1-R)h_q\left(\frac{\omega}{1-R}\right)\right)}\right) \tag{2.6}$$

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 2.4.1 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 2.4.4. *Soient un code linéaire $[n,k]_q$ et un entier w tels que $k := \lfloor Rn \rceil$ et $w := \lfloor \omega n \rceil$ 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 :*

$$T_{\text{Lee-Brickell}} = \tilde{O}(q^{\alpha n}) \tag{2.7}$$

où :

$$\alpha := \gamma \log_q(q-1) + h_q(\omega) - (1-R)h_q\left(\tfrac{\omega-\gamma}{1-R}\right) \tag{2.8}$$

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

2.5 Le décodage par recherche de collisions

2.5.1 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 2.5.1, 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 2.4 (éventuellement à permutation près des lignes).

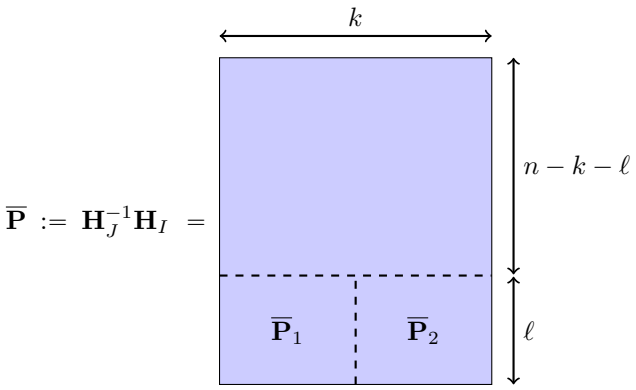


Figure 2.4 – Découpage de $\overline{\mathbf{P}}$ pour la méthode de Stern.

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

$$\mathscr{L}_1 := \left\{ \bar{\mathbf{s}}_L - \overline{\mathbf{P}}_1 \mathbf{x}_1^\top \in \mathbb{F}_q^\ell \ : \ \mathbf{x}_1 \in \mathbb{F}_q^{\frac{k}{2}} \text{ et } |\mathbf{x}_1| = \tfrac{p}{2} \right\} \tag{2.9}$$

$$\mathscr{L}_2 := \left\{ \overline{\mathbf{P}}_2 \mathbf{x}_2^\top \in \mathbb{F}_q^\ell \ : \ \mathbf{x}_2 \in \mathbb{F}_q^{\frac{k}{2}} \text{ et } |\mathbf{x}_2| = \tfrac{p}{2} \right\} \tag{2.10}$$

Remarque 2.5.1. 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 :

$$\mathbb{P}_{\text{succ}} = \frac{\binom{k/2}{p/2}^2 \binom{n-k-\ell}{w-p}}{\binom{n}{w}} \tag{2.11}$$

Algorithme 2.5.1 : 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 $|I| = k$;
  $J \leftarrow [\![1, n]\!] \setminus I$;
  $\overline{\mathbf{s}} \leftarrow \mathbf{H}_J^{-1}\mathbf{s}^\top$; `/* si H_J n'est pas inversible revenir à l'étape 2 */`
  $\overline{\mathbf{P}} \leftarrow \mathbf{H}_J^{-1}\mathbf{H}_I$;
  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 2.4 */`
  $\mathscr{L}_1 \leftarrow \left\{ \overline{\mathbf{s}}_L - \overline{\mathbf{P}}_1\mathbf{x}_1^\top \ : \ \mathbf{x}_1 \in \mathbb{F}_q^{\frac{k}{2}} \text{ et } |\mathbf{x}_1| = \frac{p}{2} \right\}$;
  $\mathscr{L}_2 \leftarrow \left\{ \overline{\mathbf{P}}_2\mathbf{x}_2^\top \ : \ \mathbf{x}_2 \in \mathbb{F}_q^{\frac{k}{2}} \text{ et } |\mathbf{x}_2| = \frac{p}{2} \right\}$;
  pour tout $\overline{\mathbf{s}}_L - \overline{\mathbf{P}}_1\mathbf{x}_1^\top = \overline{\mathbf{P}}_2\mathbf{x}_2^\top \in \mathscr{L}_1 \cap \mathscr{L}_2$ **faire**
    $\mathbf{x} \leftarrow (\mathbf{x}_1, \mathbf{x}_2)$;
    construire $\mathbf{e}$ tel que $\mathbf{e}_I = \mathbf{x}$ et $\mathbf{e}_J^\top = \overline{\mathbf{s}} - \overline{\mathbf{P}}\mathbf{x}^\top$;
    si $|\mathbf{e}| = w$ **alors**
      retourner $\mathbf{e}$;
    finSi
  finPour
finRépéter

Théorème 2.5.1. *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 :*

$$T_{Stern} = \tilde{O}(q^{\alpha n}) \tag{2.12}$$

où :

$$\alpha := h_q(\omega) - \lambda - (1 - R - \lambda) h_q\left(\frac{\omega - \gamma}{1 - R - \lambda}\right) \tag{2.13}$$

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 $p := 2\lfloor \frac{\gamma n}{2} \rfloor$.

2.5.2 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 2.5.2 (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 2.5.2 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 2.5.

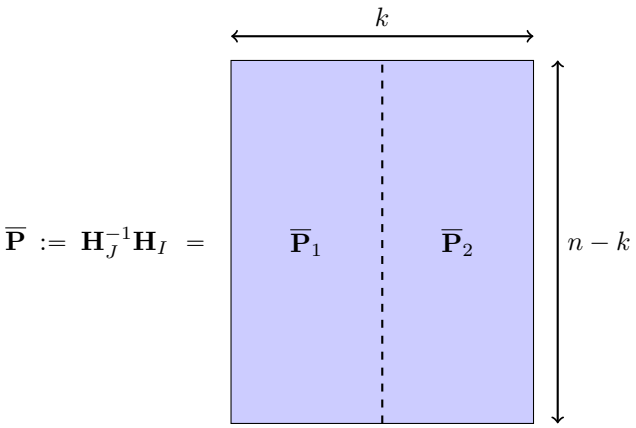


Figure 2.5 – Découpage de $\overline{\mathbf{P}}$ pour la version de May et Ozerov de la méthode de Stern.

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

$$\begin{aligned}\mathscr{L}_1 &:= \left\{\overline{\mathbf{s}} - \overline{\mathbf{P}}_1\mathbf{x}_1^\top \in \mathbb{F}_q^{n-k} \ : \ \mathbf{x}_1 \in \mathbb{F}_q^{k/2} \text{ et } |\mathbf{x}_1| = \tfrac{p}{2}\right\}\\ \mathscr{L}_2 &:= \left\{\overline{\mathbf{P}}_2\mathbf{x}_2^\top \in \mathbb{F}_q^{n-k} \ : \ \mathbf{x}_2 \in \mathbb{F}_q^{k/2} \text{ et } |\mathbf{x}_2| = \tfrac{p}{2}\right\}\end{aligned}$$

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 2.5.2 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 2.5.2 dans [MO15]. Nous présenterons leur méthode plus en détail dans le chapitre 5 section 5.5. Nous proposerons ensuite une méthode utilisant des codes dans les chapitres 5 à 9. Dans ce chapitre, nous supposons une boite noire qui résout le problème 2.5.2.

Finalement, le pseudo-code 2.5.2 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 2.5.2 : 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 2.5;
  $\mathscr{L}_1 \leftarrow \left\{ \overline{\mathbf{s}} - \overline{\mathbf{P}}_1\mathbf{x}_1^\top \in \mathbb{F}_q^{n-k} \ : \ \mathbf{x}_1 \in \mathbb{F}_q^{k/2} \text{ et } |\mathbf{x}_1| = \frac{p}{2} \right\}$;
  $\mathscr{L}_2 \leftarrow \left\{ \overline{\mathbf{P}}_2\mathbf{x}_2^\top \in \mathbb{F}_q^{n-k} \ : \ \mathbf{x}_2 \in \mathbb{F}_q^{k/2} \text{ et } |\mathbf{x}_2| = \frac{p}{2} \right\}$;
  $\mathscr{L}^* \leftarrow \textsc{PresqueCollisions}\,(\mathscr{L}_1,\ \mathscr{L}_2,\ w-p)$;
  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 $\mathbf{e}$;
  finSi
finRépéter

Théorème 2.5.3. *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 2.5.2 paramétré avec $p := 2\left\lfloor \frac{\gamma n}{2} \right\rfloor$ est :*

$$T_{Stern+MO} = \tilde{O}(q^{\alpha n}) \tag{2.14}$$

où :

$$\alpha := \beta + h_q(\omega) - Rh_q\left(\frac{\gamma}{R}\right) - (1-R)h_q\left(\frac{\omega-\gamma}{1-R}\right) \tag{2.15}$$

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.

2.5.3 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 :

$$I_1 \subseteq [\![1,n]\!] \text{ tel que } \#I_1 = \tfrac{n}{2} \tag{2.16}$$
$$I_2 := [\![1,n]\!] \setminus I_1 \tag{2.17}$$

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 :

$$\mathscr{L}_1 := \left\{ \mathbf{s} - \mathbf{P}_1 \mathbf{x}_1^\top \in \mathbb{F}_q^{n-k} \ : \ \mathbf{x}_1 \in \mathbb{F}_q^{\frac{n}{2}} \text{ et } |\mathbf{x}_1| = \tfrac{w}{2} \right\} \tag{2.18}$$

$$\mathscr{L}_2 := \left\{ \mathbf{P}_2 \mathbf{x}_2^\top \in \mathbb{F}_q^{n-k} \ : \ \mathbf{x}_2 \in \mathbb{F}_q^{\frac{n}{2}} \text{ et } |\mathbf{x}_2| = \tfrac{w}{2} \right\} \tag{2.19}$$

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 :

$$\mathbb{P}_{\text{succ}} = \frac{\binom{n/2}{w/2}^2}{\binom{n}{w}} = \text{poly}\,(n) \tag{2.20}$$

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

$$T_{\text{Dumer89}} = \tilde{O}\left(\binom{n/2}{w/2}(q-1)^{\frac{w}{2}} + \frac{\binom{n}{w}(q-1)^w}{q^{n-k}} \right) \tag{2.21}$$

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 2.5.4. *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 :*

$$\mathcal{C}' := \left\{ \mathbf{x} \in \mathbb{F}_q^{n-r} \ : \ \exists \mathbf{c} \in \mathcal{C}, \mathbf{c}_I = \mathbf{x} \right\} \tag{2.22}$$

Proposition 2.5.5. *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 2.5.5.

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 2.6 résume la construction de $\mathbf{H}'$. □

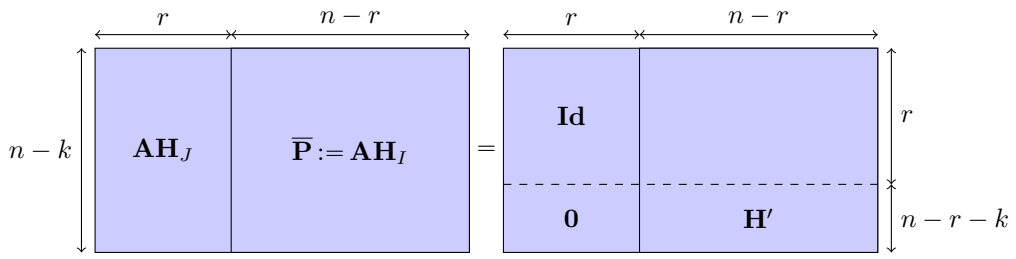


Figure 2.6 – Construction d'une matrice de parité $\mathbf{H}'$ du code poinçonné de $\mathcal{C}$ en J à partir d'une matrice de parité $\mathbf{H}$ de $\mathcal{C}$.

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 2.5.5) ;

(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 2.5.6. *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 :*

$$T_{Dumer91} = \tilde{O}(q^{\alpha n}) \tag{2.23}$$

où :

$$\alpha := h_q(\omega) - 1 + \rho + R - \rho h_q\left(\tfrac{\omega - \gamma}{\rho}\right) \tag{2.24}$$

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 2.5.6.

D'après l'équation (2.21), 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 :

$$\mathbb{P}_{\text{succ}} = \frac{\binom{n-r}{p}\binom{r}{w-p}}{\binom{n}{w}} \tag{2.25}$$

Finalement, on a :

$$T_{\text{Dumer91}}(r,p) = \tilde{O}\left(\frac{\binom{n}{w}}{\binom{n-r}{p}\binom{r}{w-p}}\left(\binom{(n-r)/2}{p/2}(q-1)^{\frac{p}{2}} + \frac{\binom{n-r}{p}(q-1)^p}{q^{n-r-k}}\right)\right) \quad (2.26)$$

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. □

2.6 La technique des représentations

2.6.1 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 2.6.1. *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 2.6.2. *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 :*

$$R_q(n, w, \overline{w}) := \tilde{\Theta}\left(q^{\alpha n}\right) \quad (2.27)$$

où :

$$\begin{aligned} \alpha \quad := \quad & (1-\omega) h_q\left(\frac{\gamma}{1-\omega}\right) + \omega h_q\left(\frac{\overline{\omega}-\gamma}{\omega}\right) + (\overline{\omega}-\gamma) h_q\left(2 - \frac{\omega}{\overline{\omega}-\gamma}\right) \\ & + (2(\overline{\omega}-\gamma)-\omega)\log_q(q-2) - (3(\overline{\omega}-\gamma)-\omega)\log_q(q-1) \end{aligned} \quad (2.28)$$

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 2.6.2.

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 $\mathcal{S}(\mathbf{x}, \overline{w})$:

$$\mathcal{S}(\mathbf{x}, \overline{w}) := \left\{\mathbf{x}_1 \in \mathbb{F}_q^n : \Delta(\mathbf{x}, \mathbf{x}_1) = \overline{w}\right\} \quad (2.29)$$

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 2.7.

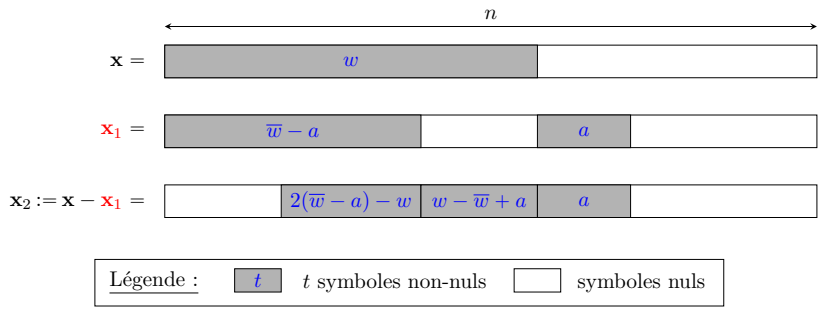


Figure 2.7 – Une configuration possible du vecteur $\mathbf{x}_1 \in \mathcal{S}(\mathbf{x}, \overline{w})$.

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

$$\begin{cases} 0 \leqslant & a & \leqslant n-w \\ 0 \leqslant & \overline{w}-a & \leqslant w \\ 0 \leqslant & 2(\overline{w}-a)-w & \leqslant \overline{w}-a \end{cases} \tag{2.30}$$

C'est-à-dire :

$$a \in [\![0, n-w]\!] \cap \left[\!\left[\overline{w}-w, \overline{w}-\frac{w}{2}\right]\!\right] \tag{2.31}$$

On a alors :

$$\#\left(\mathcal{S}(\mathbf{x}, \overline{w})\right) = \sum_{a=\max(0,\overline{w}-w)}^{\min\left(n-w, \overline{w}-\frac{w}{2}\right)} \binom{n-w}{a}\binom{w}{\overline{w}-a}\binom{\overline{w}-a}{2(\overline{w}-a)-w}(q-1)^a(q-2)^{2(\overline{w}-a)-w} \tag{2.32}$$

La formule de Stirling (cf. théorème 1.1.4) nous donne une version asymptotique de cette formule :

$$\#\left(\mathcal{S}(\mathbf{x}, \overline{w})\right) = \tilde{\Theta}\left(\sup_{\gamma=\max(0,\overline{\omega}-\omega)}^{\min\left(1-\omega, \overline{\omega}-\frac{\omega}{2}\right)} \left(q^{f(\gamma)n}\right)\right) \tag{2.33}$$

où f est définie par :

$$\begin{aligned} f(\gamma) \quad := \quad & (1-\omega)h_q\left(\tfrac{\gamma}{1-\omega}\right) + \omega h_q\left(\tfrac{\overline{\omega}-\gamma}{\omega}\right) + (\overline{\omega}-\gamma)h_q\left(2-\tfrac{\omega}{\overline{\omega}-\gamma}\right) \\ & +(2(\overline{\omega}-\gamma)-\omega)\log_q(q-2) - (3(\overline{\omega}-\gamma)-\omega)\log_q(q-1) \end{aligned} \tag{2.34}$$

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 $\left[\max(0, \overline{\omega}-\omega), \min\left(1-\omega, \overline{\omega}-\frac{\omega}{2}\right)\right]$ de l'équation :

$$(q-1)(1-\omega-x)(2(\overline{\omega}-x)-\omega)^2 = x(q-2)^2(\omega-\overline{\omega}+x)^2 \tag{2.35}$$

On peut alors utiliser la méthode de Cardan pour déterminer les solutions de l'équation (2.35). □

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 :

$$R_2(n, w, \overline{w}) = \binom{n-w}{\overline{w}-\frac{w}{2}}\binom{w}{\frac{w}{2}} = \tilde{\Theta}\left(2^{n\left((1-\omega)h_2\left(\frac{\overline{\omega}-\frac{\omega}{2}}{1-\omega}\right)+\omega\right)}\right) \tag{2.36}$$

On retrouve en fait exactement le théorème 2.6.2 où a vaut la seule valeur qui n'annule pas l'équation (2.27) ; 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 $\overline{w} \in \left[\!\left[\frac{w}{2}, n - \frac{w}{2}\right]\!\right]$.

2.6.2 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 2.8.

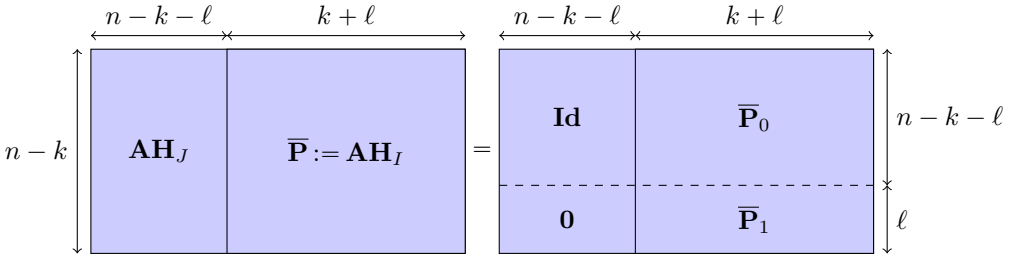


Figure 2.8 – Découpage de $[\mathbf{AH}_J|\mathbf{AH}_I]$ pour la méthode BJMM.

Dans la figure 2.8, $\mathbf{A}$ 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 $p_1 \in \left[\!\left[\frac{p}{2}, k+\ell\right]\!\right]$ un paramètre à optimiser. Toutes les p_1-représentations $(\mathbf{x}_1, \mathbf{x}_2)$ de $\mathbf{e}_I$ vérifient les deux équations suivantes :

$$\overline{\mathbf{s}}_L - \overline{\mathbf{P}}_1\mathbf{x}_1^\top = \overline{\mathbf{P}}_1\mathbf{x}_2^\top \tag{2.37}$$

$$\Delta\left(\overline{\mathbf{s}}_{\overline{L}} - \overline{\mathbf{P}}_0\mathbf{x}_1^\top, \overline{\mathbf{P}}_0\mathbf{x}_2^\top\right) = w - p \tag{2.38}$$

Soit $\mathscr{L}^*$, l'ensemble des couples de vecteurs de poids p_1 vérifiant l'équation (2.37) :

$$\mathscr{L}^* := \left\{(\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{F}_q^{k+\ell} \times \mathbb{F}_q^{k+\ell} \text{ tel que } |\mathbf{x}_1| = |\mathbf{x}_2| = p_1 \text{ et } \overline{\mathbf{s}}_L - \overline{\mathbf{P}}_1\mathbf{x}_1^\top = \overline{\mathbf{P}}_1\mathbf{x}_2^\top\right\} \tag{2.39}$$

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 :

$$\mathscr{L}_1 := \left\{ \bar{\mathbf{s}}_L - \overline{\mathbf{P}}_1 \mathbf{x}_1^\top \ : \ \mathbf{x}_1 \in \mathbb{F}_q^{k+\ell} \text{ et } |\mathbf{x}_1| = p_1 \text{ et } \left(\bar{\mathbf{s}}_L - \overline{\mathbf{P}}_1 \mathbf{x}_1^\top\right)_{[r_1]} = \mathbf{t}_1 \right\} \quad (2.40)$$

$$\mathscr{L}_2 := \left\{ \overline{\mathbf{P}}_1 \mathbf{x}_2^\top \ : \ \mathbf{x}_2 \in \mathbb{F}_q^{k+\ell} \text{ et } |\mathbf{x}_2| = p_1 \text{ et } \left(\overline{\mathbf{P}}_1 \mathbf{x}_2^\top\right)_{[\mathbf{r_1}]} = \mathbf{t}_1 \right\} \quad (2.41)$$

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 :

$$S_1 := \frac{\binom{k+\ell}{p_1}(q-1)^{p_1}}{q^{r_1}} = \frac{\#\mathscr{L}^*}{R_q(k+\ell,p,p_1)} \quad (2.42)$$

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

$$T_1 := O\left(S_1 + \frac{S_1^2}{q^{\ell - r_1}}\right) \quad (2.43)$$

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 (2.38).

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 :

$$\mathscr{L}_{2j}^{(0)} = \left\{ \mathbf{x} \in \mathbb{F}_q^n \ : \ \mathbf{x}_J = \overline{\mathbf{P}}_0 \mathbf{x}_I^\top \ , \ |\mathbf{x}_{I_1}| = p^{(0)} \text{ et } |\mathbf{x}_{I_2}| = 0 \right\} \quad \forall j \in [\![1, 2^m]\!] \quad (2.44)$$

$$\mathscr{L}_{2j-1}^{(0)} = \left\{ \mathbf{x} \in \mathbb{F}_q^n \ : \ \mathbf{x}_J = \overline{\mathbf{P}}_0 \mathbf{x}_I^\top \ , \ |\mathbf{x}_{I_1}| = 0 \text{ et } |\mathbf{x}_{I_2}| = p^{(0)} \right\} \quad \forall j \in [\![1, 2^m]\!] \quad (2.45)$$

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

$$\mathscr{L}_j^{(i)} := \left\{ \mathbf{x} \in \mathbb{F}_q^n \ : \ \mathbf{x}_I = \mathbf{x}_1 + \mathbf{x}_2 \ , \ \mathbf{x}_1 \in \mathscr{L}_1^{(i-1)} \ , \ \mathbf{x}_2 \in \mathscr{L}_2^{(i-1)} \ , \ |\mathbf{x}_I| = p^{(i)} \text{ et } \left(\overline{\mathbf{P}}_1 \mathbf{x}_I^\top\right)_{[r_i]} = \mathbf{t}_j^{(i)} \right\} \quad (2.46)$$

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 $q = 2$:

$$\frac{p^{(i+1)}}{2} \leqslant p^{(i)} \leqslant k + \ell - \frac{p^{(i+1)}}{2} \qquad \text{pour tout } i \in [\![1, m-1]\!] \tag{2.47}$$

– si $q > 2$:

$$\frac{p^{(i+1)}}{2} \leqslant p^{(i)} \leqslant k + \ell \qquad \text{pour tout } i \in [\![1, m-1]\!] \tag{2.48}$$

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 :

$$\begin{aligned} \overline{\mathscr{L}}_j^{(i)} \quad := \quad \Big\{ \mathbf{x} \in \mathbb{F}_q^n \ : \ & \mathbf{x}_I = \mathbf{x}_1 + \mathbf{x}_2 \ , \\ & \mathbf{x}_1 \in \mathscr{L}_1^{(i-1)} \ , \ \mathbf{x}_2 \in \mathscr{L}_2^{(i-1)} \text{ et } \left(\overline{\mathbf{P}}_1 \mathbf{x}_I^\top\right)_{[r_i]} = \mathbf{t}_j^{(i)} \Big\} \end{aligned} \tag{2.49}$$

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 2.5.2) 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 :

$$\begin{aligned} \overline{\mathscr{L}}_1^{(m)} \quad := \quad \Big\{ \mathbf{x} \in \mathbb{F}_q^n \ : \ & \mathbf{x}_I = \mathbf{x}_1 + \mathbf{x}_2 \ , \ \mathbf{x}_1 \in \mathscr{L}_1^{(m-1)} \ , \\ & \mathbf{x}_2 \in \mathscr{L}_2^{(m-1)} \ , \ \text{et } \left|\overline{\mathbf{P}}_0 \mathbf{x}_I^\top - \overline{\mathbf{s}}_{\overline{L}}\right| = w - p \Big\} \end{aligned} \tag{2.50}$$

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

$$\mathscr{L}_1^{(m)} := \left\{ \mathbf{x} \in \overline{\mathscr{L}}_1^{(m)} \ : \ |\mathbf{x}_I| = p \right\} \tag{2.51}$$

Remarque 2.6.1. 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 2.6.1.

Algorithme 2.6.1 : 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 (2.48) (ou (2.47)).
Sortie : $\mathbf{e} \in \mathbb{F}_q^n$ tel que $\mathbf{He}^\top = \mathbf{s}^\top$ et $|\mathbf{e}| = w$.

calculer $r_1, \cdots, r_{m-1} := \ell$ tels que $\forall i \in [\![1, m-1]\!]$, $r_i = \left\lceil \log_q(R_q(k+\ell, p^{(i+1)}, p^{(i)})) \right\rceil$;
répéter
  choisir un ensemble $I \subseteq [\![1, n]\!]$ tel que $|I| = k + \ell$;
  $J \leftarrow [\![1, n]\!] \setminus I$ calculer $\mathbf{A} \in \mathbb{F}_q^{(n-k)\times(n-k)}$ telle que $[\mathbf{A}|\mathbf{H}_J] = \left[\frac{\mathbf{Id}}{\mathbf{0}}\right]$; `/* si` $\mathbf{H}_J$ `est de rang` $< n-k-\ell$`, revenir à l'étape 3 */`
  $\overline{\mathbf{s}} \leftarrow \mathbf{As}^\top$ $\overline{\mathbf{P}} \leftarrow \mathbf{AH}_I$;
  produire la liste $\mathscr{L}_1^{(m)}$ comme décrit par l'équation (2.51) ;
tant que $\mathscr{L}_1^{(m)} \neq \emptyset$;
retourner $\mathbf{e} \in \mathscr{L}_1^{(m)}$;

2.6.3 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 :

$$\mathbb{P}_{\text{succ}} := \frac{\binom{k+\ell}{p}\binom{n-k-\ell}{w-p}}{\binom{n}{w}} \tag{2.52}$$

Il nous faudra donc répéter en moyenne $\frac{1}{\mathbb{P}_{\text{succ}}}$ fois la procédure de l'algorithme 2.6.1 pour que celui-ci s'arrête.

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

$$S^{(0)} := \binom{\frac{k+\ell}{2}}{p^{(0)}} (q-1)^{p^{(0)}} \tag{2.53}$$

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

$$T^{(0)} = S^{(0)} \tag{2.54}$$

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

$$S^{(i)} := \frac{\binom{k+\ell}{p^{(i)}}(q-1)^{p^{(i)}}}{q^{r_i}} \tag{2.55}$$

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

$$T^{(i)} := O\left(S^{(i-1)} + \frac{\left(S^{(i-1)}\right)^2}{q^{r_i - r_{i-1}}}\right) \tag{2.56}$$

(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 5 section 5.5 et dans les chapitres 5 à 9, 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 :

$$T^{(m)} := \beta(q, n-k-\ell, S^{(m-1)}, w-p) \tag{2.57}$$

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

$$T_{\text{BJMM}} = \frac{1}{\mathbb{P}_{\text{succ}}} \sum_{i=0}^{m} 2^{m-i} T^{(i)} = O\left(\frac{\binom{n}{w} \cdot \max\limits_{i \in [\![0,m]\!]} \left(T^{(i)} \right)}{\binom{k+\ell}{p}\binom{n-k-\ell}{w-p}} \right) \tag{2.58}$$

et la complexité mémoire est :

$$S_{\text{BJMM}} = \sum_{i=0}^{m-1} 2^{m-i} S^{(i)} = O\left(\max_{i \in [\![0,m-1]\!]} \left(S^{(i)} \right) \right) \tag{2.59}$$

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.

2.7 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.

2.7.1 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 :

$$\overline{\mathbf{P}} := \mathbf{H}_J^{-1}\mathbf{H}_I \in \mathbb{F}_q^{(n-k)\times k} \tag{2.60}$$

$$\overline{\mathbf{s}} := \mathbf{H}_J^{-1}\mathbf{s}^\top \in \mathbb{F}_q^{n-k} \tag{2.61}$$

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 :

$$|\mathbf{e}_I| = p \tag{2.62}$$
$$\left|\overline{\mathbf{s}} - \overline{\mathbf{P}}\mathbf{e}_I^\top\right| = w - p \tag{2.63}$$

Dans la sous-section 2.5.2, 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 2.6.1) 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 :

$$\sum_{j=1}^m r_j = n - k \tag{2.64}$$
$$\sum_{i=1}^m w_i^{(m)} = w - p \tag{2.65}$$
$$p^{(1)} = 2p^{(0)} \tag{2.66}$$
$$\frac{p^{(\ell)}}{2} \leqslant p^{(\ell-1)} \leqslant k \qquad \text{pour tout } \ell \in [\![2, m]\!] \tag{2.67}$$
$$0 \leqslant p^{(m)} = p \leqslant k \tag{2.68}$$
$$\frac{w_i^{(\ell)}}{2} \leqslant w_i^{(\ell-1)} \leqslant r_i \qquad \text{pour tout } \ell \in [\![2, m]\!] \text{ et } i \in [\![1, \ell - 1]\!] \tag{2.69}$$
$$0 \leqslant w_i^{(m)} \leqslant r_i \qquad \text{pour tout } i \in [\![1, m]\!] \tag{2.70}$$

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 :

$$\begin{aligned} |\mathbf{x}_I| &= p^{(\ell)} && (2.71)\\ |\mathbf{x}_{J_i}| &= w_i^{(\ell)} \quad \text{pour } i \in [\![1,\ell]\!] && (2.72)\\ \mathbf{x}_J^\top &= \overline{\mathbf{P}}\mathbf{x}_I^\top && (2.73) \end{aligned}$$

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

$$\begin{aligned} |\mathbf{x}_I| &= p^{(\ell)} && (2.74)\\ |\mathbf{x}_{J_i}| &= w_i^{(\ell)} \quad \text{pour } i \in [\![1,\ell]\!] && (2.75)\\ \mathbf{x}_J^\top &= \overline{\mathbf{s}}^\top - \overline{\mathbf{P}}\mathbf{x}_I^\top && (2.76) \end{aligned}$$

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 :

$$\begin{aligned} |\mathbf{e}| &= |\mathbf{e}_I| + |\mathbf{e}_J| \\ &= p^{(m)} + \sum_{i=1}^{m} w_i^{(m)} \ \text{(d'après les équations (2.74) et (2.75))} \\ &= p + w - p \ \text{(d'après les équations (2.68) et (2.65))} \\ &= w \end{aligned}$$

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 (2.73) :

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

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 :

$$\begin{aligned} \mathscr{L}_1^{(0)} &= \left\{ \mathbf{x} \in \mathbb{F}_q^n \ : \ \mathbf{x}_J^\top = \overline{\mathbf{s}} - \overline{\mathbf{P}}\mathbf{x}_I^\top \ , \ |\mathbf{x}_{I_1}| = p^{(0)} \text{ et } |\mathbf{x}_{I_2}| = 0 \right\} && (2.77)\\ \mathscr{L}_{2j}^{(0)} &= \left\{ \mathbf{x} \in \mathbb{F}_q^n \ : \ \mathbf{x}_J^\top = \overline{\mathbf{P}}\mathbf{x}_I^\top \ , \ |\mathbf{x}_{I_1}| = 0 \text{ et } |\mathbf{x}_{I_2}| = p^{(0)} \right\} \quad \forall j \in [\![1, 2^m]\!] && (2.78)\\ \mathscr{L}_{2j-1}^{(0)} &= \left\{ \mathbf{x} \in \mathbb{F}_q^n \ : \ \mathbf{x}_J^\top = \overline{\mathbf{P}}\mathbf{x}_I^\top \ , \ |\mathbf{x}_{I_1}| = p^{(0)} \text{ et } |\mathbf{x}_{I_2}| = 0 \right\} \quad \forall j \in [\![2, 2^m]\!] && (2.79) \end{aligned}$$

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 2.7.1 :

Algorithme 2.7.1 : 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 (2.74), (2.75) et (2.76) si` $j = 1$ `et (2.71), (2.72) et (2.73) 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 5 à 9 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 2.7.2.

Algorithme 2.7.2 : 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 $|I| = k$;
  $J = J_1 \cup \cdots \cup J_m \leftarrow [\![1, n]\!] \backslash I$; `/* 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 2.9 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.

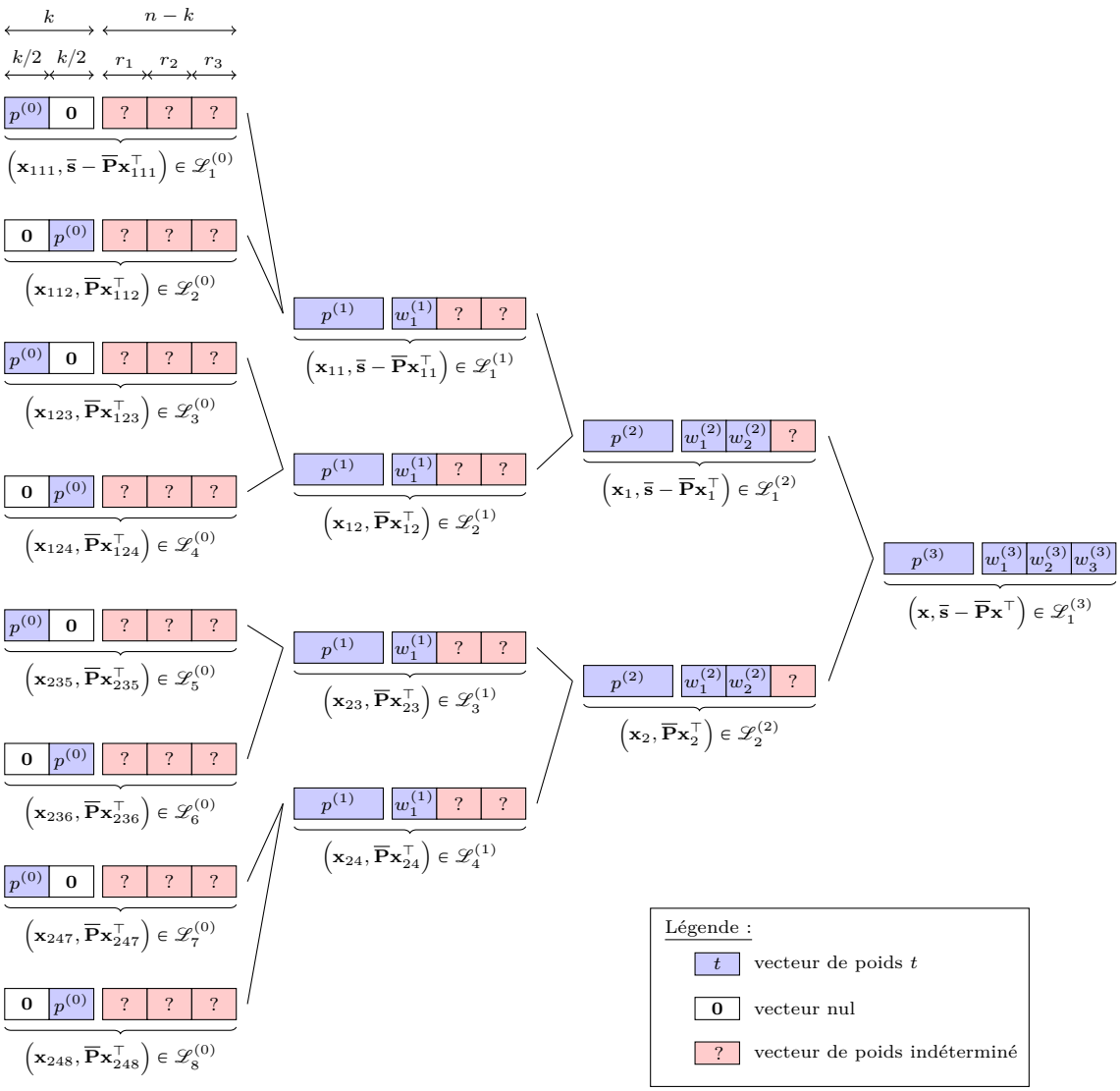


Figure 2.9 – Description de l'algorithme Both-May pour $m = 3$

Remarque 2.7.1. La version de May et Ozerov de l'algorithme de Stern que nous avons rappelée dans le pseudo-code 2.5.2 de la sous-section 2.5.2 est en fait l'instance de l'algorithme de Both et May où $m = 1$.

2.7.2 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 :

$$\begin{array}{cccccccccc} & & r_1 & ; & \cdots & \cdots & ; & r_m & ; \\ p^{(m)} & ; & w_1^{(m)} & ; & \cdots & \cdots & ; & w_m^{(m)} & ; \\ & & & & \vdots & & & & \\ p^{(\ell)} & ; & w_1^{(\ell)} & ; & \cdots & ; & w_\ell^{(\ell)} & ; & \\ & & & & \vdots & & & & \\ p^{(2)} & ; & w_1^{(2)} & ; & w_2^{(2)} & ; & & & \\ p^{(1)} & ; & w_1^{(1)} & ; & & & & & \\ p^{(0)} & . & & & & & & & \end{array} \tag{2.80}$$

Ceux-ci sont contraints par les équations (2.64), (2.65), (2.66), (2.67), (2.68), (2.69) et (2.70). Nous ajoutons une autre contrainte inspirée par le *correctness lemma* [BM18, Lemma 2] de Both et May :

Lemme 2.7.1 (Correctness)**.** *Soit* $\mathbf{e} \in \mathbb{F}_q^n$ *une solution du problème de décodage par syndrome 2.3.2 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 2.6.1). Si les paramètres de l'algorithme Both-May satisfont :*

$$E(\ell) := R_q\left(k, p^{(\ell)}, p^{(\ell-1)}\right) \cdot \prod_{i=1}^{\ell-1} \frac{R_q\left(r_i, w_i^{(\ell)}, w_i^{(\ell-1)}\right)}{q^{r_i}} \geqslant 1 \qquad \text{pour tout } \ell \in [\![2, m]\!] \tag{2.81}$$

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

Démonstration du lemme 2.7.1.

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 2.7.1 par les contraintes d'égalité suivantes :

$$E(\ell) = 1 \qquad \text{pour tout } \ell \in [\![2, m]\!] \tag{2.82}$$

Finalement, les paramètres de l'algorithme de Both et May (cf. équation (2.80)) sont soumis à de nombreuses contraintes d'égalité toutes données par les équations (2.82), (2.64), (2.65) et (2.66) (attention, l'équation (2.68) 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 (2.67), (2.69) et (2.70) 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* 2.7.1.

2.7.3 Complexité de l'algorithme

D'après le lemme 2.7.1, en choisissant nos paramètres de telle sorte que l'équation (2.81) (ou même (2.82)) 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 :

$$|\mathbf{e}_I| = p^{(m)} \tag{2.83}$$

$$|\mathbf{e}_{J_i}| = w_i^{(m)} \quad \text{pour tout } i \in [\![1, m]\!] \tag{2.84}$$

$$\tag{2.85}$$

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 :

$$\mathbb{P}_{\text{succ}} := \frac{\binom{k}{p} \cdot \prod_{i=1}^{m} \binom{r_i}{w_i^{(m)}}}{\binom{n}{w}} \tag{2.86}$$

$$= \tilde{O}\left(q^{k h_q\left(\frac{p}{k}\right) + \sum_{i=1}^{m} r_i h_q\left(\frac{w_i^{(m)}}{r_i}\right) - n h_q\left(\frac{w}{n}\right)} \right) \tag{2.87}$$

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 :

$$S^{(0)} = \binom{k/2}{p^{(0)}} (q-1)^{p^{(0)}} = \tilde{O}\left(q^{\frac{k}{2} h_q\left(\frac{2p^{(0)}}{k}\right)} \right) \tag{2.88}$$

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 $\ell \in [\![1, m]\!]$:

$$\begin{aligned} S^{(\ell)} \leqslant\ & \#\left\{\mathbf{x} \in \mathbb{F}_q^k \,|\, |\mathbf{x}| = p^{(\ell)}\right\} \cdot \mathbb{P}\left(|\mathbf{x}_0| = w_\ell^{(\ell)}\right) \\ & \cdot \prod_{i=1}^{\ell-1} \mathbb{P}\left(\Delta(\mathbf{x}_i, \mathbf{y}_i) = w_i^{(\ell)} \;\middle|\; |\mathbf{x}_i| = |\mathbf{y}_i| = w_i^{(\ell-1)}\right) \end{aligned} \tag{2.89}$$

Ce qui nous donne :

$$\begin{aligned} S^{(\ell)} \quad \leqslant \quad & \binom{k}{p^{(\ell)}}(q-1)^{p^{(\ell)}} \cdot \frac{\binom{r_\ell}{w_\ell^{(\ell)}}(q-1)^{w_\ell^{(\ell)}}}{q^{r_\ell}} \\ & \cdot \prod_{i=1}^{\ell-1} \frac{\binom{r_i}{w_i^{(\ell)}}(q-1)^{w_i^{(\ell)}} \cdot R\left(r_i, w_i^{(\ell)}, w_i^{(\ell-1)}\right)}{\left(\binom{r_i}{w_i^{(\ell-1)}}(q-1)^{w_i^{(\ell-1)}}\right)^2} \end{aligned} \tag{2.90}$$

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 $\ell \in [\![1, m]\!]$:

$$S^{(\ell)} = \tilde{O}(q^\gamma) \tag{2.91}$$

avec :

$$\begin{aligned} \gamma \quad := \quad & k h_q\left(\tfrac{p^{(\ell)}}{k}\right) - r_\ell\left(1 - h_q\left(\tfrac{w_\ell^{(\ell)}}{r_\ell}\right)\right) \\ & + \sum_{i=1}^{\ell-1} r_i\left(h_q\left(\tfrac{w_i^{(\ell)}}{r_i}\right) - 2h_q\left(\tfrac{w_i^{(\ell-1)}}{r_i}\right)\right) + \log_q\left(R\left(r_i, w_i^{(\ell)}, w_i^{(\ell-1)}\right)\right) \end{aligned} \tag{2.92}$$

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 $\ell \in [\![1, m]\!]$:

$$T^{(\ell)} = \beta\left(q, r_\ell, S^{(\ell-1)}, w_\ell^{(\ell)}\right) \tag{2.93}$$

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

$$T_{\mathrm{BM}} = \tilde{O}\left(\frac{1}{\mathbb{P}_{\mathrm{succ}}} \cdot \sum_{\ell=1}^{m} 2^{m-\ell} T^{(\ell)}\right) \tag{2.94}$$

$$= \tilde{O}\left(\frac{1}{\mathbb{P}_{\mathrm{succ}}} \cdot \max_{\ell \in [\![1,m]\!]}\left(T^{(\ell)}\right)\right) \tag{2.95}$$

Nous avons aussi sa complexité en mémoire qui est :

$$S_{\mathrm{BM}} = O\left(\sum_{\ell=0}^{m} 2^{m-\ell} S^{(\ell)}\right) \tag{2.96}$$

$$= O\left(\max_{\ell \in [\![0,m]\!]}\left(S^{(\ell)}\right)\right) \tag{2.97}$$

2.8 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 2.5.2 et 2.6.1. Nous avons aussi présenté une version généralisée aux espaces de Hamming non-binaires de l'algorithme de [BM18] dans le pseudo-code 2.7.2. 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 2.8.1, 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 2.8.2 nous évaluons la complexité du décodage de Both et May que nous avons généralisé aux espaces non-binaires dans le pseudo-code 2.7.2. 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.

2.8.1 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 2.5.2. 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 2.5.3. Cette complexité est intrinsequement liée à la méthode de recherche de presque-collisions utilisée. La méthode LSH des projections (cf. section 5.4.2) 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 1.2) 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 8.1.1 du chapitre 8 nous permet finalement d'avoir les meilleures complexités. Tous ces résultats sont résumés dans le tableau 2.3. Dans ce tableau, R est le rendement permettant d'obtenir la pire complexité pour la méthode de décodage considérée.

q	[Hir16]		algo. 2.5.2 et 5.3.1 (projections)		algo. 2.5.2 et 6.2.1 (codes)		algo. 2.5.2 et 8.1.1 (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 2.3 – Complexités de l'algorithme Stern-May-Ozerov 2.5.2 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 2.2 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 2.5.2, 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 [DA19, 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 2.10 permet de comparer ces méthodes avec le décodage Stern-May-Ozerov instancié avec notre algorithme 8.1.2 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.

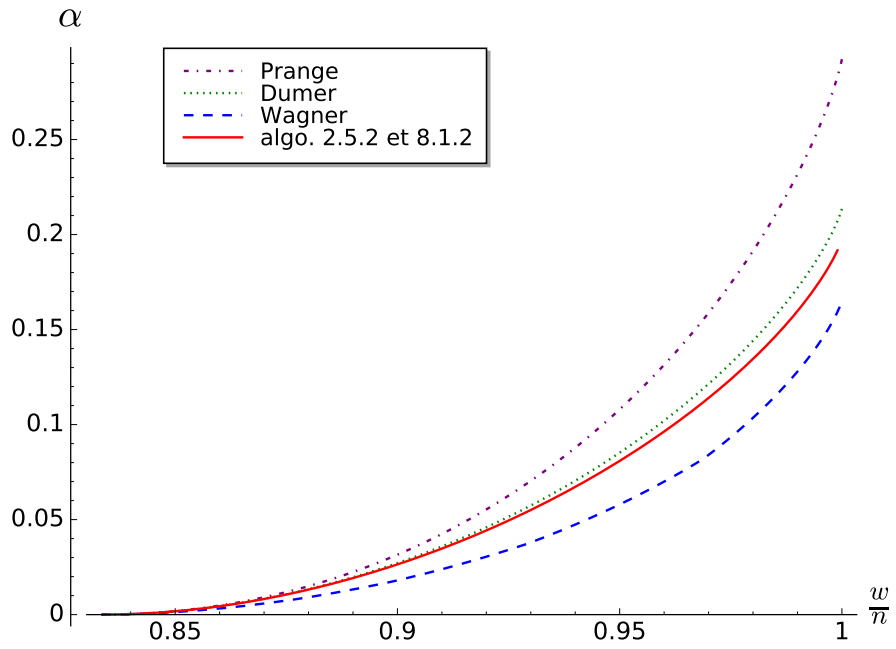


Figure 2.10 – Exposant de la complexité asymptotique pour le décodage en en grandes distances avec Stern-May-Ozerov et notre méthode de recherche de couples éloignés ($q = 3$ et $\frac{k}{n} = \frac{1}{2}$).

2.8.2 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 8.1.1 du chapitre 8. 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 2.5 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 9.2.1 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 2.7, nous avons généralisé la méthode de décodage générique de Both et May aux espaces de Hamming non-binaires ; obtenant ainsi le pseudo-code 2.7.2. Nous combinons cet algorithme avec notre recherche de presque-collisions hybride 8.1.1 du chapitre 8. Pour simplifier la phase d'optimisation des paramètres de l'algorithme 2.7.2, 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 2.4. 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 2.8.1. 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 2.6.2 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. 2.7.2 et 8.1.1	
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 2.4 – 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* 2.7.1 ; 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 8.1.1 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 2.5 – Paramètres de l'algorithme 2.7.2.

Le tableau 2.5 donne les paramètres de l'algorithme 2.7.2 qui nous ont permis d'obtenir les complexités données dans le tableau 2.4.

Remarque 2.8.2. 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 2.8.3. 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.

2.9 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 2.9.1 (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 :*

$$(\mathbf{x}, \widetilde{c}) := (\mathbf{x}, \langle \mathbf{x}, \mathbf{s} \rangle + e) \tag{2.98}$$

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 :

$$\mathbf{G} := \left[\begin{array}{c|c|c|c} \mathbf{x}_1^\top & \mathbf{x}_2^\top & \cdots & \mathbf{x}_n^\top \end{array} \right] \tag{2.99}$$

et :

$$\widetilde{\mathbf{c}} := (\widetilde{c}_1, \widetilde{c}_2, \cdots, \widetilde{c}_n) \tag{2.100}$$

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 :

$$\mathbf{G} = \left[\begin{array}{c} \mathbf{G}' \\ \hline \mathbf{0}_{k_1 \times n} \end{array} \right] \tag{2.101}$$

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 2.9.1. 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 $\mathbf{G'}$.

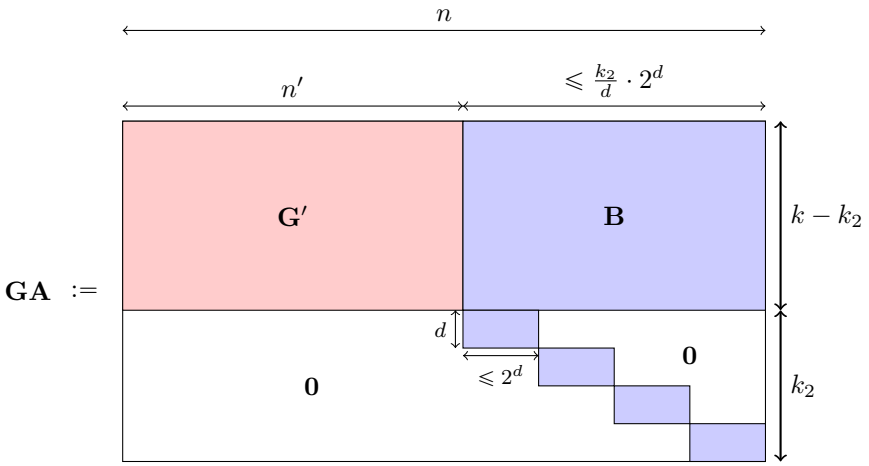
Le problème de cette deuxième astuce est que le mot $\tilde{\mathbf{c}}'$ est beaucoup plus bruité que le mots $\tilde{\mathbf{c}}$. On a donc réduit la dimension du problème LPN au prix d'une augmentation importante du taux d'erreur. Il est possible de quantifier exactement cette augmentation grâce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

$$\tau' := \mathbb{P}\left(e' = 1\right) = \frac{1 - (1 - 2\tau)^{k_2/2}}{2} \tag{2.102}$$

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 2.9.2. 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 $\mathbf{G}'$.

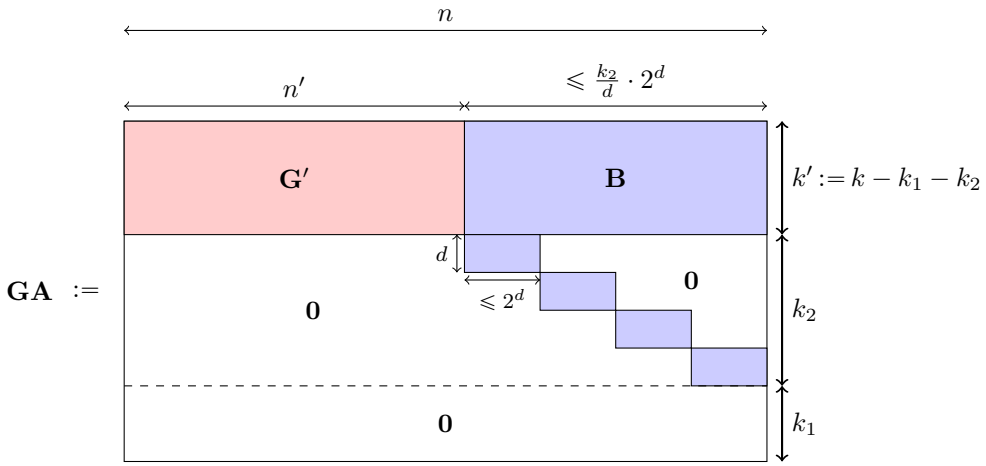
Notre objectif de limiter l'augmentation du taux d'erreur dans le vecteur $\widetilde{\mathbf{c}}\mathbf{A}$ et donc dans le vecteur $\widetilde{\mathbf{c}}'$ est bien rempli puisque cette fois-ci, le *piling-up lemma* nous dit que :

$$\tau' := \mathbb{P}\left(e' = 1\right) = \frac{1-(1-2\tau)^{k_2/d}}{2} \tag{2.103}$$

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 3

La reconnaissance de codes correcteurs d'erreurs

Le problème de reconnaissance de codes correcteurs d'erreurs se pose lorsqu'un ensemble de mots bruités provenant d'un code $\mathcal{C}$ inconnu a été observé et que l'on souhaite déterminer le code correcteur d'erreurs utilisé à l'émission. Ce problème trouve des applications dans différents domaines tels que la reconstruction de communications dans un contexte non-coopératif, la construction de récepteurs intelligents dans le domaine des radios cognitives [MGB12], la modélisation de l'ADN comme un code [Ros06], dans la cryptanalyse de crypto-systèmes à clé privée combinant un chiffrement avec de la correction d'erreurs comme dans [RN86, ST87, BY00, SAEA09, EDG14, MEP17]. Dans ces derniers exemples, retrouver la clé secrète est en fait un problème de reconnaissance de code (même si cette connexion n'est jamais mentionnée explicitement dans les papiers cités juste avant).

Nous nous intéressons essentiellement à la reconnaissance de codes linéaires de longueur connue. La version décisionnelle de ce problème peut être énoncée en ces termes : "Est-il possible de modifier au plus w bits de chacun des mots reçus de sorte que les mots modifiés génèrent un espace vectoriel de dimension au plus k ?". Ce problème a été montré NP-complet par Valembois en 2001 [Val01]. Il peut aussi être reformulé comme le *matrix rigidity problem* de [Val77] qui a lui-aussi été prouvé NP-complet dans un modèle plus général que le notre [Des07].

En pratique, nous émettons une hypothèse supplémentaire, notamment sur la famille à laquelle appartient le code. Par exemple, $\mathcal{C}$ peut etre un code LDPC. Dans ce cas, le problème devient essentiellement polynomial [CT08, CF09b] avec un degré de polynôme qui est relié au poids des équations de parité du code LDPC. Le problème de reconnaissance de code a aussi été abordé pour d'autres codes tels que les codes cycliques [Cha09, LYSL10, WYY10, ZHSY13, YVK14], les codes convolutifs [Ric95, Fil97, Fil00, LSLZ04, DH07, HZ07, CF09a, CS09, Mar09, MGB12, BT14] ou encore les turbo-codes [Bar05, Bar07, CS10, CFT10, NAF11, TTS14]. De façon générale, pour les familles de codes possédant une structure algébrique forte telles que les codes BCH, les codes de Reed-Solomon ou les codes de Reed-Muller, le problème de reconnaissance de codes est résolu. En revanche, pour de nombreuses autres familles de codes, de nombreux progrès reste à faire.

Dans ce chapitre, les codes que nous cherchons à reconstruire sont des codes linéaires ayant la particularité de posséder des équations de parité creuses (ou des codes construits à partir de tels codes) ; c'est-à-dire de poids de Hamming faible. Parmi ces codes, on peut citer bien évidemment les codes LDPC mais aussi les codes convolutifs, les turbo-codes, les codes raptors, les codes fontaines ou encore les codes polaires.

Plus formellement, un observateur reçoit M mots bruités $\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_M$ provenant

d'un code en bloc linéaire inconnu $\mathcal{C}[n,k]$ qu'il place dans une matrice $\widetilde{\mathbf{G}}$ de la manière suivante :

$$\widetilde{\mathbf{G}} := \left[\begin{array}{c} \mathbf{y}_1 \\ \hline \vdots \\ \hline \mathbf{y}_M \end{array}\right] \tag{3.1}$$

Il est supposé que le canal de transmission est un canal binaire symétrique de probabilité d'erreur τ (noté BSC(τ)). L'objectif de l'observateur est de retrouver des équations de parité creuses du code $\mathcal{C}$; typiquement, des équations de parité de poids fixé égal à w.

Dans la section 3.1, nous détaillons les travaux de Tixier et Tillich pour reconstruire des codes LDPC et notamment des codes LDPC structurés [Tix15]. Dans la section 3.2, nous continuons notre état de l'art en présentant deux méthodes classiques pour retrouver des équations de parités creuses dans un code : la méthode de Cluzeau-Finiasz [CF09b] et celle de Sicot-Houcke-Barbier [BSH06, SHB09]. Dans la section 3.3, nous présentons notre méthode de reconnaissance de code qui interpole entre les deux méthodes précédentes et améliore les outils connus à ce jour pour trouver des équations de parité creuses dans un code. Ce travail est l'objet de l'article que nous avons présenté au 10$^{\text{ème}}$ *International Workshop on Coding and Cryptography* en 2017 à St Petersbourg [CT17]. Une version longue de ce papier a été publiée dans le journal *Designs, Codes and Cryptography* en 2019 [CT19b]. Dans la dernière section de ce chapitre nous proposons une généralisation de notre méthode utilisant l'algorithme BKW [BKW03].

3.1 La méthode Tixier-Tillich

La première méthode de reconnaissance de codes que nous décrivons dans ce chapitre s'inspire directement des décodages ISD ; notamment de la méthode de Dumer [Dum91]. Elle a été imaginée par Tixier et Tillich et on en trouve une description dans [Tix15].

Les méthodes ISD présentées dans la section 2.4 recherchent l'unique vecteur d'erreur de poids $\leqslant w$ qui possède le même syndrome que le mot reçu. Ainsi, les décodages ISD peuvent être adaptés à la recherche de mots de petit poids d'un code. Pour cela, il suffit de supposer le syndrome nul et de fixer un seuil de poids w supérieur à la distance minimale du code.

Les équations de parité du code $\mathcal{C}$ sont aussi les mots de son dual $\mathcal{C}^\perp$ et une matrice de parité de $\mathcal{C}^\perp$ est aussi une matrice génératrice de $\mathcal{C}$. Ainsi, en travaillant sur le code dual plutôt que sur le code lui-même, les méthodes de décodage ISD permettent aussi de retrouver des équations de parité creuses de $\mathcal{C}$ à partir d'une matrice génératrice. Cependant, nous ne possédons pas une matrice génératrice de $\mathcal{C}$ mais seulement une matrice génératrice bruitée. Nous allons donc construire un code intermédiaire $\mathcal{C}'$ qui va nous permettre de prendre en compte les éventuelles erreurs.

Rappelons que les mots observés sont bruités par un canal binaire symétrique de probabilité d'erreur τ. Nous connaissons donc la probabilité qu'une équation de parité $\mathbf{h}$ de poids w satisfasse un mot de code bruité $\mathbf{y}$ (et donc une ligne de la matrice $\widetilde{\mathbf{G}}$). Cette probabilité est donnée par le *piling-up lemma* [Mat93] que nous rappelons dans le lemme 3.1.1.

Lemme 3.1.1 (*piling-up lemma*)**.** *Soit* $\mathbf{y}$*, un mot de* $\mathcal{C}$ *bruité par un canal binaire symétrique de probabilité d'erreur* τ*. Soit* $\mathbf{h}$*, une équation de parité de poids* w *de* $\mathcal{C}$*. La probabilité que* $\mathbf{y}$ *vérifie* $\mathbf{h}$ *est :*

$$\mathbb{P}\left(\mathbf{y} \times \mathbf{h}^\top = \mathbf{0} \;\middle|\; \mathbf{h} \in \mathcal{C}^\perp \text{ et } |\mathbf{h}| = w\right) = \frac{1 + (1-2\tau)^w}{2}$$

Démonstration du lemme 3.1.1.

Soit i, le nombre de bits erronés du mot $\mathbf{y}$ sur le support de $\mathbf{h}$. Pour que $\mathbf{y}$ vérifie l'équation de parité $\mathbf{h}$ de $\mathcal{C}$, il faut et il suffit que i soit pair :

$$\begin{aligned}\mathbb{P}\left(\mathbf{y}\times\mathbf{h}^\top=\mathbf{0}\mid\mathbf{h}\in\mathcal{C}^\perp \text{ et } |\mathbf{h}|=w\right) &= \sum_{\substack{i=0\\ i\ pair}}^{w}\binom{w}{i}\tau^i(1-\tau)^{w-i}\\ &= \frac{1+(1-2\tau)^w}{2}\end{aligned}$$

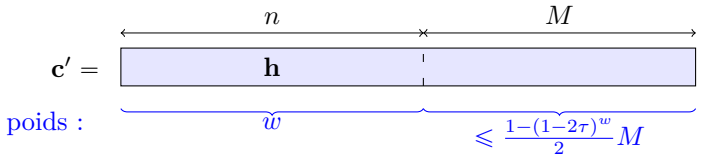
□

Remarque 3.1.1. Plus $\frac{1-(1-2\tau)^w}{2}M$ est proche de $\frac{M}{2}$, plus il sera difficile de distinguer une équation de parité de $\mathcal{C}$ d'un mot de poids w aléatoire. Cela peut se produire lorsque τ est proche de $\frac{1}{2}$ ou lorsque M est petit.

On définit $\mathcal{C}'$, le code linéaire $[n+M,n]$ de matrice génératrice $\mathbf{G}'$:

$$\mathbf{G}' := \left[\mathbf{Id}_n|\widetilde{\mathbf{G}}^\top\right] \tag{3.2}$$

On pose alors $\mathbf{c}' := \mathbf{h}\mathbf{G}'$ où $\mathbf{h}\in\mathbb{F}_2^n$ est une équation de parité de poids w du code $\mathcal{C}$. D'après le *piling-up lemma*, le poids de $\mathbf{c}'$ se répartit ainsi (avec une bonne probabilité) :

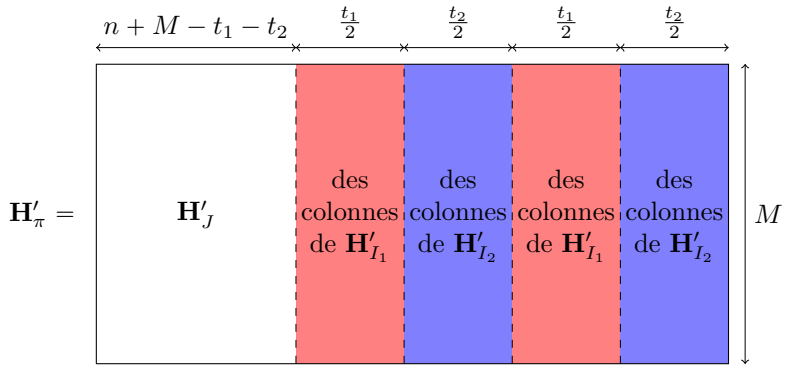


Pour rechercher ces mots de petits poids, nous pouvons utiliser les méthodes ISD. Celles-ci travaillent sur les matrices de parité mais la matrice génératrice $\mathbf{G}'$ du code $\mathcal{C}'$ étant systématique, nous pouvons facilement en construire une :

$$\mathbf{H}' := \left[\widetilde{\mathbf{G}}|\mathbf{Id}_M\right] \tag{3.3}$$

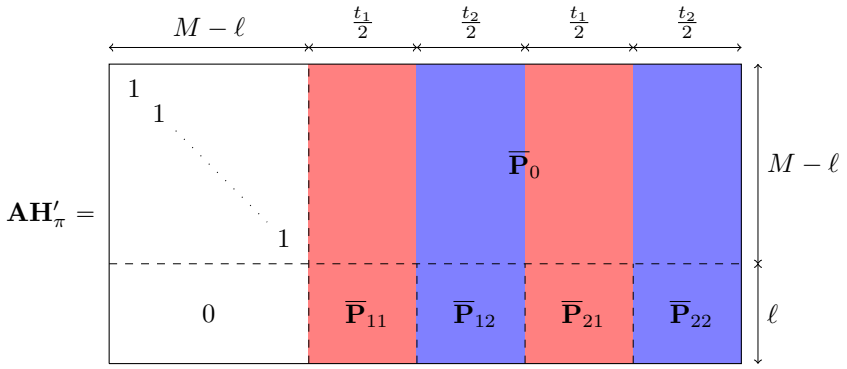
3.1.1 Une méthode inspirée du décodage ISD de Dumer

Tixier et Tillich suggèrent d'utiliser la méthode de Dumer avec toutefois quelques modifications. Ces modifications visent à prendre en compte la répartition de poids bien particulière des mots $\mathbf{c}'$ recherchés. Tout d'abord, un ensemble $I := I_1 \cup I_2$ contenant un ensemble d'information du code $\mathcal{C}'$ est choisi à chaque itération de l'algorithme. Nous notons $J := [\![1, n+M]\!]\setminus I$ l'ensemble d'indices complémentaire. Nous distinguons les positions I_1 provenant de la sous-matrice $\widetilde{\mathbf{G}}$ des autres positions I_2. Les tailles respectives t_1 et t_2 de I_1 et I_2 seront des paramètres à optimiser. Soit une permutation $\pi\in S_{n+M}$ telle que la matrice $\mathbf{H}'_\pi$ obtenue en permutant les colonnes de $\mathbf{H}'$ selon cette permutation soit de la forme :



où les colonnes en rouge sont les colonnes de $\mathbf{H}'$ indexées par I_1 et où les colonnes en bleu sont celles indexées par I_2.

Comme pour la méthode de Dumer, nous appliquons un pivot de Gauss partiel sur la matrice $\mathbf{H}'_\pi$ pour obtenir une matrice $\mathbf{AH}'_\pi$ que l'on découpe comme suit :



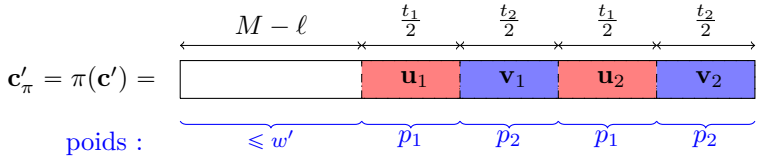
où $\ell := t_1 + t_2 - n$.

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

$$\mathscr{L}_1 := \left\{ \left(\overline{\mathbf{P}}_{11}\mathbf{u}_1^\top, \overline{\mathbf{P}}_{12}\mathbf{v}_1^\top\right) \ : \ \mathbf{u}_1 \in \mathbb{F}_2^{\frac{t_1}{2}} \ , \ \mathbf{v}_1 \in \mathbb{F}_2^{\frac{t_2}{2}} \ , \ |\mathbf{u}_1| = p_1 \text{ et } |\mathbf{v}_1| = p_2 \right\} \quad (3.4)$$

$$\mathscr{L}_2 := \left\{ \left(\overline{\mathbf{P}}_{21}\mathbf{u}_2^\top, \overline{\mathbf{P}}_{22}\mathbf{v}_2^\top\right) \ : \ \mathbf{u}_2 \in \mathbb{F}_2^{\frac{t_1}{2}} \ , \ \mathbf{v}_2 \in \mathbb{F}_2^{\frac{t_2}{2}} \ , \ |\mathbf{u}_2| = p_1 \text{ et } |\mathbf{v}_2| = p_2 \right\} \quad (3.5)$$

La fin de l'itération de la méthode est analogue à la méthode de Dumer et consiste essentiellement à trouver des collisions dans $\mathscr{L}_1 \times \mathscr{L}_2$. Ceci permet alors de trouver des vecteurs ayant la répartition de poids suivante :



où $w' := w + \frac{1-(1-2\tau)^w}{2} M - 2(p_1 + p_2)$.

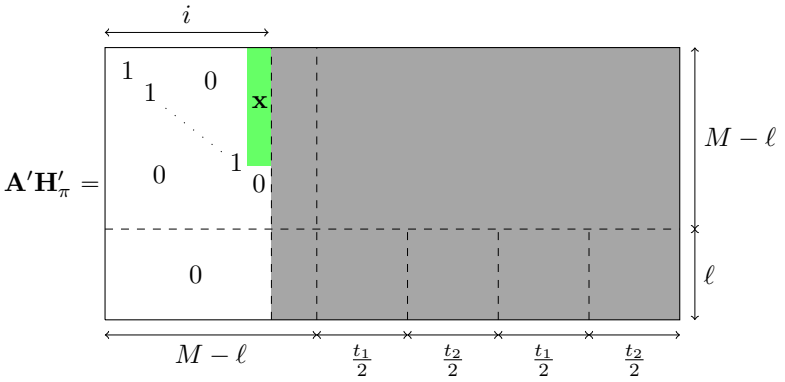
Après application de la permutation inverse, nous vérifions que les mots $\mathbf{c}'$ du code $\mathcal{C}'$ que nous obtenons sont bien de poids w sur les n premières positions. Si c'est le cas, alors les n premiers bits de $\mathbf{c}'$ forment une potentielle équation de parité de poids w du code $\mathcal{C}$.

3.1.2 À propos de l'élimination gaussienne partielle

Dans la méthode Tixier-Tillich, l'élimination gaussienne partielle est effectuée en opérant sur les lignes de la sous-matrice $\mathbf{H}'_J$ de $\mathbf{H}'$ composée des colonnes indexées par $J := [\![1, n+M]\!] \setminus I$. Supposons alors que $\mathbf{H}'_J$ ne soit pas de rang plein. Cela signifie que lors de l'élimination gaussienne, des colonnes sans pivot ont été détectées. Cela arrive par exemple systématiquement lorsque $\tau = 0$, $t_2 = M$ et $M - \ell > k$. En effet, en prenant $t_2 = M$ dans le cas non bruité, les $M - \ell$ premières colonnes de $\mathbf{H}'_\pi$ ne peuvent pas être de rang supérieur à k car elles forment une sous-matrice de la matrice génératrice du code $\mathcal{C}$ (en supposant que les lignes de $\mathbf{H}'$ sont linéairement indépendantes).

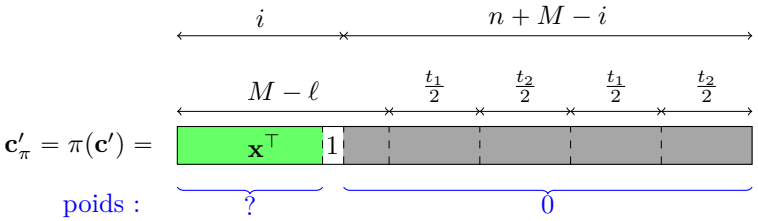
Lors de l'élimination gaussienne, si une colonne de $\mathbf{H}'_J$ ne possède pas de pivot, alors nous l'échangeons avec une colonne de $\mathbf{H}'_I$ qui contient un pivot. Dans ce cas, il faut modifier la permutation π en conséquence. Cependant, avant d'effectuer cette opération,

nous pouvons utiliser la colonne sans pivot pour déterminer une équation de parité de $\mathcal{C}$. En effet, supposons que l'algorithme du pivot de Gauss ne trouve pas de pivot dans la $i^{\text{ème}}$ colonne. La matrice que nous obtenons est alors de la forme suivante :



où $\mathbf{A}'$ est la matrice de passage obtenue après i pivots de Gauss.

La $i^{\text{ème}}$ colonne de $\mathbf{A}'\mathbf{H}'_\pi$ est donc composée d'un vecteur $\mathbf{x}$ de longueur $i-1$ (en vert sur la figure) suivi de zéros. Soit $\mathbf{c}'_\pi$, le vecteur de taille $n+M$ suivant :



Finalement, $\mathbf{c}'_\pi$ est un mot permuté par π du code $\mathcal{C}'$ car $\mathbf{A}'\mathbf{H}'_\pi {\mathbf{c}'_\pi}^\top = \mathbf{0}$ donc $\mathbf{H}'_\pi {\mathbf{c}'_\pi}^\top = \mathbf{0}$. On pose alors $\mathbf{c}' = \pi^{-1}(\mathbf{c}'_\pi)$ et on vérifie que les n premiers bits de $\mathbf{c}'$ forment une équation de parité $\mathbf{h}$ du code $\mathcal{C}$ (qui n'est pas nécessairement de poids w). Pour cela, il suffit de compter parmi les M mots bruités $\mathbf{y}_i$ du code $\mathcal{C}$, le nombre qui satisfont l'équation $\mathbf{h}$. Ce nombre doit être proche de celui que l'on obtient avec le *piling-up lemma* et il doit de ce fait se distinguer de $\frac{M}{2}$.

3.1.3 Complexité de la méthode Tixier-Tillich

Soit une équation de parité $\mathbf{h}$ du code $\mathcal{C}$ de poids w. La probabilité que l'algorithme Tixier-Tillich retourne $\mathbf{h}$ est la probabilité que le vecteur $\mathbf{c}' := \mathbf{h}\mathbf{G}'$ ait la bonne répartition de poids :

$$\mathbb{P}_{\text{succ}} = \frac{\binom{t_1/2}{p_1}^2 \binom{n-t_1}{w-2p_1}}{\binom{n}{w}} \cdot \binom{t_2/2}{p_2}^2 \left(\frac{1-(1-2\tau)^w}{2}\right)^{2p_2} \left(\frac{1+(1-2\tau)^w}{2}\right)^{t_2-2p_2} \tag{3.6}$$

De plus, le coût d'une itération C_{iter} est essentiellement le coût de la recherche de collisions :

$$C_{\text{iter}} = O\left(\binom{t_1/2}{p_1}\binom{t_2/2}{p_2} + \frac{\binom{t_1/2}{p_1}^2 \binom{t_2/2}{p_2}^2}{2^\ell}\right) \tag{3.7}$$

Finalement, la complexité en temps de la méthode Tixier-Tillich est :

$$T_{\text{Tixier-Tillich}} := \frac{C_{\text{iter}}}{\mathbb{P}_{\text{succ}}} \tag{3.8}$$

En pratique, une minimisation de cette complexité donnera souvent les paramètres optimaux $p_2 = 0$ et $t_2 = M$. L'algorithme Tixier-Tillich revient alors à effectuer la méthode de Dumer sur la matrice $\widetilde{\mathbf{G}}$ en supposant que l'équation de parité recherchée vérifie tous les mots bruités composant la matrice $\widetilde{\mathbf{G}}$. Il faudra donc renouveler la matrice $\widetilde{\mathbf{G}}$ à chaque itération en utilisant un nouvel ensemble de M mots bruités. En outre, il est alors possible d'utiliser une autre méthode ISD que celle de Dumer.

3.1.4 Accélération de la reconnaissance de codes LDPC

Les méthodes que nous avons présentées jusqu'ici permettent de trouver des équations de parité de poids faible dans dans un code linéaire. Celles-ci sont donc particulièrement adaptées à la reconnaissance de codes LDPC. En pratique, les codes LDPC possèdent souvent d'autres propriétés comme nous l'avons vu par exemple dans la section 1.3. Nous pouvons alors utiliser ces propriétés pour accélérer la reconnaissance des codes LDPC. Dans la thèse [Tix15], trois méthodes d'accélération sont présentées.

Correction partielle des mots bruités. Une première technique pour accélérer la reconnaissance d'un code LDPC consiste à corriger partiellement les mots bruités au fur et à mesure que des équations de parité sont trouvées. En effet, des algorithmes de décodage comme l'algorithme *sum-product* que nous avons vu dans la section 1.3 permettent de corriger partiellement un ensemble de mots bruités en n'utilisant qu'une partie des équations de parité. Cependant, moins nous possédons d'équations, plus le décodage aura des risques de diverger et donc de ne pas corriger certains mots. Il est donc important de fixer un nombre d'itérations maximal dans l'algorithme *sum-product*.

Nous savons d'expérience que les premières équations sont plus faciles à trouver ; c'est-à-dire que plus nous trouvons des équations de parité du code, plus il est difficile d'en trouver de nouvelles. Ce phénomène est le même que celui du problème du collectionneur de coupons [FS14]. Ainsi, il est judicieux de corriger les mots bruités au fur et à mesure que les équations sont trouvées pour ainsi faciliter la recherche de nouvelles équations. Il n'est pas nécessaire non plus d'effectuer une correction partielle dès qu'une nouvelle équation est trouvée... En effet, il sera souvent plus judicieux d'interrompre la reconnaissance avec une fréquence raisonnable.

D'autre part, il est important de corriger l'ensemble des mots bruités de départ à chaque interruption de la méthode de recherche d'équations de parité creuse et non l'ensemble des mots corrigés à l'interruption précédente. En effet, une correction peut détériorer certains mots bruités de façon irréversible, même avec une matrice de parité complète.

La figure 3.1 montre l'évolution de la probabilité qu'un bit soit erroné en fonction de la proportion d'équations de parité utilisées pour le décodage. Chaque courbe correspond à un taux d'erreur initial τ différent. Le code $\mathcal{C}_{2304}$ utilisé pour réaliser cette figure est un code LDPC régulier $[2304, 1152]$ possédant 768 équations de parité de poids 6 et 384 équations de poids 6 (ce code fait partie d'une plus large famille de codes LDPC normés).

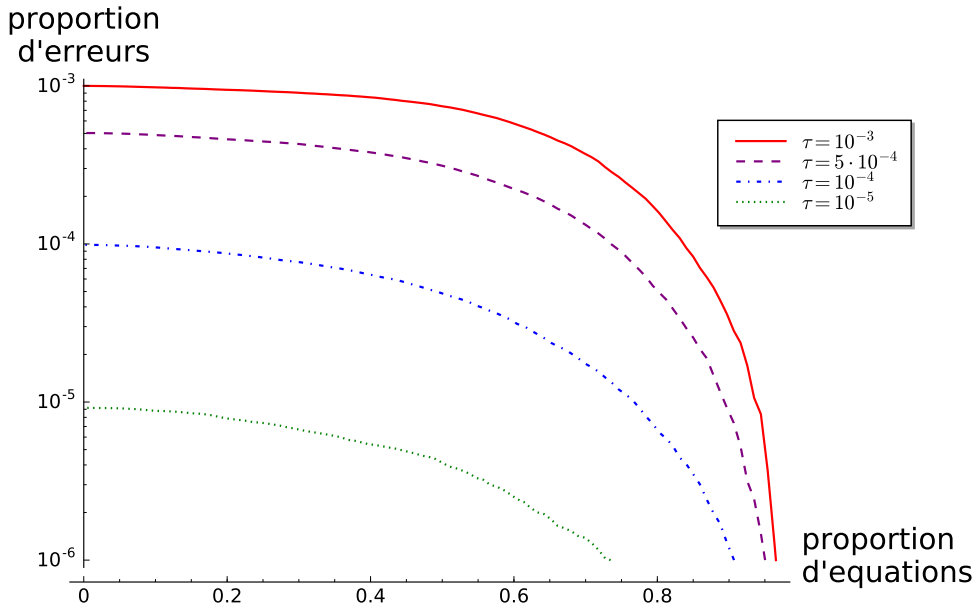


Figure 3.1 – Évolution de la probabilité d'erreur en fonction de la proportion d'équations de parité utilisées pour le décodage *sum-product* du code $\mathcal{C}_{2304}$.

Reconnaissance de codes LDPC structurés. Pour faciliter le codage et pour pouvoir les stocker plus facilement, les codes LDPC rencontrés en pratique sont très souvent structurés. En effet, dans de nombreuses normes, les matrices de parité des codes LDPC sont quasi-cycliques (cf. section 1.3) ; c'est-à-dire que la matrice de parité est découpée en blocs de matrices carrées de même taille Z et que chacune de ces sous-matrices est soit nulle, soit l'identité à rotation près des colonnes. Un exemple d'une telle matrice est donné dans la figure 3.2. Dans cette figure, les pixels blancs représentent des 0 tandis que les pixels noirs représentent des 1. Les pointillés rouges délimitent les blocs cycliques. En plus d'avoir une structure quasi-cyclique, la matrice de parité de la figure 3.2 a une structure convolutive (cf. section 1.3). Cela s'observe par la disposition particulière en forme de double diagonale des blocs des $n - k - Z$ dernières colonnes de la matrice de parité.

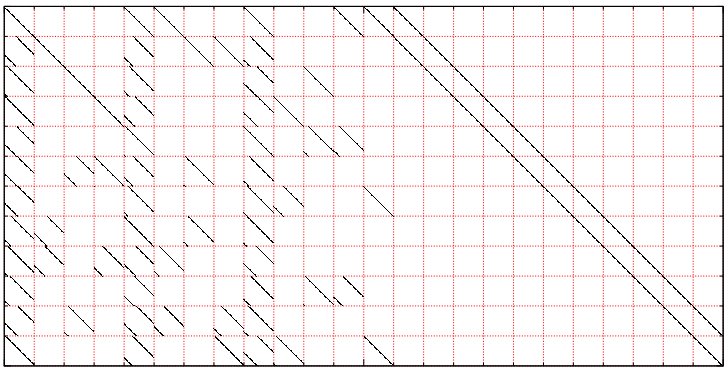


Figure 3.2 – Structure quasi-cyclique et convolutive de la matrice de parité d'un code LDPC de la norme 802.11n [IEE09].

Si un code est quasi-cyclique et que l'on connaît la taille des blocs circulants, alors il suffit de ne connaître qu'une seule équation par bloc pour définir l'ensemble des équations de parité de la matrice de parité. Rappelons qu'il est facile de vérifier qu'une équation est une équation de parité du code en observant le nombre de mots bruités qui la satisfont. Ainsi, dès lors que nous avons trouvé au moins une équation de parité de petit poids, il suffit généralement de tester exhaustivement toutes les tailles de bloc possibles.

Il est toutefois possible de déterminer la taille des blocs circulants de façon beaucoup plus efficace lorsque l'on connait un petit ensemble d'équations de parité de petit poids. En effet, soient $\mathbf{h}_1$ et $\mathbf{h}_2$, deux équations de parité distinctes de même poids w. Si ces deux équations appartiennent à un même bloc alors :

$$\exists Z \in [\![1, n]\!], \exists a \in [\![1, Z-1]\!], \forall i \in [\![1, w]\!], |\mathrm{supp}(\mathbf{h}_1)[i] - \mathrm{supp}(\mathbf{h}_2)[i]| \in \{a, Z-a\} \quad (3.9)$$

où $\mathrm{supp}(\mathbf{x})[i]$ est le $i^{\text{ème}}$ indice du support de $\mathbf{x}$. Un algorithme de mesure de la taille des blocs consiste alors à rechercher un couple d'équations de même poids parmi celles déjà trouvées qui vérifie l'assertion (3.9). Un tel couple est donc associé à des entiers Z et a où Z est potentiellement la taille des blocs circulants. Si après vérification, il s'avère que Z n'est pas la quantité recherchée, alors nous continuons à explorer les couples d'équations de parité de même poids. Notons qu'il n'est pas nécessaire de connaître beaucoup d'équations de parité pour appliquer cette méthode car celle-ci aboutira dès lors qu'il existe au moins un couple d'équations appartenant à un même bloc. Ainsi, en supposant que Z est la valeur recherchée, au plus $\frac{n}{Z} + 1$ équations suffisent ($\frac{n-1}{Z-1}$ en moyenne) pour trouver Z.

Enfin, en triant les équations selon l'indice de leur dernier bit à 1, l'algorithme devient sous-quadratique en le nombre d'équations de parité utilisés dès lors que le code possède une structure convolutive. En effet, pour tout équation $\mathbf{h}$, si l'équation qui succède à $\mathbf{h}$ dans la liste triée des équations connues n'appartient pas au même bloc que l'équation $\mathbf{h}$, alors aucune des équations suivantes n'est dans ce même bloc.

Étude de l'histogramme. Enfin, dans sa thèse, Tixier propose une troisième méthode d'accélération de la reconnaissance des codes LDPC qui est particulièrement pertinente pour des codes réguliers ou réguliers par bloc ; c'est-à-dire possédant une matrice de parité dont les colonnes sont de poids constant ou au moins de poids constant par bloc. C'est notamment le cas des codes LDPC quasi-cycliques. Ces poids peuvent etre déduits d'un ensemble incomplet d'équations de parité distinctes. En effet, pour chacun des indices de colonnes d'un même bloc, le nombre d'équations de parité faisant intervenir cette colonne – c'est-à-dire le poids de cette colonnes dans la matrice $\mathbf{H_p}$ composée des équations distinctes déjà trouvées – sera inférieur ou égal au poids de chacune des colonnes de ce bloc de $\mathbf{H}$ (la matrice de parité complète recherchée). Si le nombre d'équations est suffisant, ce nombre sera majoritairement égal au poids de la colonne. Ainsi, en traçant l'histogramme représentant le poids de chaque colonnes de $\mathbf{H_p}$, il est possible de déduire les poids des colonnes de la matrice de parité du code LDPC.

La figure 3.3 représente l'histogramme d'une matrice $\mathbf{H_p}$ composée de 1901 équations de parité de poids 6 et 7 du code $\mathcal{C}_{2304}$ que nous avons déjà vu précédemment. La matrice $\mathbf{H_p}$ contient donc environ 95% des équations de parité recherchées. Cette matrice partielle a été obtenue en exécutant la méthode Tixier-Tillich pendant 4 minutes sur des mots bruité par un canal binaire symétrique de probabilité d'erreur $4 \cdot 10^{-4}$. Notons qu'il a nécessité 3 heures supplémentaire pour obtenir 98% des équations, ce qui illustre bien le phénomène du collectionneur de coupons.

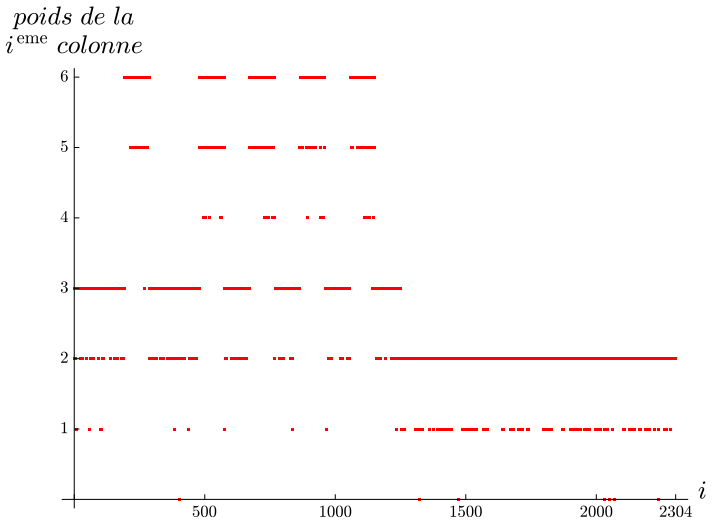


Figure 3.3 – Poids de la colonne de $\mathbf{H_p}$ en fonction de l'indice de colonne.

La figure 3.4 représente l'histogramme de la matrice de parité creuse $\mathbf{H}$ du code $\mathcal{C}_{2304}$ (composée des 1152 équations de parité de poids 6 et 7 du code). Il est possible de déduire cette nouvelle figure à partir de la figure 3.3.

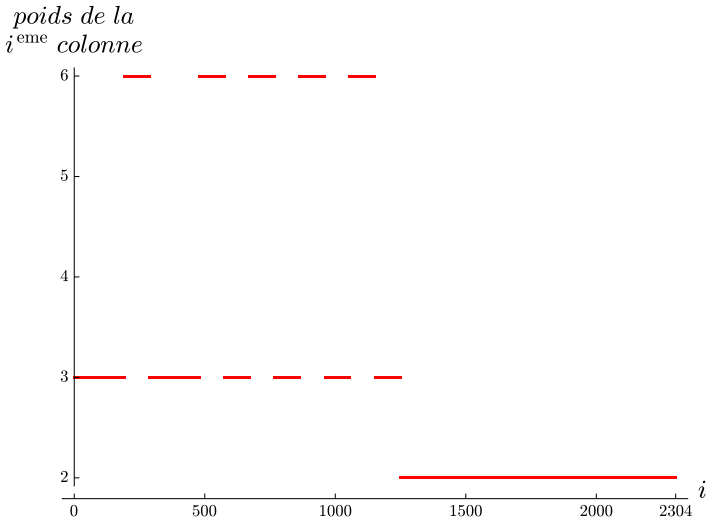


Figure 3.4 – Poids de la colonne de $\mathbf{H}$ en fonction de l'indice de colonne.

Finalement, connaissant les poids des colonnes de la matrice de parité recherchée, il est possible de déterminer si l'ensemble des équations retrouvées jusqu'ici contient toutes les équations faisant intervenir un indice de colonne particulier. Si c'est le cas, alors on ne recherche aucune nouvelle équation faisant intervenir cet indice. La recherche de ces équations peut alors se faire sur les mots bruités poinçonnés sur les indices des colonnes qui n'interviennent pas dans les équations manquantes. La longueur du code nouvellement considéré est alors réduite sans que le poids des équations recherchées soit augmenté.

Cela permet d'accélérer la recherche des équations manquantes ; notamment des dernières équations qui sont souvent plus difficiles à retrouver.

Dans l'exemple précédent, les 4 minutes d'exécution de la méthode Tixier-Tillich ont permis de poinçonner les mots bruités du code $\mathcal{C}_{2304}$ sur 1952 positions, nous ramenant ainsi à la reconnaissance d'un code de longueur 352 (et dont les équations sont toujours de poids 6 et 7). Quelques minutes supplémentaires ont alors suffit pour retrouver les dernières équations.

3.2 Deux méthodes classiques

Les méthodes d'accélération de la reconnaissance des codes LDPC nécessitent toujours de trouver quelques équations de parité au préalable. En outre, elles ne sont pas toujours applicables ; notamment lorsque le code recherché ne possède pas la structure adéquate. La reconnaissance de codes LDPC est donc intrinsèquement liée à la recherche d'équations de parités creuses dans un code inconnu aléatoire (sans structure supposée). Dann cette section, nous explorons d'autres méthodes pour résoudre ce problème : celle de Cluzeau et Finiasz [CF09b] et celle de Sicot,Houcke et Barbier [BSH06, SHB09]. Ces méthodes ne sont pas aussi efficaces que la méthode Tixier-Tillich, mais elles vont nous permettre d'introduire une nouvelle méthode dans la section suivante. Notre méthode interpole les deux algorithmes que nous allons voir dans cette section. Elle est toutefois plus efficace que l'une et l'autre et même plus efficace que la méthode Tixier-Tillich.

3.2.1 La méthode de Cluzeau et Finiasz

Rappelons que notre problème de reconnaissance de codes est de trouver un vecteur $\mathbf{h} \in \mathbb{F}_2^n$ de poids w tel que $\widetilde{\mathbf{G}}\mathbf{h}^\top$ soit de poids $\leqslant \frac{1-(1-2\tau)^w}{2} M$ où $\widetilde{\mathbf{G}}$ est une matrice de taille $M \times n$. Ce problème revient donc à trouver un ensemble de w colonnes de $\widetilde{\mathbf{G}}$ dont la somme est de poids faible. Nous avons vu dans la sous-section 2.5.2 comment effectuer cette recherche avec une recherche de presque-collisions. L'algorithme 3.2.1 décrit ainsi la méthode Cluzeau-Finiasz [CF09b] obtenue en résolvant le problème des presque-collisions avec des projections.

Une équation de parité $\mathbf{h}$ de poids w du code $\mathcal{C}$ est trouvée lors d'une itération de l'algorithme de Cluzeau et Finiasz si et seulement si $|\mathbf{h}_I| = |\mathbf{h}_J| = \frac{w}{2}$ et si $\mathbf{h}$ vérifie les ℓ mots bruités indexés par L. Ainsi, la probabilité de succès d'une itération de la méthode Cluzeau-Finiasz est :

$$\mathbb{P}_{\text{succ}} = \frac{\binom{n/2}{w/2}^2}{\binom{n}{w}} \left(\frac{1 + (1-2\tau)^w}{2} \right)^\ell \tag{3.10}$$

Il faut donc itérer $s := \frac{1}{\mathbb{P}_{\text{succ}}}$ fois la procédure dans l'algorithme 3.2.1 pour trouver les équations de parité de poids w.

Algorithme 3.2.1 : La méthode Cluzeau-Finiasz

Entrées : la longueur n de $\mathcal{C}$;
$\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_M$: M mots de $\mathcal{C}$ bruités par un BSC(τ) ;
le poids w des équations recherchées.
Paramètres : un entier $\ell \in [\![1, M]\!]$.
Sortie : une liste $\mathscr{L}$ de potentielles équations de parité de $\mathcal{C}$.

$\widetilde{\mathbf{G}} \leftarrow$ placer les mots reçus dans une matrice $M \times n$;
$\mathscr{L} \leftarrow \varnothing$;
répéter s **fois**
  choisir un ensemble $L \subseteq [\![1, M]\!]$ tel que $|L| = \ell$;
  choisir un ensemble $I \subseteq [\![1, n]\!]$ tel que $|I| = \frac{n}{2}$;
  $J \leftarrow [\![1, n]\!] \setminus I$;
  $\mathscr{L}_1 \leftarrow \left\{ \left(\mathbf{x} \widetilde{\mathbf{G}}_I^\top \right)_L \ : \ \mathbf{x} \in \mathbb{F}_2^{\frac{n}{2}} \text{ et } |\mathbf{x}| = \frac{w}{2} \right\}$;
  $\mathscr{L}_2 \leftarrow \left\{ \left(\mathbf{x} \widetilde{\mathbf{G}}_J^\top \right)_L \ : \ \mathbf{x} \in \mathbb{F}_2^{\frac{n}{2}} \text{ et } |\mathbf{x}| = \frac{w}{2} \right\}$;
  $\mathscr{L} \leftarrow \left\{ \mathbf{h} := (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{F}_2^{\frac{n}{2}} \times \mathbb{F}_2^{\frac{n}{2}} \ : \ \left(\mathbf{x}_1 \widetilde{\mathbf{G}}_I^\top \right)_L = \left(\mathbf{x}_2 \widetilde{\mathbf{G}}_J^\top \right)_L \text{ et } |\mathbf{x}_1| = |\mathbf{x}_2| = \frac{w}{2} \right\}$;
  `/* cette liste est obtenue en effectuant une recherche de collisions sur` $\mathscr{L}_1 \times \mathscr{L}_2$ `*/`
finRépéter
retourner $\mathscr{L}$

3.2.2 La méthode de Sicot, Houcke et Barbier

Dans [BSH06, SHB09], il est proposé une autre façon de produire des sommes creuses de colonnes dans une matrice en effectuant simplement une élimination gaussienne sur la matrice $\widetilde{\mathbf{G}}$. La méthode Sicot-Houcke-Barbier effectue cette élimination gaussienne en opérant sur les colonnes de la matrice. De plus, nous choisissons $M = n$ mots bruités pour construire la matrice $\widetilde{\mathbf{G}}$ qui est donc une matrice carrée $n \times n$:

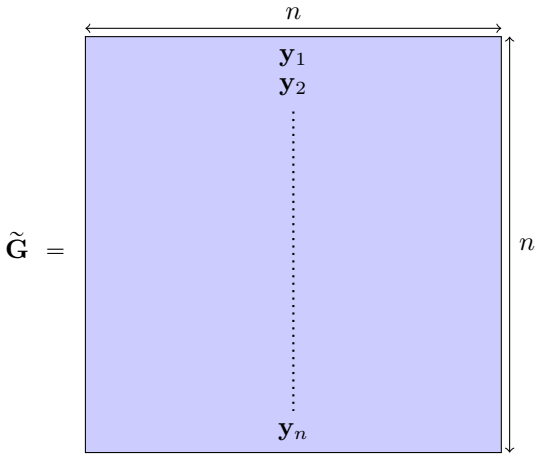


Figure 3.5 – Description de la matrice $\widetilde{\mathbf{G}}$ pour la méthode Sicot-Houcke-Barbier.

Supposons dans un premier temps le cas trivial où τ est nul. Alors la matrice $\widetilde{\mathbf{G}}$ est de rang égal à la dimension k du code $\mathcal{C}$ (en supposant que les mots $\mathbf{y}_i$ soient linéairement

indépendants). Une élimination gaussienne opérant sur les colonnes de la matrice $\widetilde{\mathbf{G}}$ permet alors de produire la matrice triangulaire inférieure $\widetilde{\mathbf{G}}\mathbf{A}$ de la figure 3.6.

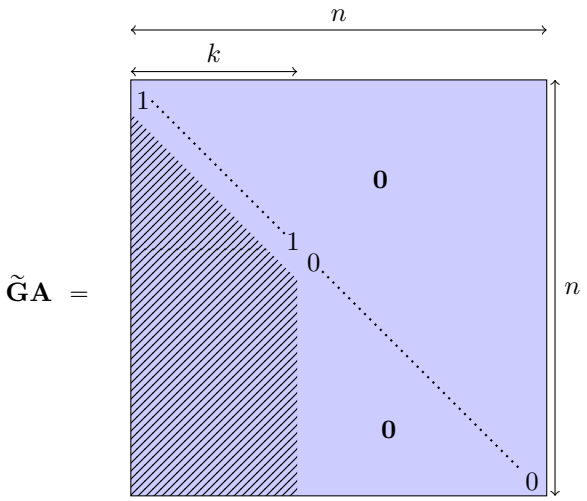


Figure 3.6 – La matrice obtenue après une élimination gaussienne opérant sur les colonnes de $\widetilde{\mathbf{G}}$ pour $\tau = 0$.

Dans la figure 3.6, $\mathbf{A}$ est la matrice de passage de l'élimination gaussienne. C'est une matrice carrée de taille n.

Remarque 3.2.1. On s'autorise ici des permutations de lignes qui n'ont aucun impact sur le code $\mathcal{C}$.

Après l'élimination gaussienne, nous observons $n - k$ défauts de rang ; ce sont les colonnes nulles de la matrice $\widetilde{\mathbf{G}}\mathbf{A}$. Or, étant donnée la nature des opérations effectuées lors de l'élimination gaussienne, chacune de ces colonnes nulles est en fait la somme de colonnes de la matrice $\widetilde{\mathbf{G}}$ d'origine. Ces sommes sont sauvegardées dans la matrice $\mathbf{A}$. Par exemple, la $i^{\text{ème}}$ colonne $(\widetilde{\mathbf{G}}\mathbf{A})_{*,i}$ de $\widetilde{\mathbf{G}}\mathbf{A}$ est égale à $\widetilde{\mathbf{G}}\mathbf{A}_{*,i}$; c'est-à-dire la somme des colonnes de $\widetilde{\mathbf{G}}$ indexées par le support de la $i^{\text{ème}}$ colonne de $\mathbf{A}$. Ainsi, cette méthode permet de trouver $n - k$ mots $\mathbf{A}_{*,k+1}, \cdots, \mathbf{A}_{*,n}$ dont le produit à gauche par la matrice $\widetilde{\mathbf{G}}$ vaut $\mathbf{0}$. Ces mots sont donc des équations de parité de $\mathcal{C}$.

Dans leur article [SHB09], Sicot, Houcke et Barbier remarquent qu'un constat similaire peut être fait dans le cas bruité. En effet, supposons cette fois-ci que $\tau > 0$. Alors la matrice $\widetilde{\mathbf{G}}$ est de rang strictement supérieur à k. Si le bruit n'est pas excessif, alors nous pouvons toujours supposer que $\widetilde{\mathbf{G}}$ n'est pas de rang plein et donc qu'une élimination gaussienne permettra de détecter un certain nombre de défauts de rang ($< n - k$). La figure 3.7 représente une telle matrice obtenue après élimination gaussienne.

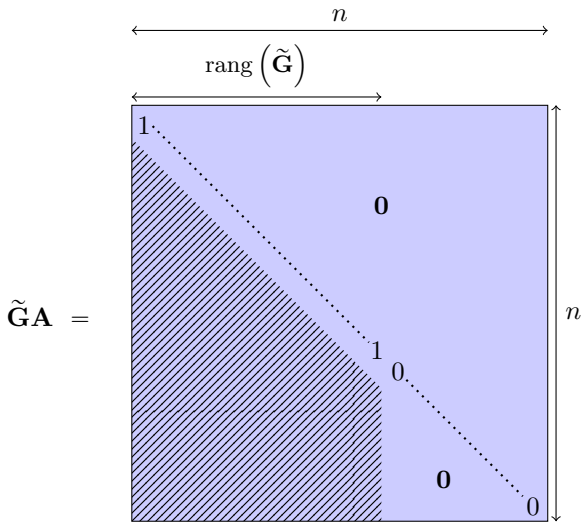


Figure 3.7 – La matrice obtenue après une élimination gaussienne opérant sur les colonnes de $\widetilde{\mathbf{G}}$ pour $\tau > 0$

Finalement, la méthode Sicot-Houcke-Barbier est résumée dans l'algorithme 3.2.2.

Algorithme 3.2.2 : La méthode de Sicot-Houcke-Barbier

Entrées : la longueur n de $\mathcal{C}$;
$\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_M$: $M \geqslant n$ mots de $\mathcal{C}$ bruités par un BSC(τ) ;
le poids w des équations recherchées.

Sortie : une liste $\mathscr{L}$ de potentielles équations de parité de $\mathcal{C}$.

$\mathscr{L} \leftarrow \varnothing$;
répéter s **fois**
  $\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n \leftarrow$ choisir n mots aléatoirement parmi les mots bruité reçus ;
  $\widetilde{\mathbf{G}} \leftarrow$ placer les mots $\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n$ en ligne dans une matrice carrée $n \times n$;
  Effectuer une élimination gaussienne en opérant sur les colonnes de $\widetilde{\mathbf{G}}$ pour produire une matrice $\mathbf{A}$ telle que $\widetilde{\mathbf{G}}\mathbf{A}$ soit une matrice triangulaire inférieure ;
  pour tout $i \in [\![1, n]\!]$ *tel que la* $i^{\text{ème}}$ *colonne de* $\widetilde{\mathbf{G}}\mathbf{A}$ *soit nulle* **faire**
    ajouter la $i^{\text{ème}}$ colonne de $\mathbf{A}$ à la liste $\mathscr{L}$;
  finPour
finRépéter
retourner $\mathscr{L}$

La méthode de Sicot-Houcke-Barbier n'a aucune emprise sur les poids des équations trouvées. En effet, contrairement à la méthode Cluzeau-Finiasz, l'algorithme 3.2.2 ne retourne pas des équations de poids fixe. Les équations retournées sont de poids quelconque sur rang $\left(\widetilde{\mathbf{G}}\right)$ bits et de poids 1 sur les autres positions. Toutefois, dès qu'une équation de parité $\mathbf{h}$ vérifie les n mots bruités choisis lors d'une itération, une équation $\mathbf{h}'$ est trouvée durant cette itération. Notons que $\mathbf{h}'$ n'est pas nécessairement $\mathbf{h}$ mais peut être une combinaison linéaire de $\mathbf{h}$ et d'une autre équation de parité. Ainsi la probabilité de

succès d'une itération de la méthode de Sicot-Houcke-Barbier est :

$$\mathbb{P}_{\text{succ}} = \left(\frac{1+(1-2\tau)^w}{2}\right)^n \tag{3.11}$$

Nous choisissons alors $s := \frac{1}{\mathbb{P}_{\text{succ}}}$ dans l'algorithme 3.2.2.

La sensibilité au bruit de la méthode Sicot-Houcke-Barbier. La méthode Sicot-Houcke-Barbier est extrêmement sensible au niveau de bruit τ et cette sensibilité est directement liée à l'élimination gaussienne. En effet, les différentes opérations que nous effectuons sur la matrice $\widetilde{\mathbf{G}}$ lors de l'élimination gaussienne propagent les erreurs dans la matrice. Par exemple, soit $\mathcal{C}[200, 100, 12, 6]$ un code LDPC régulier possédant une matrice de parité dont les lignes sont de poids 12 et les colonnes de poids 6. Les figures 3.8, 3.9 et 3.10 représentent une matrice formée de 150 mots bruités du code $\mathcal{C}[200, 100, 12, 6]$ à différents stades de l'élimination gaussienne. Pour produire ces figures, nous avons simulé un canal de transmission de probabilité d'erreur égale à 0.003. La matrice $\widetilde{\mathbf{G}}$ est donc égale à la somme $\mathbf{G} + \mathbf{E}$ où les lignes de $\mathbf{G}$ sont des mots du code et où les bits de $\mathbf{E}$ valent 1 avec une probabilité τ. On note $\mathbf{A}$ la matrice de passage donnant l'une des figures précédemment citées. Sur la figure considérée, les carrés noirs représentent alors les bits à 1 de $\widetilde{\mathbf{G}}\mathbf{A}$ et les carrés blancs les bits à 0 de cette même matrice. Les carrés rouges représentent quant à eux les bits erronés ; c'est-à-dire les bits à 1 de la matrice $\mathbf{EA}$. Nous pouvons constater que plus nous effectuons d'étapes de l'élimination gaussienne, plus le nombre d'erreurs dans la matrice est important.

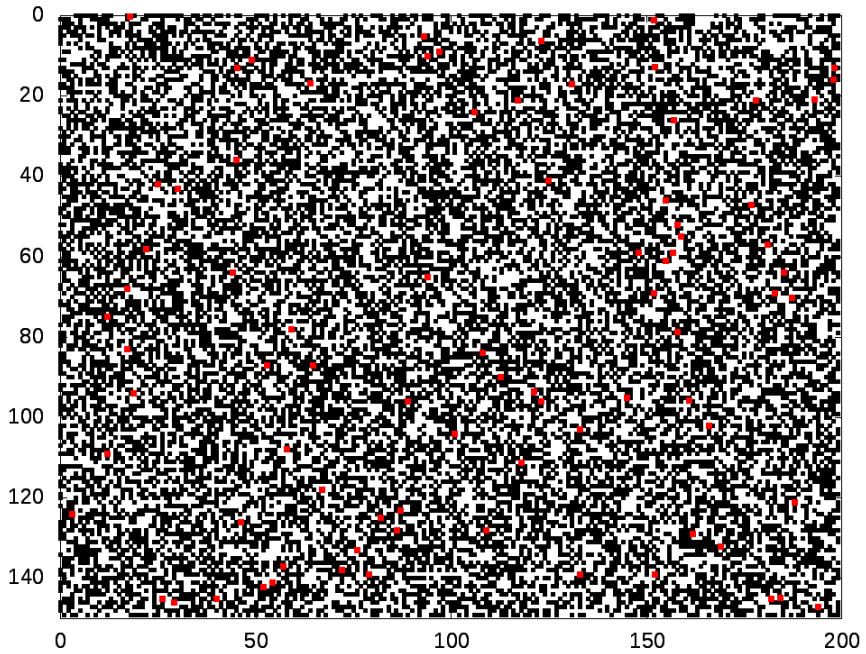


Figure 3.8 – Les bits erronés dans la matrice formée de 150 mots bruités du code LDPC $\mathcal{C}[200, 500, 6, 12]$.

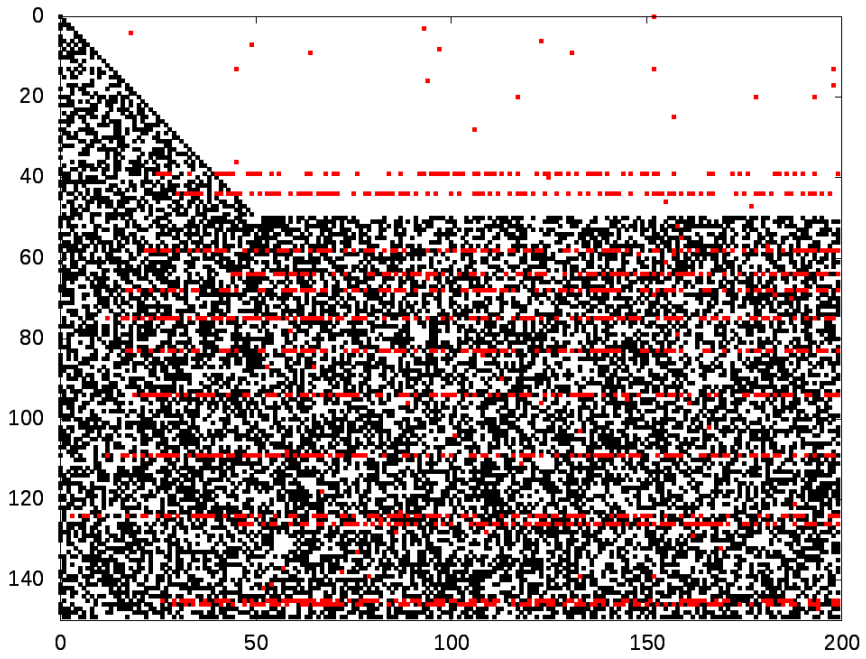


Figure 3.9 – Les bits erronés après 50 étapes de l'élimination gaussienne.

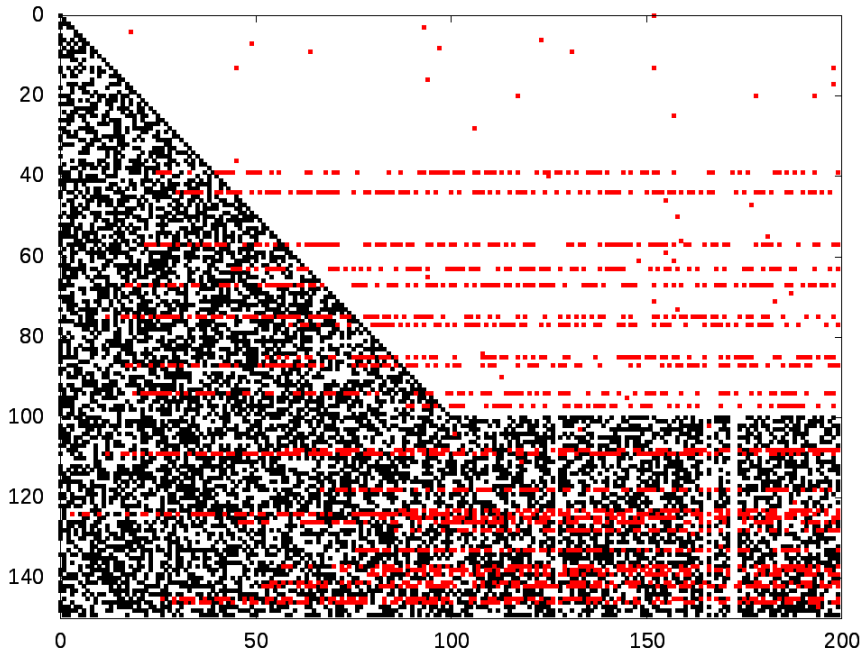


Figure 3.10 – Les bits erronés après 100 étapes de l'élimination gaussienne.

Remarque 3.2.2. Le *piling-up lemma* (cf. lemme 3.1.1) permet de mesurer précisément la propagation de l'erreur lors de l'élimination gaussienne.

3.3 Reconnaitre un code LDPC par élimination gaussienne partielle

Nous allons à présent construire une nouvelle méthode interpolant celles de Cluzeau-Finiasz et Sicot-Houcke-Barbier. Remarquons tout d'abord que nous pouvons limiter la propagation du bruit par rapport à la méthode Sicot-Houcke-Barbier en n'effectuant qu'une élimination gaussienne partielle de la matrice $\widetilde{\mathbf{G}}$.

Notre méthode commence donc par choisir l'ensemble $J \subseteq [\![1, n]\!]$ de taille t contenant les indices des colonnes pivots (t est un paramètre à optimiser). On pose alors $I := [\![1, n]\!] \backslash J$. Il est supposé que $\widetilde{\mathbf{G}}_J$ est de rang plein. L'élimination gaussienne partielle opérant sur les colonnes de $\widetilde{\mathbf{G}}$ permet alors de produire la matrice de la figure 3.11.

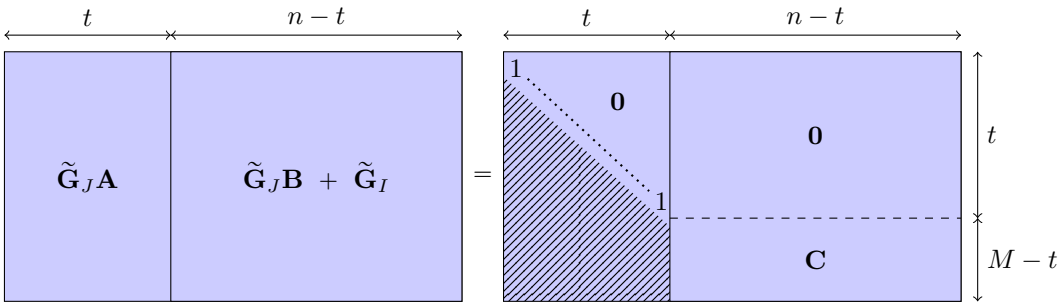


Figure 3.11 – La matrice produite par une élimination gaussienne partielle opérant sur les colonnes de $\widetilde{\mathbf{G}}$.

Dans la figure 3.11, les matrices $\mathbf{A}$, $\mathbf{B}$ et $\mathbf{C}$ sont des matrices de tailles respectives $t \times t$, $t \times (n-t)$ et $(M-t) \times (n-t)$.

Remarque 3.3.1. Contrairement à la méthode de Sicot-Houcke-Barbier, notre méthode n'impose pas d'observer un nombre de mots bruités égal à la longueur du code.

Remarque 3.3.2. Lors de l'élimination gaussienne, on s'autorise des permutations de lignes qui n'ont aucun impact sur notre méthode.

Par construction, la $i^{\text{ème}}$ colonne de la matrice $\widetilde{\mathbf{G}}_J\mathbf{B} + \widetilde{\mathbf{G}}_I$ est une somme de colonnes de la matrice $\widetilde{\mathbf{G}}_J$ et de la colonne $(\widetilde{\mathbf{G}}_I)_{*,i}$. Plus précisément :

$$(\widetilde{\mathbf{G}}_J\mathbf{B} + \widetilde{\mathbf{G}}_I)_{*,i} = \widetilde{\mathbf{G}}_J\mathbf{B}_{*,i} + (\widetilde{\mathbf{G}}_I)_{*,i} \tag{3.12}$$

$$= (\widetilde{\mathbf{G}}_I)_{*,i} + \sum_{j \in \text{supp}(\mathbf{B}_{*,i})} (\widetilde{\mathbf{G}}_J)_{*,j} \tag{3.13}$$

Ainsi, soit $S \subseteq [\![1, n-t]\!]$ tel que $\#S = p$ (avec p un paramètre à optimiser) et tel que :

$$\sum_{i \in S} \left(\widetilde{\mathbf{G}}_J\mathbf{B} + \widetilde{\mathbf{G}}_I\right)_{*,i} \tag{3.14}$$

soit creux. Alors le vecteur $\mathbf{h}$ tel que $\mathbf{h}_J^\top = \sum_{i \in S} \mathbf{B}_{*,i}$ et tel que $\mathbf{h}_I$ soit de support S est une potentielle équation de parité du code puisqu'elle vérifie la propriété que $\widetilde{\mathbf{G}}\mathbf{h}^\top$ est creux. En effet :

$$\begin{aligned}\widetilde{\mathbf{G}}\mathbf{h}^\top &= \widetilde{\mathbf{G}}_J\mathbf{h}_J^\top + \widetilde{\mathbf{G}}_I\mathbf{h}_I^\top \\ &= \widetilde{\mathbf{G}}_J\left(\sum_{i\in S}\mathbf{B}_{*,i}\right) + \sum_{i\in S}(\widetilde{\mathbf{G}}_I)_{*,i} \\ &= \sum_{i\in S}(\widetilde{\mathbf{G}}_J\mathbf{B})_{*,i} + \sum_{i\in S}(\widetilde{\mathbf{G}}_I)_{*,i} \\ &= \sum_{i\in S}\left(\widetilde{\mathbf{G}}_J\mathbf{B} + \widetilde{\mathbf{G}}_I\right)_{*,i}\end{aligned}$$

En outre, remarquons que rechercher des sommes creuses de colonnes de $\widetilde{\mathbf{G}}_J\mathbf{B} + \widetilde{\mathbf{G}}_I$ revient exactement à rechercher des sommes creuses de colonnes de la sous-matrice $\mathbf{C}$ puisque les t premières lignes de la matrice $\widetilde{\mathbf{G}}_J\mathbf{B} + \widetilde{\mathbf{G}}_I$ sont nulles.

L'algorithme 3.3.1 résume notre méthode.

Algorithme 3.3.1 : La méthode de l'élimination gaussienne partielle opérant sur les colonnes

Entrées : la longueur n de $\mathcal{C}$;
$\mathbf{y}_1, \cdots, \mathbf{y}_M$: un ensemble de M mots de $\mathcal{C}$ bruités par un BSC(τ).
Paramètres : un entier $t \in [\![0, \min(n, M)]\!]$;
un entier $p \in [\![0, n-t]\!]$;
un entier θ.
Sortie : une liste $\mathscr{L}$ de potentielles équations de parité de $\mathcal{C}$.

1 $\widetilde{\mathbf{G}} \leftarrow$ placer les mots reçus dans les lignes d'une matrice $M \times n$;
2 $\mathscr{L} \leftarrow \varnothing$;
3 **répéter indéfiniment**
4 choisir un ensemble $I \subseteq [\![1, n]\!]$ tel que $|I| = n - t$;
5 $J \leftarrow [\![1, n]\!] \backslash I$;
6 $(\mathbf{A}, \mathbf{B}, \mathbf{C}) \leftarrow$ effectuer une élimination gaussienne en opérant sur les colonnes de $\widetilde{\mathbf{G}}$ pour produire les matrices $\mathbf{A}$, $\mathbf{B}$ et $\mathbf{C}$ comme décrit précédemment ;
7 **pour tout** $\mathbf{x} \in \mathbb{F}_2^{n-t}$ *tel que* $|\mathbf{x}| = p$ *et* $\left|\mathbf{C}\mathbf{x}^\top\right| \leqslant \theta$ **faire**
8 construire $\mathbf{h}$ tel que $\mathbf{h}_J = \mathbf{x}$ et $\mathbf{h}_I = \mathbf{B}\mathbf{x}^\top$;
9 ajouter $\mathbf{h}$ à la liste $\mathscr{L}$;
10 **finPour**
11 **finRépéter**
12 **retourner** $\mathscr{L}$

Pour simplifier l'algorithme, nous avons décrit celui-ci dans le cas où $\widetilde{\mathbf{G}}_J$ est de rang plein. De façon analogue à la sous-section 3.1.2, nous pouvons traiter les itérations qui ne vérifient pas cette propriété.

D'autre part, le paramètre θ est choisi approximativement au milieu des quantités $\frac{1-(1-2\tau)^w}{2}(M-t)$ et $\frac{M-t}{2}$ où w représente le poids typique des équations de parité recherchées.

Notons aussi que nous n'avons pas détaillé comment énumérer les éléments $\mathbf{x}$ dans la boucle 7. Cette étape consiste à rechercher des sommes creuses de p colonnes dans la matrice $\mathbf{C}$. Nous pouvons alors utiliser une (ou des) recherche de presque-collisions comme nous l'avons déjà fait dans la méthode de décodage ISD de Stern (cf. sous-section 2.5.2) ou encore celle de Both et May (cf. sous-section 2.7). Dans tous les cas, nous devons expliciter une méthode pour résoudre le problème des presque-collisions ; ceci sera l'objet des chapitre 5 à 9 de ce manuscrit.

Finalement, le théorème 3.3.1 explicite les équations de parité retournées par l'algorithme 3.3.1.

Théorème 3.3.1. *Soit une itération de l'algorithme 3.3.1 associée au choix d'un ensemble J. Supposons que $\widetilde{\mathbf{G}}_J$ est de rang plein. Alors, les équations produites par l'algorithme lors de cette itération est l'ensemble des mots $\mathbf{h} \in \mathbb{F}_2^n$ satisfaisant les conditions suivantes :*

(a) $|\mathbf{h}_J| = p$;

(b) $\forall i \in [\![1, t]\!] \ , \ <\mathbf{y}_i, \mathbf{h}> = \ 0$;

(c) $\sum_{i=t+1}^{M} <\mathbf{y}_i, \mathbf{h}> \ \leqslant \ \theta.$

Remarque 3.3.3. Le produit scalaire $<\cdot,\cdot>$ est sur $\mathbb{F}_2$ tandis que la somme $\sum \cdot$ est sur $\mathbb{Z}$.

3.3.1 Analyse de la méthode

Le théorème suivant donne le nombre moyen d'équations de parité trouvées par l'algorithme 3.3.1 lors d'une itération.

Théorème 3.3.2. *Soit la variable aléatoire U représentant le nombre d'équations de parité de $\mathcal{C}$ satisfaisant les conditions (a), (b) et (c) du théorème 3.3.1.*
L'espérance de U est :

$$\mathbb{E}(U) = \sum_{w=p}^{t+p} A_w \mathbb{P}_{(a)}^w \mathbb{P}_{(b)}^w \mathbb{P}_{(c)}^w \tag{3.15}$$

avec A_w, *le nombre d'équations de parité de poids w dans le code $\mathcal{C}$;*

$$\mathbb{P}_{(a)}^w = \frac{\binom{t}{w-p}\binom{n-t}{p}}{\binom{n}{w}} \quad ;$$

$$\mathbb{P}_{(b)}^w = \left(\frac{1+(1-2\tau)^w}{2}\right)^{\top} ;$$

$$et \quad \mathbb{P}_{(c)}^w = \sum_{\ell=0}^{\theta} \binom{M-t}{\ell} \left(\frac{1-(1-2\tau)^w}{2}\right)^{\ell} \left(\frac{1+(1-2\tau)^w}{2}\right)^{M-t-\ell}.$$

Démonstration du théorème 3.3.2.

Nous commençons la preuve en observant que

$$\begin{aligned} U &= \sum_{\mathbf{h}\in\mathcal{C}^\perp} \mathbf{1}_{\{\mathbf{h} \text{ satisfait (a), (b) et (c)}\}} \\ &= \sum_{w=0}^{n} \sum_{\mathbf{h}\in\mathcal{C}^\perp, |\mathbf{h}|=w} \mathbf{1}_{\{\mathbf{h} \text{ satisfait (a), (b) et (c)}\}} \end{aligned}$$

Noter que les mots de poids $< p$ ou $> t+p$ ne peuvent pas satisfaire la condition (a).
C'est pourquoi on a :

$$U = \sum_{w=p}^{t+p} \sum_{\mathbf{h}\in\mathcal{C}^\perp, |\mathbf{h}|=w} \mathbf{1}_{\{\mathbf{h} \text{ satisfait (a), (b) et (c)}\}}$$

Ce qui implique que :

$$\mathbb{E}(U) = \sum_{w=p}^{t+p} \sum_{\mathbf{h}\in\mathcal{C}^\perp, |\mathbf{h}|=w} \mathbb{P}(\mathbf{h} \text{ satisfait (a), (b) et (c)})$$

Or les conditions (a), (b) et (c) sont indépendantes. D'où :

$$\mathbb{P}(\mathbf{h} \text{ satisfait (a), (b) et (c)}) = \mathbb{P}(\mathbf{h} \text{ satisfait (a)}) \times \mathbb{P}(\mathbf{h} \text{ satisfait (b)}) \times \mathbb{P}(\mathbf{h} \text{ satisfait (c)})$$

Les probabilités $\mathbb{P}(\mathbf{h} \text{ satisfait (a)})$, $\mathbb{P}(\mathbf{h} \text{ satisfait (b)})$ et $\mathbb{P}(\mathbf{h} \text{ satisfait (c)})$ sont respectivement dénotées $\mathbb{P}^w_{(a)}$, $\mathbb{P}^w_{(b)}$ et $\mathbb{P}^w_{(c)}$. Ces probabilités sont déterminées par de simples calculs de dénombrement et de probabilités en lien avec le *piling-up lemma*.

□

Le nombre d'itérations nécessaires pour trouver une équation de parité est estimé à $\frac{1}{\mathbb{E}(U)}$.

Notons toutefois que la condition (a) suppose que nous énumérions toutes les sommes de p colonnes d'une matrice $(M-t) \times (n-t)$. En pratique, nous résolvons cette étape à la manière de Stern (cf. sous-section 2.5.2). Ainsi, le poids p du vecteur $\mathbf{h}_I$ est réparti équitablement sur les deux moitiés de ce vecteur. La condition (a) devient alors :

$$(a') \quad |\mathbf{h}_{I_1}| = \frac{p}{2} \quad \text{et} \quad |\mathbf{h}_{I_2}| = \frac{p}{2} \tag{3.16}$$

où $I = I_1 \cup I_2$ et $\#I_1 = \#I_2 = \frac{n-t}{2}$.

La probabilité $\mathbb{P}^w_{(a')}$ de satisfaire cette condition est :

$$\mathbb{P}^w_{(a')} = \frac{\binom{t}{w-p}\binom{(n-t)/2}{p/2}^2}{\binom{n}{w}} \tag{3.17}$$

De plus, le coût d'une itération est alors de l'ordre de :

$$C_{\text{iter}} = O\left(tnM + \left(\binom{(n-t)/2}{p/2} + \frac{\binom{(n-t)/2}{p/2}^2}{2^\ell}\right)\left(\frac{1-(1-2\tau)^w}{2}\right)^{-\ell}\right) \tag{3.18}$$

avec w le poids typique des équations recherchées et ℓ un nouveau paramètre à optimiser.

Le théorème 3.3.2 fait intervenir le nombre A_w d'équations de parité de poids w dans le code que l'on cherche à reconnaitre. Cette quantité n'est pas forcément connue mais nous pouvons souvent en faire une estimation précise. Nous verrons notamment dans le chapitre suivant comment affiner cette estimation.

3.3.2 Quelques résultats numériques

Nous donnons ici quelques résultats numériques sur nos méthodes de recherche d'équations de parité creuses. Tout d'abord, nous définissons le *workfactor* comme étant le coût moyen pour trouver une équation de parité du code inconnu. Le *workfactor* est donc égal au rapport du coût d'une itération sur l'espérance du nombre d'équations trouvées lors d'une itération. La minimisation du *workfactor* nous permet de déterminer le paramètre t optimal pour nos deux méthodes.

En outre, les résultats que nous donnons ici ne prennent pas en compte nos travaux sur la résolution du problème de presque-collisions que nous développerons plus loin dans ce manuscrit. Nous considérons donc que nous résolvons ce problème par la méthode naïve des projections qui est essentiellement la même méthode que celle que nous avons déjà vu pour le décodage ISD de Stern (cf. sous-section 2.5.2).

Un code LDPC régulier de type $[n, k, w_l, w_c]$ est un code de longueur n et de dimension k possédant une matrice de parité dont les colonnes sont de poids w_c et les lignes de poids w_l. Nous avons étudié les performances de notre algorithme pour différents codes LDPC réguliers de type $[n, n/2, w, w/2]$. Reconnaitre ces codes consiste à trouver $n - k := \frac{n}{2}$ équations de parité indépendantes de poids w.

Les figures 3.12, 3.13 et 3.14 comparent les méthodes Sicot-Houcke-Barbier, Cluzeau-Finiasz ainsi que notre méthode. Pour cette dernière, nous avons choisi les paramètres

t et p qui minimise le *workfactor* de l'algorithme 3.3.1. Dans la figure 3.12, nous avons tracé la complexité pour trouver une équation de parité d'un code LDPC régulier de type $[1024, 512, w, w/2]$ en fonction de w. Le canal binaire symétrique considéré a une probabilité d'erreur $\tau = 0.005$.

Nous observons sur la figure 3.12 que la complexité de la méthode Sicot-Houcke-Barbier est trop importante pour le niveau de bruit considéré : le coût pour produire juste une équation de parité est déjà de l'ordre de 2^{65} lorsque w vaut seulement 6. Ainsi, pour un tel niveau de bruit, l'approche de Sicot-Houcke-Barbier est à exclure. D'un autre coté, la méthode Cluzeau-Finiasz a une complexité raisonnable pour $w \in \{6, 8\}$ mais devient vraiment coûteuse lorsque $w \geqslant 12$ où la complexité pour trouver une seule équation de parité est de l'ordre de 2^{50}. Notre méthode est elle aussi coûteuse pour des poids w supérieurs à 12 mais elle reste toutefois au moins 1000 fois plus rapide que la méthode Cluzeau-Finiasz.

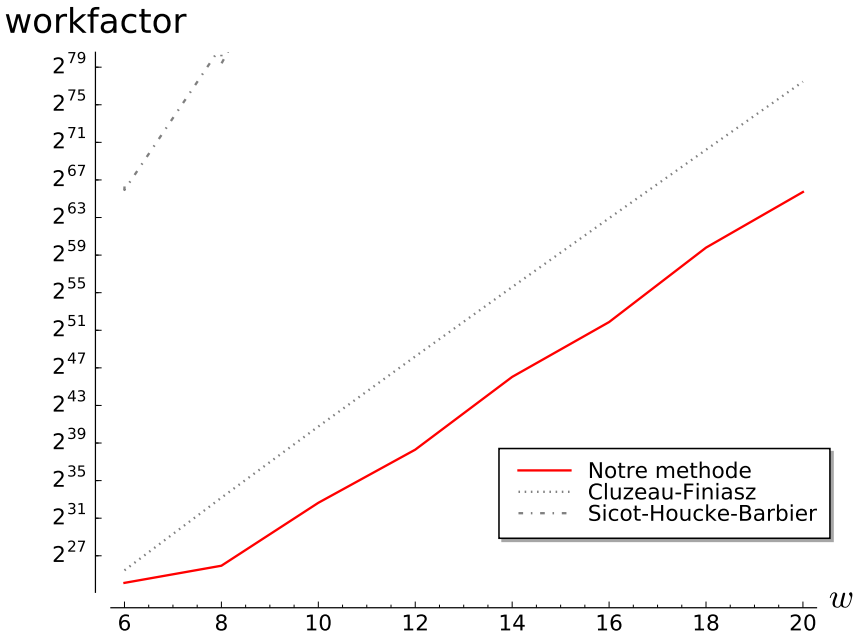


Figure 3.12 – Complexité pour trouver une équation de parité d'un code LDPC régulier de type $[1024, 512, w, w/2]$ avec $\tau = 0.005$

La figure 3.13 représente le coût pour trouver une équation de parité d'un code LDPC régulier de type $[1024, 512, 12, 6]$ pour différents niveaux de bruit. On constate qu'il est difficile de trouver des équations de parité lorsque la valeur de τ dépasse plusieurs millièmes. En revanche, pour des niveaux de bruit relativement faibles, la complexité de la méthode Sicot-Houcke-Barbier est proche de la notre et permet de trouver des équations de parité très rapidement.

L'approche de Cluzeau-Finiasz est cependant trop coûteuse pour être implémentée en pratique ici ; et ce quelque soit le niveau de bruit.

Sur la figure 3.13, nous pouvons aussi remarquer que notre méthode améliore significativement les méthodes existantes lorsque τ est proche de 0.002 par exemple.

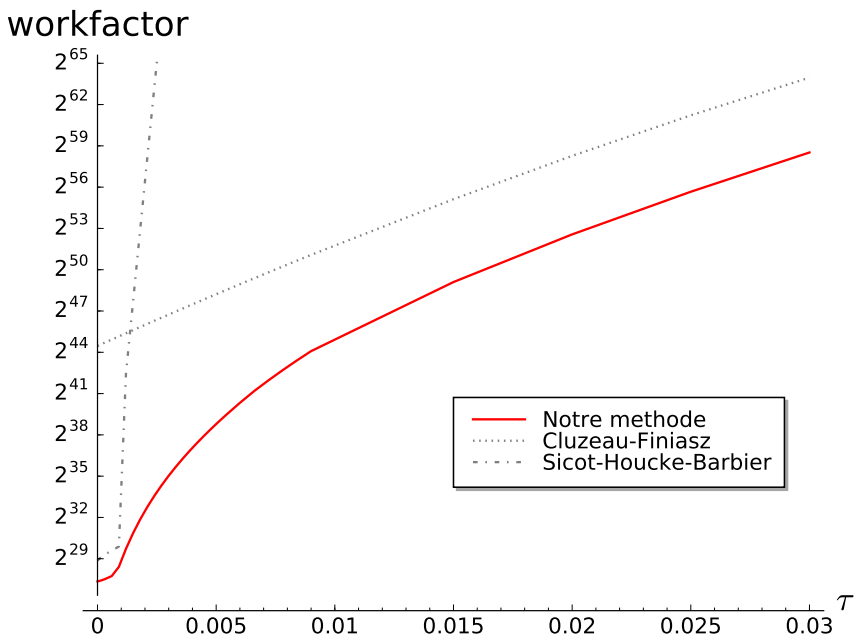


Figure 3.13 – Complexité pour trouver une équation de parité d'un code LDPC régulier de type $[1024, 512, 12, 6]$ en fonction de τ.

Enfin, la figure 3.14 représente la complexité pour trouver une équation de parité d'un code LDPC régulier de type $[n, n/2, 12, 6]$ en fonction de la longueur du code. La probabilité du canal binaire symétrique est $\tau = 0.005$.

La figure 3.14 montre que dans ce contexte, notre méthode améliore significativement les méthodes existantes ; et ce quelque soit la longueur du code.

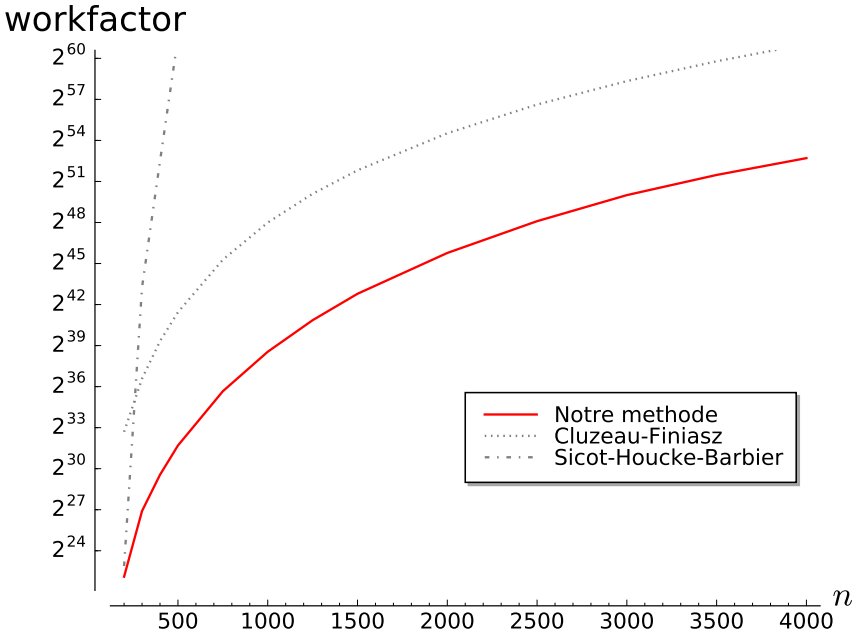


Figure 3.14 – Complexité pour trouver une équation de parité d'un code LDPC régulier de type $[n, n/2, 12, 6]$ avec $\tau = 0.005$.

3.4 Reconnaitre un code en résolvant LPN

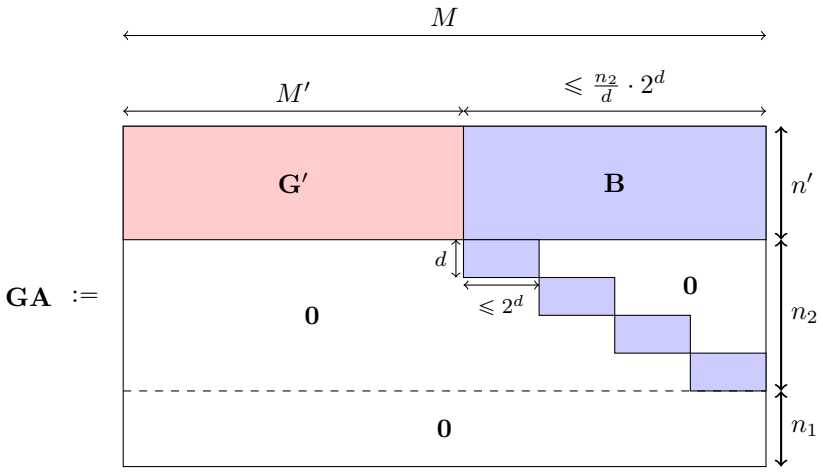
Nous allons voir à présent comment utiliser les méthodes de résolution du problème LPN de la sous-section 2.9 pour rechercher des équations de parité creuses dans un code.

Soit $\mathcal{C}$ un code linéaire $[n, k]$ inconnu et soient $\mathbf{y}_1, \cdots, \mathbf{y}_M$, M mots de code bruités par un canal binaire symétrique de probabilité d'erreur τ. Nous plaçons cette fois-ci nos mots de code bruités sur les colonnes d'une matrice de taille $n \times M$:

$$\mathbf{G} := \left[\mathbf{y}_1^\top | \cdots | \mathbf{y}_M^\top\right] \tag{3.19}$$

Soit $\mathbf{h} \in \mathbb{F}_2^n$ une équation de parité de poids w du code $\mathcal{C}$. Chaque bit de $\mathbf{hG}$ a une probabilité $\gamma := \frac{1-(1-2\tau)^w}{2}$ de valoir 1 (toujours via le *piling-up lemma*). Retrouver $\mathbf{h}$ revient donc à résoudre la problème LPN(n, γ) où l'oracle de LPN a produit M échantillons de la forme $(\mathbf{y}, < \mathbf{h}, \mathbf{y} > +e) = (\mathbf{y}, 0)$ avec $\mathbf{y} \in \mathbb{F}_2^n$ et e qui vaut 1 avec probabilité γ. Le problème LPN est même ici plus simple qu'un problème LPN classique car nous possédons une information importante sur le vecteur secret $\mathbf{h}$ que nous recherchons : il est de poids faible w.

De la même façon que dans la section 2.9, nous appliquons la méthode BKW [BKW03] pour produire une matrice inversible $\mathbf{A}$ de taille $M \times M$ telle que :



où $\mathbf{G}'$ est une matrice de taille $n' \times M'$ et $\mathbf{B}$ est une matrice de taille $n' \times (M - M')$ où $n' := n - n_1 - n_2$ et M' est une longueur qui dépend de l'élimination gaussienne. Les paramètres n_1, n_2 et d sont des paramètres à optimiser.

Le vecteur $\mathbf{h}' \in \mathbb{F}_2^{n'}$ constitué des n' premiers bits de l'équation de parité $\mathbf{h}$ vérifie :

$$\mathbf{h}'\mathbf{G}' = \mathbf{e}' \tag{3.20}$$

où les bits du vecteur d'erreur $\mathbf{e}'$ sont distribués selon une loi de Bernoulli de paramètre :

$$\gamma' := \frac{1 - (1 - 2\gamma)^{\frac{n_2}{d}}}{2} \tag{3.21}$$

donné par le *piling-up lemma*. Notons que le vecteur $\mathbf{h}'$ est de poids moyen $\frac{n' \cdot w}{n}$. Nous pouvons alors retrouver cette partie du vecteur secret en recherchant des sommes creuses de lignes de la matrice $\mathbf{G}'$. Pour cela, nous pouvons utiliser la technique utilisée dans la méthode de Stern (cf. sous-section 2.5.2 ou bien celle de Both et May (cf. sous-section 2.7.

Pour trouver le vecteur $\mathbf{e}'$ (et donc $\mathbf{h}'$) dans l'équation (3.20), nous pouvons aussi utiliser une méthode de décodage générique qui trouve un mot de code distant de $\simeq \gamma' M'$ du vecteur $\mathbf{0}$. Dans ce cas précis, le décodage générique ne corrige pas une erreur mais trouve un mot de poids faible dans le code.

Remarque 3.4.1. En prenant $n_1 = 0$ et $d = 1$, nous obtenons un l'algorithme équivalent à l'algorithme 3.3.1 dans le sens où il trouve exactement les mêmes équations de parité.

Chapitre 4

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 4.1, 4.2 et 4.3 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 4.4 et 4.5, 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 [CT17, CT19b].

4.1 Polynôme énumérateur de poids d'un code

Définition 4.1.1. *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 4.1.1. 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 4.1.2. *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 4.1.2.

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 4.1.3. *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.*

4.2 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 :

$$P_{\mathcal{C}}(X) \simeq \sum_{w=0}^{n} \frac{\binom{n}{w}}{q^n} \times q^k \times X^w = \sum_{w=0}^{n} \frac{\binom{n}{w}}{q^{n-k}} X^w \tag{4.1}$$

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 (4.1). De la même façon, le polynôme énumérateur de poids typique du code dual de $\mathcal{C}$ est :

$$P_{\mathcal{C}^\perp}(X) \simeq \sum_{w=0}^{n} \frac{\binom{n}{w}}{q^k} X^w \tag{4.2}$$

La figure 4.1 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 4.2.1. Si le code est de rendement $\frac{1}{2}$, alors $k = n - k$ et donc il s'en déduit que $P_{\mathcal{C}}(X) = P_{\mathcal{C}^\perp}(X)$.

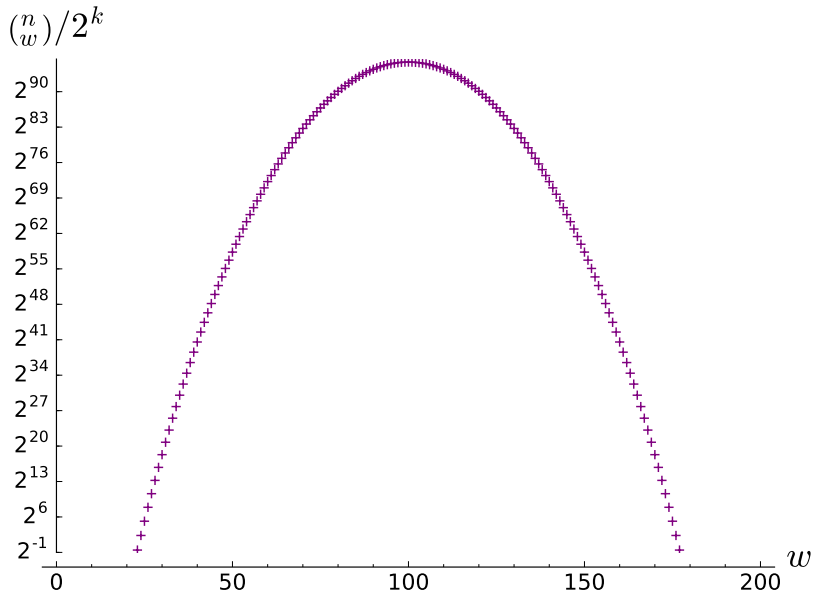


Figure 4.1 – Énumérateur de poids typique d'un code linéaire binaire aléatoire $\mathcal{C}[200, 100]_2$ ou de son dual.

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.

4.3 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 4.3.1 (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 4.3.1.

Une démonstration du théorème de MacWilliams est donnée dans le cours d'algèbre de Michel Demazure [Dem08]. □

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 4.3.2. *L'identité de MacWilliams :*

$$\begin{array}{rccc} \varphi & : & E_{n,R} & \longrightarrow & E_{n,1-R} \\ & & P_{\mathcal{C}} & \longmapsto & P_{\mathcal{C}^\perp} \end{array} \tag{4.3}$$

donnée par le théorème 4.3.1 est une application linéaire.

La proposition 4.3.2 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 :

$$\frac{(1+(q-1)X)^n}{q^1}\left(1+(q-1)\left(\frac{1-X}{1+(q-1)X}\right)^n\right) = \frac{(1+(q-1)X)^n + (q-1)(1-X)^n}{q} \tag{4.4}$$

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

4.4 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 [DRU06] 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 1.3. Nous rappelons leur définition :

Définition 4.4.1. *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 4.2.

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

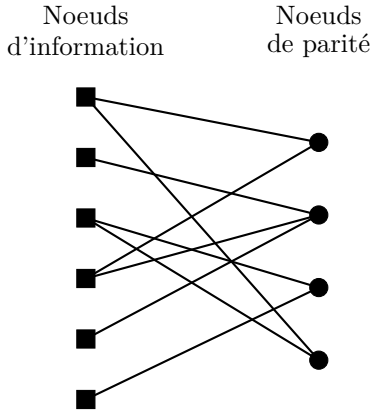


Figure 4.2 – Exemple de graphe de Tanner d'un code linéaire $[6,2]_2$.

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 4.3. Sur celle-ci, les nœuds de répétition sont représentés par le symbole $\circledcirc$ 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 $\circledcirc$ et les nœuds d'addition $\bigoplus$. Ces nœuds intermédiaires sont représentés par des disques noirs $\bullet$ sur la figure 4.3. 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.

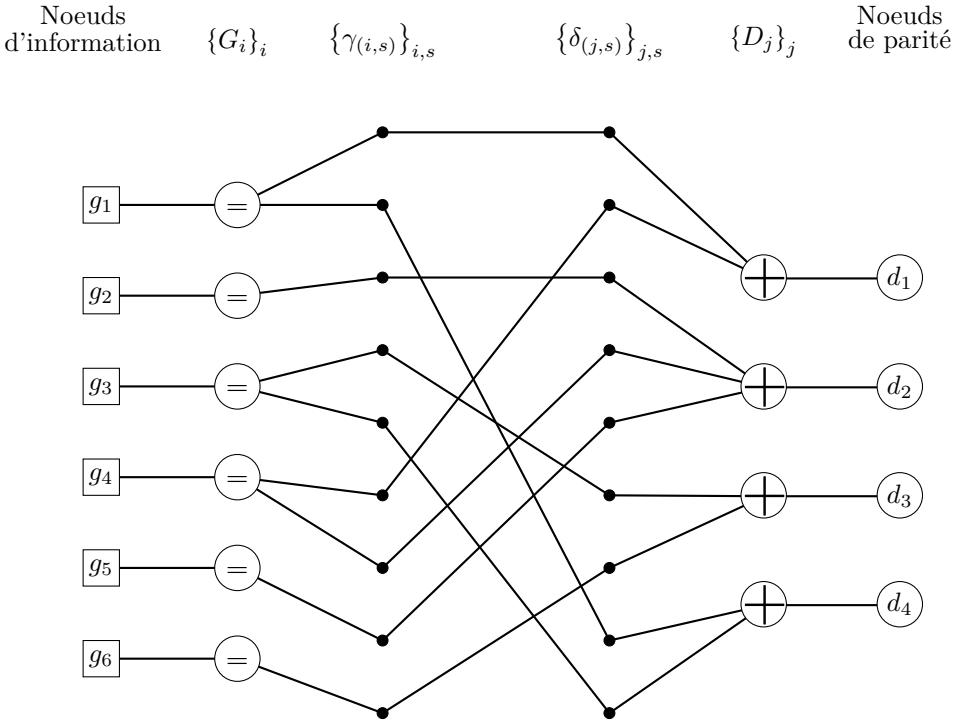
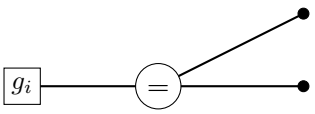


Figure 4.3 – Graphe normal de Tanner associé à la matrice de parité $\mathbf{H}_0$ de la figure 4.2.

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 :

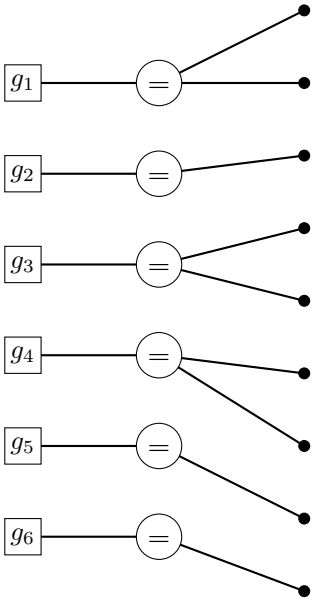
$$P_{\mathcal{C}_i^{\text{rep}}}(X) = 1 + X^{\alpha_i} \tag{4.5}$$

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 $\gamma_{(i,1)}$, $\cdots$, $\gamma_{(i,\alpha_i)}$:

$$P_{\mathcal{C}_i^{\text{rep}}}(X, Y) = 1 + XY^{\alpha_i} \tag{4.6}$$

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 $\mathcal{C}^{\text{rep}} := \mathcal{C}_1^{\text{rep}} \times \cdots \times \mathcal{C}_n^{\text{rep}}$.



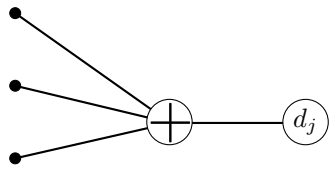
D'après la proposition 4.1.2, le polynôme énumérateur de poids du code $\mathcal{C}^{\text{rep}}$ est :

$$P_{\mathcal{C}^{\text{rep}}}(X,Y) = \prod_{i=1}^{n} P_{\mathcal{C}_i^{\text{rep}}}(X,Y) \tag{4.7}$$

$$= \prod_{i=1}^{n} (1 + XY^{\alpha_i}) \tag{4.8}$$

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 :

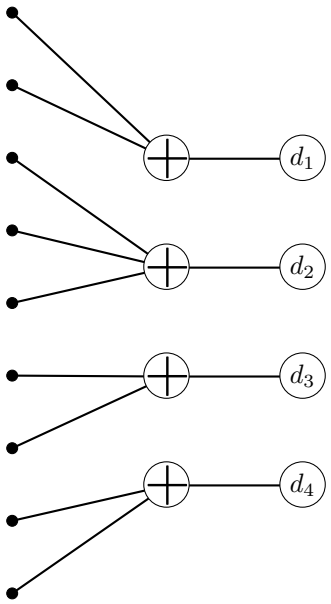
$$P_{\mathcal{C}_j^{\text{par}}}(X) = \frac{(1+X)^{\beta_j} + (1-X)^{\beta_j}}{2} \tag{4.9}$$

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 :

$$P_{\mathcal{C}_j^{\text{par}}}(Y,Z) = \frac{(1+Z)(1+Y)^{\beta_j} + (1-Z)(1-Y)^{\beta_j}}{2} \tag{4.10}$$

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 4.1.2, le polynôme énumérateur de poids du code $\mathcal{C}^{\text{par}}$ est :

$$P_{\mathcal{C}^{\text{par}}}(Y,Z) = \prod_{j=1}^{n-k} P_{\mathcal{C}_j^{\text{par}}}(Y,Z) \tag{4.11}$$

$$= \prod_{j=1}^{n-k} \left(\frac{(1+Z)(1+Y)^{\beta_j} + (1-Z)(1-Y)^{\beta_j}}{2} \right) \tag{4.12}$$

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 :

$$\frac{\text{coef}\left(P_{\mathcal{C}^{\text{par}}}(Y,Z)\ ;\ Y^\ell Z^0\right)}{\binom{N}{\ell}} = \frac{\text{coef}\left(P_{\mathcal{C}^{\text{par}}}(Y,0)\ ;\ Y^\ell\right)}{\binom{N}{\ell}} \quad \text{avec} \quad N := \sum_{i=1}^{n} \alpha_i = \sum_{j=1}^{n} \beta_j \tag{4.13}$$

Finalement, l'espérance A_w du nombre de mots de poids w de $\mathcal{C}$ est :

$$A_w = \sum_{\ell=0}^{N} \frac{\text{coef}\left(P_{\mathcal{C}^{\text{rep}}}(X,Y)\ ;\ X^w Y^\ell\right) \cdot \text{coef}\left(P_{\mathcal{C}^{\text{par}}}(Y,0)\ ;\ Y^\ell\right)}{\binom{N}{\ell}} \tag{4.14}$$

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 :

$$P_{\mathcal{C}^{\text{par}}}(Y,0) = \left(\frac{(1+Y)^{\beta}+(1-Y)^{\beta}}{2}\right)^{n-k} \tag{4.15}$$

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 :

$$P_{\mathcal{C}^{\text{rep}}}(X,Y) \;=\; \sum_{w=0}^{n}\binom{n}{w}X^{w}Y^{\alpha w} \tag{4.16}$$

et donc l'espérance A_ω devient :

$$A_w = \frac{\binom{n}{w}\cdot \text{coef}\left(P_{\mathcal{C}^{\text{par}}}(Y,0)\;;\; X^{\alpha w}\right)}{\binom{N}{\alpha w}} \tag{4.17}$$

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 4.3), 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 4.4 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.

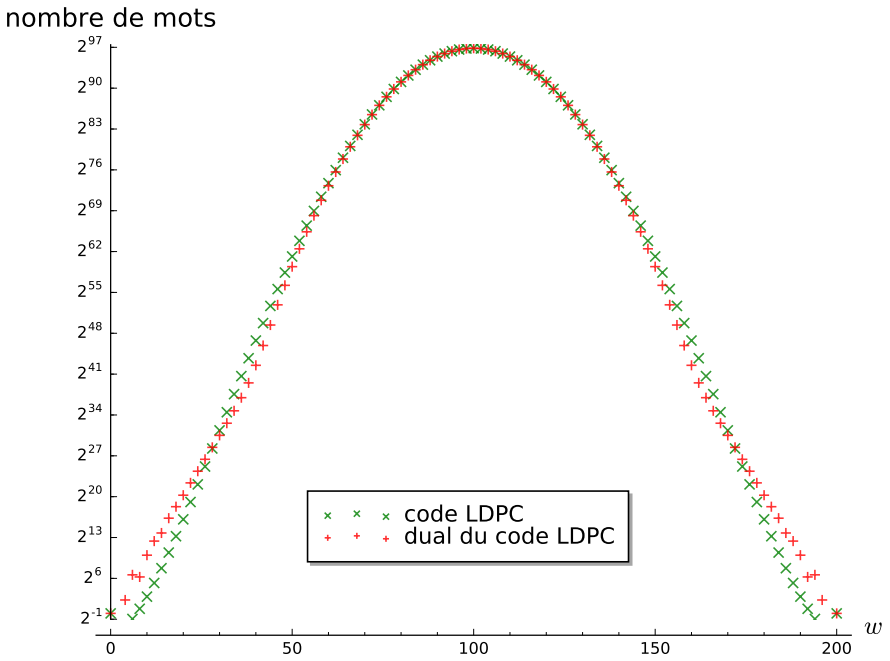


Figure 4.4 – Énumérateur de poids typique d'un code de $\Omega[200, 100, 3, 6]_2$ et de son dual.

Remarque 4.4.1. 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 4.4, les valeurs sont nulles une fois sur deux.

4.5 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 [For01], 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 $\circledcirc$ sont remplacés par des nœuds de parité $\bigoplus$ et inversement.

Les nœuds intermédiaires de degré 2 que nous avons représentés par des disques noirs $\bullet$ sont aussi des nœuds logiques puisque ce sont en fait des nœuds de répétition du même ordre que les nœuds représentés par le symbole $\circledcirc$. 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 4.5 représente le graphe normal de Tanner modélisant le code généré par la

matrice $\mathbf{H}_0$ de la figure 4.2.

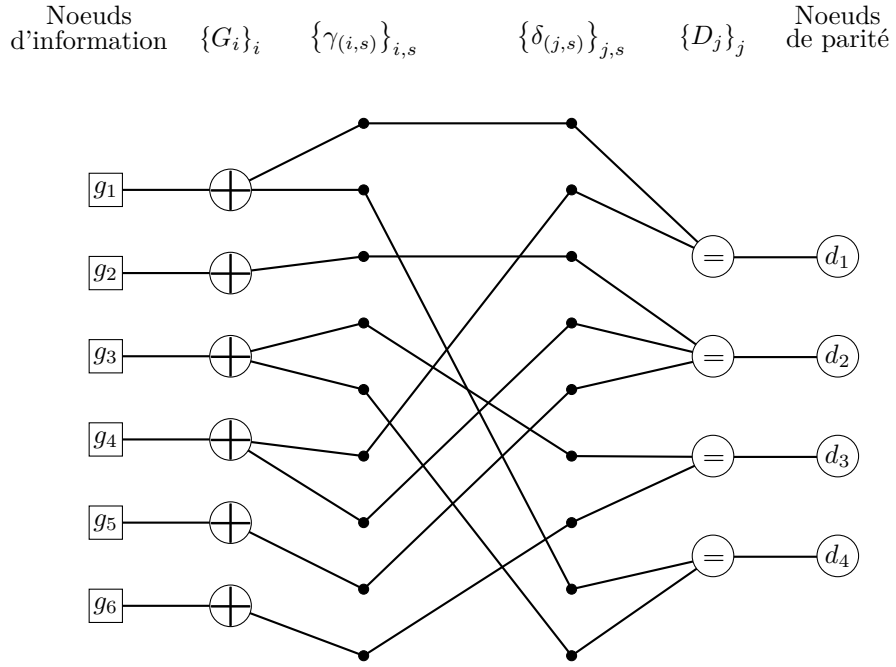
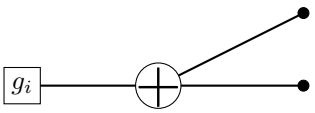


Figure 4.5 – Graphe normal de Tanner modélisant le code généré par la matrice $\mathbf{H}_0$ de la figure 4.2.

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 4.5.1. 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 $\mathcal{C}_i^{\text{par}}$.



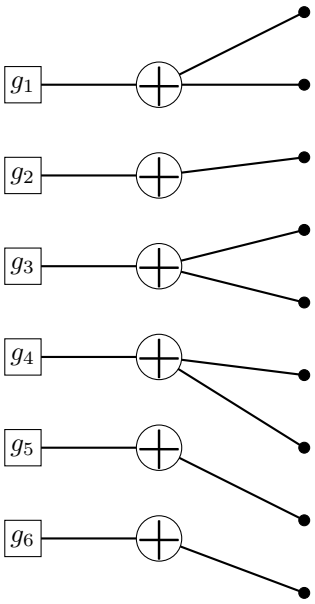
Un mot $(c_0, \cdots, c_{\alpha_i})$ de longueur $\alpha_i + 1$ est un mot du code $\mathcal{C}_i^{\text{par}}$ si et seulement si le nœud logique G_i est satisfait lorsque l'on affecte les valeurs c_0, c_1, $\cdots$, c_{α_i} aux nœuds respectifs g_i, $\gamma_{(i,1)}$, $\cdots$, $\gamma_{(i,\alpha_i)}$.

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

$$P_{\mathcal{C}_i^{\text{par}}}(X, Y) = \frac{(1+X)(1+Y)^{\alpha_i} + (1-X)(1-Y)^{\alpha_i}}{2} \tag{4.18}$$

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 $\mathcal{C}^{\text{par}} := \mathcal{C}_1^{\text{par}} \times \cdots \times \mathcal{C}_n^{\text{par}}$.



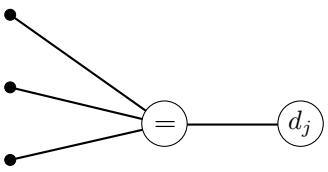
D'après la proposition 4.1.2, le polynôme énumérateur de poids du code $\mathcal{C}^{\text{par}}$ est :

$$P_{\mathcal{C}^{\text{par}}}(X,Y) = \prod_{i=1}^{n} P_{\mathcal{C}_i^{\text{par}}}(X,Y) \tag{4.19}$$

$$= \prod_{i=1}^{n} \frac{(1+X)(1+Y)^{\alpha_i} + (1-X)(1-Y)^{\alpha_i}}{2} \tag{4.20}$$

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 $\mathcal{C}_j^{\text{rep}}$.



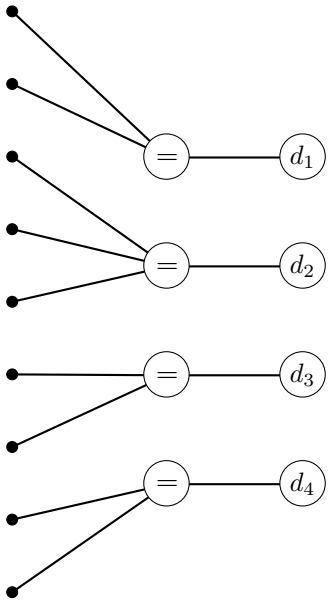
Un mot $(c_0, \cdots, c_{\beta_j})$ de longueur $\beta_j + 1$ est un mot du code $\mathcal{C}_j^{\text{rep}}$ si et seulement si le nœud logique D_j est satisfait lorsque l'on affecte les valeurs c_0, c_1, $\cdots$, c_{β_j} aux nœuds respectifs d_j, $\delta_{(j,1)}$, $\cdots$, $\delta_{(j,\beta_j)}$.

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

$$P_{\mathcal{C}_j^{\text{rep}}}(Y,Z) = 1 + ZY^{\beta_j} \tag{4.21}$$

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 4.1.2, le polynôme énumérateur de poids du code $\mathcal{C}^{\text{rep}}$ est :

$$\begin{aligned} P_{\mathcal{C}^{\text{rep}}}(Y,Z) &= \prod_{j=1}^{n-k} P_{\mathcal{C}_j^{\text{rep}}}(Y,Z) \\ &= \prod_{j=1}^{n-k} \left(1 + ZY^{\beta_j}\right) \end{aligned} \tag{4.22}$$

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 :

$$\frac{\text{coef}\left(P_{\mathcal{C}^{\text{rep}}}(Y,1)\ ;\ Y^\ell\right)}{\binom{N}{\ell}} \qquad \text{avec} \qquad N := \sum_{i=1}^{n}\alpha_i = \sum_{j=1}^{n}\beta_j \tag{4.23}$$

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 (4.23), 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 :

$$A_w = \sum_{\ell=0}^{N} \frac{\text{coef}\left(P_{\mathcal{C}^{\text{par}}}(X,Y)\ ;\ X^wY^\ell\right)\cdot\text{coef}\left(P_{\mathcal{C}^{\text{rep}}}(Y,1)\ ;\ Y^\ell\right)}{\binom{N}{\ell}} \tag{4.24}$$

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 :

$$P_{\mathcal{C}^{rep}}(Y,1) = \left(1+Y^{\beta}\right)^{n-k} \tag{4.25}$$

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 :

$$P_{\mathcal{C}^{\text{par}}}(X,Y) \;=\; \left(\frac{(1+X)(1+Y)^{\alpha}+(1-X)(1-Y)^{\alpha}}{2}\right)^{n} \tag{4.26}$$

Le théorème 4.5.1 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 4.5.1. *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 :

$$A_w = \sum_{\ell=0}^{N} \frac{\text{coef}\left(Q(X,Y)\;;\; X^wY^{\ell}\right)\cdot \text{coef}\left(P(Y)\;;\; Y^{\ell}\right)}{\binom{N}{\ell}} \tag{4.27}$$

avec :

$$N \;:=\; \sum_{i=1}^{n}\alpha_i = \sum_{j=1}^{n-k}\beta_j \tag{4.28}$$

$$P(Y) \;:=\; \prod_{j=1}^{n-k}\left(1+Y^{\beta_j}\right) \tag{4.29}$$

$$Q(X,Y) \;:=\; \prod_{i=1}^{n}\frac{(1+Y)^{\alpha_i}(1+X)+(1-Y)^{\alpha_i}(1-X)}{2} \tag{4.30}$$

Il est possible de déduire une forme asymptotique de l'énumérateur de poids du dual d'un code LDPC binaire à partir du théorème 4.5.1. Dans le théorème 4.5.2, 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 4.5.2. *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 :

$$A_w = \tilde{O}(2^{n\rho}) \tag{4.31}$$

avec :

$$\rho := \sup_{\lambda\in[0,\alpha]}\left(\log_2\left(f(x_2,y_2)\right)+(1-R)h_2\left(\tfrac{\lambda}{(1-R)\beta}\right)-\alpha h_2\left(\tfrac{\lambda}{\alpha}\right)\right) \tag{4.32}$$

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 4.5.2.

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 :

$$\begin{aligned} A_w &= \sum_{\ell=0}^{\alpha n} \frac{\operatorname{coef}\left((Q(X,Y))^n \ ; \ X^w Y^\ell\right) \cdot \operatorname{coef}\left((P(Y))^{n-k} \ ; \ Y^\ell\right)}{\binom{\alpha n}{\ell}} \\ &= \tilde{O}\left(\sup_{\lambda \in [0,\alpha]} \frac{\operatorname{coef}\left((Q(X,Y))^n \ ; \ X^{\lfloor \omega n \rceil} Y^{\lfloor \lambda n \rceil}\right) \cdot \operatorname{coef}\left((P(Y))^{n-k} \ ; \ Y^{\left\lfloor \frac{\lambda}{1-R}(n-k) \right\rfloor}\right)}{2^{\alpha n h_2\left(\frac{\lambda}{\alpha}\right)}} \right) \end{aligned}$$

avec $\quad P(Y) = 1 + Y^\beta \quad$ et $\quad Q(X,Y) = \dfrac{(1+Y)^\alpha (1+X) + (1-Y)^\alpha (1-X)}{2}$.

Les coefficients de P sont positifs. Ainsi, pour tout $y > 0$ et tout entier $s \geqslant 0$, on a :

$$(P(y))^{n-k} \geqslant \operatorname{coef}\left((P(Y))^{n-k} \ ; \ Y^s\right) \cdot y^s$$

et donc notamment :

$$\operatorname{coef}\left((P(Y))^{n-k} \ ; \ Y^{\left\lfloor \frac{\lambda}{1-R}(n-k) \right\rfloor}\right) \leqslant \left(\inf_{y>0} \left(\frac{P(y)}{y^{\frac{\lambda}{1-R}}} \right) \right)^{n-k}$$

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 :

$$\begin{aligned} \operatorname{coef}\left((P(Y))^{n-k} \ ; \ Y^{\left\lfloor \frac{\lambda}{1-R}(n-k) \right\rfloor}\right) &= \tilde{O}\left(\left(\frac{P(y_1)}{y_1^{\frac{\lambda}{1-R}}} \right)^{n-k} \right) \\ &= \tilde{O}\left(\left(\left(\frac{\beta(1-R)}{\beta(1-R)-\lambda} \right) \cdot \left(\frac{\beta(1-R)-\lambda}{\lambda} \right)^{\frac{\lambda}{(1-R)\beta}} \right)^{n-k} \right) \\ &= \tilde{O}\left(2^{(n-k) h_2\left(\frac{\lambda}{(1-R)\beta}\right)} \right) \end{aligned}$$

Déterminons maintenant une approximation asymptotique de $\operatorname{coef}\left((Q(X,Y))^n \ ; \ X^w Y^\ell\right)$. Les coefficients de Q sont tous positifs. Ainsi, pour tous $x, y > 0$ et pour tous entiers s et t, on a :

$$(Q(X,Y))^n \geqslant \operatorname{coef}\left((Q(X,Y))^n \ ; \ X^s Y^t\right) \cdot X^s Y^t$$

et donc notamment :

$$\operatorname{coef}\left((Q(X,Y))^n \ ; \ X^{\lfloor \omega n \rceil} Y^{\lfloor \lambda n \rceil}\right) \leqslant \left(\inf_{x,y>0} \left(\frac{Q(x,y)}{x^\omega y^\lambda} \right) \right)^n$$

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 [GS91] et ceux de Good dans [Goo57] montrent que lorsque n tend vers l'infini, la borne du théorème est fine.

□

Remarque 4.5.2. Dans le théorème 4.5.2, 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 5

La recherche de presque-collisions

Le problème de recherche de presque-collisions s'énonce ainsi : étant données deux listes $\mathscr{L}_1$ et $\mathscr{L}_2$ d'éléments d'un espace métrique et une distance w, trouver tous les couples à distance au plus w dans $\mathscr{L}_1 \times \mathscr{L}_2$. Nous avons déjà eu affaire à ce problème dans les chapitres précédents mais nous n'avons pas proposé de méthode efficace pour le résoudre.

De nombreuses variantes de ce problème existent. Dans certaines versions, la deuxième liste est traitée comme un flux : c'est-à-dire que nous recevons les éléments de $\mathscr{L}_2$ un par un et qu'à chaque élément reçu (la requête), nous devons trouver les éléments de $\mathscr{L}_1$ qui lui sont proches. Dans la section 5.1, nous décrivons ces problèmes en distinguant le cas où l'on recherche l'élément le plus proche de la requête du cas où l'on cherche un ensemble d'éléments de $\mathscr{L}_1$ à distance au plus w de la requête. Ces problèmes sont beaucoup étudiés dans la littérature et certaines des méthodes de recherche de presque-collisions que nous décrirons par la suite s'inspirent de méthodes pour résoudre des problèmes de voisins proches.

Le problème de recherche de presque-collisions est fondamentalement lié à la façon dont sont produites les listes $\mathscr{L}_1$ et $\mathscr{L}_2$. Dans la section 5.2, nous présentons différents modèles de proximité qui décrivent chacun une distribution particulière des éléments de $\mathscr{L}_1 \times \mathscr{L}_2$. Nous traiterons alors la recherche de presque-collisions dans les différents modèles. En outre, nous verrons dans cette même section que le problème de recherche de presque-collisions dépasse très largement le simple cadre du problème de décodage générique ou de reconnaissance de code.

Dans la section 5.3.1, nous présentons l'approche LSH (*Locality Sensitive Hashing*) qui est l'une des approches les plus connues pour résoudre le problème de presque-collisions. Celle-ci consiste à utiliser des fonctions de hachage *floues* qui ont la propriété de relier des vecteurs proches avec une bonne probabilité. Dans cette section, nous clarifions l'analyse de l'approche LSH que l'on peut trouver dans la littérature. Nous veillons notamment à bien distinguer les modèles de proximité considérés.

Dans la section 5.5, nous présentons une méthode que nous avons déjà évoquée dans le chapitre 2 : la méthode de May et Ozerov [MO15]. Celle-ci est probablement la plus considérée dans le domaine de la cryptographie fondée sur les codes. Elle a permis d'améliorer significativement les méthodes de décodages génériques.

Enfin, dans la section 5.6, nous décrivons une méthode utilisant des codes correcteurs pour construire des fonctions de hachages floues. Nous présentons l'analyse de [GMO10] bien qu'elle ne soit pas pertinente dans le cadre du décodage générique. Cependant, leur algorithme a inspiré une partie de nos travaux. Cette section viendra clore notre état de l'art sur la recherche de presque-collisions. Les chapitres 6 à 9 auront pour but de proposer de nouvelles méthodes pour résoudre ce problème plus efficacement.

5.1 La recherche de voisins proches

Le problème de recherche du plus proche voisin (ou *nearest-neighbors search problem*) est un problème qui a plus de 50 ans. Il peut être rangé dans la catégorie des problèmes d'optimisation. Les applications de ce problème (ou de ses variantes) sont extrêmement nombreuses en informatique. Il se retrouve par exemple en apprentissage automatique (*machine learning*), reconnaissance de formes (*pattern recognition*), fouille dans des données multimédia, compression de données, calculs statistiques ou encore en théorie des codes correcteurs d'erreurs [DHS00, SDI05, Bis06, GG91, DW82, AI17, MO15, CT17]. Précisons que cette liste est loin d'être exhaustive.

5.1.1 La recherche du plus proche voisin

Minsky et Papert sont parmis les premiers à formaliser le problème de recherche du plus proche voisin dans [MP69, partie III chapitre 12]. Ils se sont tout d'abord intéressés au problème de recherche de collisions exactes (en anglais, *exact matches*) qu'ils formalisent ainsi :

Problème 5.1.1 (recherche de collisions exactes)**.** *Soit $\mathscr{E}$ un ensemble. Étant donné $\mathscr{L} \subseteq \mathscr{E}$, construire une structure de données permettant de répondre efficacement à la question suivante : étant donnée une requête $\mathbf{x} \in \mathscr{E}$, est-ce que $\mathbf{x}$ se trouve dans $\mathscr{L}$?*

Une façon simple de résoudre ce problème est de ranger les éléments de $\mathscr{L}$ dans une table de hachage. Cette phase de pré-calcul a une complexité en temps et en mémoire de l'ordre de $O(\#\mathscr{L})$ et la structure de données ainsi engendrée permet de résoudre n'importe quelle requête en un temps de l'ordre de $O(1)$. En effet, la résolution d'une requête consiste simplement à vérifier que la table de hachage contient un élément à l'adresse donnée par le haché de la requête. Cet algorithme n'est efficace que si les éléments de $\mathscr{L}$ sont répartis uniformément dans la table de hachage. Une bonne fonction de hachage est donc ici une fonction dont la sortie est indistinguable d'un tirage uniforme sur l'ensemble des adresses de la table ; c'est aussi une propriété souvent recherchée pour les fonctions de hachage utilisées en cryptographie.

Minsky et Papert posent alors un problème légèrement différent : ils se demandent se qu'il se passerait si, au lieu de chercher une collision exacte, nous cherchions l'élément de $\mathscr{L}$ qui "correspond le mieux" (*best match*) ; c'est-à-dire le plus proche de la requête $\mathbf{x}$. Tout d'abord, ce problème n'a de sens que si $\mathscr{E}$ est un espace métrique. Minsky et Papert constatent que ce nouveau problème est fondamentalement différent du premier et qu'il est loin d'être trivial, notamment lorsque $\mathscr{E}$ est un espace vectoriel de grande dimension. Ils conjecturent aussi un impact significatif sur la mémoire.

Dans [IM98], Indyk et Motwani s'intéressent au problème de recherche du plus proche voisin (en anglais, *nearest neighbor search problem*) qui est essentiellement le problème sus-cité et que nous reformulons ainsi :

Problème 5.1.2 (recherche du plus proche voisin)**.** *Soit $(\mathscr{E}, d)$ un espace métrique. Étant donné $\mathscr{L} \subseteq \mathscr{E}$, construire une structure de données permettant d'effectuer efficacement l'opération suivante : étant donnée une requête $\mathbf{x} \in \mathscr{E}$, trouver $\mathbf{y} \in \mathscr{L}$ qui minimise $\Delta(\mathbf{x}, \mathbf{y})$.*

Tout comme pour le problème 5.1.1, le problème 5.1.2 se décompose en deux phases de calcul : le pré-calcul générant la structure de données puis le traitement des requêtes. En rangeant les éléments de $\mathscr{L}$ dans une liste, nous obtenons une structure de données triviale qui nous permet de résoudre chaque requête en un temps linéaire en la taille de $\mathscr{L}$. Cette méthode correspond en fait à effectuer une simple recherche exhaustive. Cette solution n'est évidemment pas satisfaisante. Nous souhaitons trouver une solution qui nous permette de résoudre chaque requête en un temps sous-linéaire en $\#\mathscr{L}$; c'est-à-dire en un temps de l'ordre de $O(L^\rho)$ où $L = O(\#\mathscr{L})$ et $\rho \in [0, 1]$. Bien sûr, la phase de pré-calcul ne

devra pas être trop coûteuse et elle devra produire une structure de données qui ne soit pas trop gourmande en mémoire.

5.1.2 Le fléau de la dimension

Observons le cas où $\mathscr{E}$ est la droite réelle $\mathbb{R}$. Le problème du plus proche voisin est alors simple à résoudre : il suffit d'ordonner les éléments de $\mathscr{L}$; un parcours dichotomique permet alors de résoudre une requête. Cette solution a une complexité de l'ordre de $O(L\log(L))$ pour le pré-calcul et $O(\log(L))$ pour la résolution d'une requête. Cependant, si nous augmentons la dimension de $\mathscr{E}$ en se plaçant maintenant dans le plan euclidien $\mathbb{R}^2$ ou même plus généralement dans l'espace euclidien $\mathbb{R}^n$ alors le problème semble plus difficile. On parle de "fléau de la dimension" (en anglais, *curse of dimensionality*). Pour se convaincre de ce phénomène, nous proposons une résolution du problème du plus proche voisin en construisant une structure de données idéale basée sur les diagrammes de Voronoï. Pour rappel, le diagramme de Voronoï $\mathscr{V}$ engendré par l'ensemble discret $\mathscr{S}$ est :

$$\mathscr{V} := \{\mathscr{V}_{\mathbf{x}}\}_{\mathbf{x}\in\mathscr{S}} \tag{5.1}$$

où pour tout $\mathbf{x} \in \mathscr{S}$:

$$\mathscr{V}_{\mathbf{x}} := \{\mathbf{u} \in \mathbb{R}^n : \forall \mathbf{y} \in \mathscr{L}, \Delta(\mathbf{u},\mathbf{x}) \leqslant \Delta(\mathbf{u},\mathbf{y})\} \tag{5.2}$$

avec Δ la fonction de distance euclidienne. Résoudre une requête du problème du plus proche voisin sur une liste $\mathscr{L} \subseteq \mathbb{R}^n$ consiste alors à localiser un point dans le diagramme de Voronoï engendré par $\mathscr{L}$. Cependant, la description d'une région de Voronoï est exponentielle en la dimension. L'utilisation de tels diagrammes pour la recherche du plus proche voisin est donc éventuellement adaptée lorsque la dimension de $\mathscr{E}$ est faible mais cette solution n'est plus pertinente pour des grandes dimensions.

Les diagrammes de Voronoï se généralisent à d'autres espaces multidimensionnels tels que la sphère unitaire euclidienne $\mathscr{S}_1^n := \{\mathbf{x} \in \mathbb{R}^n : ||\mathbf{x}|| = 1\}$ ou l'espace de Hamming $\mathbb{F}_q^n$. Le fléau de la dimension est donc un problème qui dépasse le cas des espaces euclidiens.

Par la suite, nous nous intéresserons essentiellement à des espaces $\mathscr{E}$ concernés par le fléau de la dimension. Nos espaces métriques $\mathscr{E}$ seront alors naturellement associés à une notion de dimension que nous noterons toujours n. Pour les applications cryptographiques que nous considérons, les listes $\mathscr{L}$ sont de taille exponentielle en la dimension n. Nous étudions particulièrement les cas asymptotiques où n tend vers l'infini. Notons alors que si la résolution d'une requête coûte $O(L^\rho)$ avec $\rho \in]0,1[$ alors, bien que ce coût soit sous-linéaire en L, celui-ci restera exponentiel en n.

5.1.3 Un voisin proche plutôt que le plus proche

Il est plus facile d'imaginer une méthode résolvant le problème de recherche du plus proche voisin lorsque nous avons une idée de la distance qui sépare l'élément recherché de la requête. Un résultat important du papier [IM98] est la réduction du problème 5.1.2 au problème de recherche d'un voisin proche (ou *near-neighbor search problem*) que nous définissons comme suit :

Problème 5.1.3 (recherche d'un voisin proche)**.** *Soit $(\mathscr{E}, d)$ un espace métrique. Étant donnés $\mathscr{L} \subseteq \mathscr{E}$ et une distance w, construire une structure de données permettant d'effectuer efficacement l'opération suivante : étant donnée une requête $\mathbf{x} \in \mathscr{E}$, trouver un élément $\mathbf{y} \in \mathscr{L}$ tel que $\Delta(\mathbf{x},\mathbf{y}) \leqslant w$.*

La recherche du plus proche voisin consiste alors essentiellement à résoudre le problème de recherche d'un voisin proche pour une distance w judicieusement choisie. Toutefois, le problème 5.1.3 trouve un intérêt qui dépasse le simple fait d'être une réduction du

problème 5.1.2. En effet, la recherche d'un voisin proche a au moins autant d'applications que la recherche du plus proche voisin.

Enfin, une variante du problème de recherche d'un voisin proche est la recherche de **tous** les voisins proches. Dans cette variante, lorsqu'une requête $\mathbf{x} \in \mathscr{E}$ est donnée, nous recherchons l'ensemble $\{\mathbf{y} \in \mathscr{L} : \Delta(\mathbf{x}, \mathbf{y}) \leqslant w\}$ et non plus seulement un élément de cet ensemble. Le problème se formalise alors ainsi :

Problème 5.1.4 (recherche des voisins proches)**.** *Soit $(\mathscr{E}, d)$ un espace métrique. Étant donnés $\mathscr{L} \subseteq \mathscr{E}$ et une distance w, construire une structure de données permettant d'effectuer efficacement l'opération suivante : étant donnée une requête $\mathbf{x} \in \mathscr{E}$, trouver tous les éléments $\mathbf{y} \in \mathscr{L}$ tels que $\Delta(\mathbf{x}, \mathbf{y}) \leqslant w$.*

Souvent, un algorithme probabiliste permettant de trouver un seul voisin proche permettra aussi de trouver tous les voisins proches avec une bonne probabilité s'il est itéré suffisamment de fois.

5.2 La recherche de presque-collisions

5.2.1 Définition du problème

Une variante du problème de recherche des voisins proches est le problème de recherche des presque-collisions (en anglais, *near-collisions search problem*). Dans les problèmes que nous avons introduits dans la section précédente, les requêtes sont données sous forme de flux et sont donc traitées les unes après les autres. Dans le problème de recherche de presque-collisions, il n'y a plus de requêtes à proprement parler, ou alors celles-ci sont données en une seule fois et sont une donnée d'entrée du problème. En d'autres termes, nous avons deux listes $\mathscr{L}_1$ et $\mathscr{L}_2$ et nous cherchons à trouver des couples d'éléments proches dans $\mathscr{L}_1 \times \mathscr{L}_2$:

Problème 5.2.1 (recherche de presque-collisions)**.** *Soit $(\mathscr{E}, d)$ un espace métrique. Étant donnés $\mathscr{L}_1, \mathscr{L}_2 \subseteq \mathscr{E}$ et une distance w, trouver tous les couples $(\mathbf{x}, \mathbf{y}) \in \mathscr{L}_1 \times \mathscr{L}_2$ tels que $\Delta(\mathbf{x}, \mathbf{y}) \leqslant w$.*

Notons que le problème où nous cherchons des couples d'éléments proches dans une seule et même liste $\mathscr{L}$ est une instance particulière du problème 5.2.1. En effet, il suffira de poser $\mathscr{L}_1 = \mathscr{L}_2 := \mathscr{L}$. Les algorithmes que nous proposerons par la suite s'adapteront ainsi trivialement à cette instance.

D'autre part, rappelons que les instances du problème de recherche de presque-collisions qui nous intéressent sont celles où $\mathscr{E}$ est de grande dimension (nous avons supposé que $\mathscr{E}$ est muni d'une notion de dimension) et où les tailles de $\mathscr{L}_1$ et $\mathscr{L}_2$ sont exponentielles en cette dimension. En outre, les algorithmes que nous proposerons seront essentiellement probabilistes : nous considérerons alors que le problème est résolu si nous avons trouvé toutes les presque-collisions avec une probabilité $\Omega(1)$.

Lorsque $w = 0$, le problème consiste à rechercher des collisions exactes dans $\mathscr{L}_1 \times \mathscr{L}_2$. Ce problème est d'une importance majeure notamment pour étudier les propriétés cryptographiques des fonctions de hachage. Pour résoudre cette instance du problème, il suffit de traiter chaque élément de $\mathscr{L}_2$ comme une requête du problème 5.1.1 où la liste en entrée est $\mathscr{L}_1$. Ainsi, en utilisant la méthode des tables de hachage décrite au début de la section précédente, nous pouvons résoudre le problème 5.2.1 avec une complexité en temps et en mémoire de l'ordre de $O(\#\mathscr{L}_1 + \#\mathscr{L}_2)$ lorsque $w = 0$. De façon générale, nous ne pourrons jamais résoudre le problème 5.2.1 avec une complexité en temps et mémoire strictement inférieure à $O(\#\mathscr{L}_1 + \#\mathscr{L}_2)$ puisqu'il faudra toujours au moins stocker et lire

une fois les listes $\mathscr{L}_1$ et $\mathscr{L}_2$. Nous avons donc un algorithme optimal lorsque $w = 0$; mais pour d'autres valeurs de w, la tache n'est pas aussi aisée.

Nous verrons par la suite que le cas où $w > 0$ a aussi de nombreuses applications. Une recherche exhaustive de l'ensemble des presque-collisions dans $\mathscr{L}_1 \times \mathscr{L}_2$ coûte $O(\#\mathscr{L}_1 + \#\mathscr{L}_2)$ en mémoire et $O(\#\mathscr{L}_1 \times \#\mathscr{L}_2)$ en temps. Ainsi, lorsque les tailles des listes $\mathscr{L}_1$ et $\mathscr{L}_2$ sont du même ordre $O(L)$, l'objectif est de trouver un algorithme sous-quadratique en L ; c'est-à-dire un algorithme qui a une complexité en temps de l'ordre de $O(L^\rho)$ où $\rho < 2$.

Pour simplifier notre propos, nous supposerons toujours par la suite que les tailles des listes $\mathscr{L}_1$ et $\mathscr{L}_2$ sont du même ordre. On pose alors L tel que $\#\mathscr{L}_i = O(L)$ pour $i \in \{1, 2\}$.

Remarquons à ce stade qu'une méthode qui pré-calcule la structure de données du problème 5.1.4 des voisins proches en un temps de l'ordre de $O\left(L^{1+\rho}\right)$ et qui résout chaque requête en un temps de l'ordre de $O(L^\rho)$ permet de construire un algorithme qui résout le problème 5.2.1 de recherche de presque-collisions en temps $O\left(L^{1+\rho}\right)$. L'algorithme sous-entendu ici effectue une recherche des voisins proches dans la liste $\mathscr{L}_1$ pour chaque élément de la liste $\mathscr{L}_2$ (les éléments de $\mathscr{L}_2$ sont donc les requêtes). Toutefois, cette façon de procéder n'utilise pas toujours la mémoire de façon optimale. Par exemple, supposons qu'un algorithme probabiliste produise une structure de données d'une taille de l'ordre de $O(T)$ qui permette de résoudre une requête des voisins proches avec une probabilité p sur une liste de taille $O(L)$. Alors, ce même algorithme itéré $\frac{1}{p}$ fois permettra de construire une structure de données d'une taille de l'ordre de $O\left(\frac{T}{p}\right)$ qui résoudra une requête des proches voisins avec une probabilité $\Omega(1)$. Puisque les requêtes sont traitées les une après les autres de façon indépendante, la structure de données globale doit contenir toutes les structures de données produites à chaque itération de l'algorithme ; c'est en fait l'union des structures de données intermédiaires. Cependant, pour résoudre le problème des presque-collisions sur deux listes de taille $O(L)$, il n'est pas nécessaire de garder simultanément en mémoire toutes les structures de données puisqu'à chaque itération de l'algorithme nous pouvons traiter l'ensemble des requêtes (qui sont en fait les éléments de la seconde liste) puis oublier la structure de données de l'itération courante. Ainsi, cette méthode trouvera toutes les presque-collisions avec une probabilité $\Omega(1)$ en utilisant $O\left(\frac{1}{p}\right)$ fois moins de mémoire.

Par la suite, nous traiterons directement le problème des presque-collisions sans passer par le problème des voisins proches. Il faudra donc bien garder à l'esprit que nos algorithmes peuvent avoir un impact différent sur la mémoire selon qu'ils sont appliqués au problème des presque-collisions ou au problème des voisins proches.

5.2.2 Les presque-collisions en cryptographie

La recherche de presque-collisions apparaît dans de nombreux domaines. Nous nous contenterons de donner ici quelques exemples en lien avec la cryptologie. Par exemple, ce problème apparaît dans l'étude des réseaux euclidiens. La cryptographie basée sur les réseaux euclidiens repose notamment sur la difficulté à trouver le vecteur le plus court dans un réseaux euclidien lorsque ce dernier est de dimension élevée et lorsqu'il est donné par une base aléatoire ; ce problème est aussi connu sous l'acronyme SVP ou *Shortest Vector Problem*. Or il est possible d'accélérer la résolution du problème SVP en effectuant récursivement des recherches de presque-collisions dans des sphères euclidiennes via des techniques dites de "tamisage" (en anglais, *sieving*) [BGJ15, BDGL15, Laa15, BDGL16].

D'autre part, la recherche efficace de presque-collisions dans la métrique euclidienne a aussi permis la mise en place d'une attaque [dBDJdW18] contre les crypto-systèmes asymétriques de Mersenne [AJPS17].

Toujours dans le domaine de la cryptographie, une autre instance du problème 5.2.1 qui nous intéresse est celle où $\mathscr{E}$ est l'espace vectoriel $\mathbb{F}_2^n$ muni de la métrique de Hamming. Rappelons que la distance de Hamming entre deux vecteurs binaires est le nombre de

positions sur lesquelles ils diffèrent. Cette instance particulière du problème 5.2.1 apparaît par exemple lorsque l'on veut trouver des presque-collisions dans une fonction de hachage [LMRS12, Leu13]. Dans ce contexte, trouver un algorithme efficace pour rechercher des presque-collisions dans $\mathscr{L}^2$ lorsque $\mathscr{L}$ est un ensemble de hachés discréditerait la fonction de hachage qui a généré $\mathscr{L}$ en tant que *bonne* fonction de hachage cryptographique. Une conséquence encore plus pratique de tels algorithmes est qu'ils peuvent permettre de trouver des presque-collisions dans les fonctions de compression utilisées dans les fonctions de hachage pour les convertir en collisions pour la fonction de hachage elle-même [LMRS12, Sec. 2.3].

La communauté de cryptographie basée sur les codes correcteurs d'erreurs (en anglais, *code-based cryptography*) porte également un intérêt particulier pour le problème de recherche de presque-collisions dans les espaces métriques binaires de Hamming. Nous verrons dans le chapitre suivant que ce problème est étroitement lié au problème du décodage générique de codes linéaires. Dans ce contexte, May et Ozerov ont proposé un algorithme qui, à ce jour, a la meilleure complexité en temps au moins asymptotiquement lorsque la dimension du problème augmente [MO15]. D'autres s'intéressent aussi à une généralisation du problème de recherche de presque-collisions sur $\mathbb{F}_2^n$; à savoir le problème k-liste approché (en anglais, *approximate k-list*) [BM17a]. Dans ce problème, on ne cherche pas seulement des couples proches dans le produit cartésien de deux listes mais plus généralement des k-uplets de vecteurs dans le produit cartésien de k listes tels que leur somme soit creuse. Un algorithme efficace pour résoudre le problème de recherche de presque-collisions permettrait alors d'accélérer les algorithmes de résolution du problème k-liste approché.

En outre, la cryptographie basée sur les codes étend notre intérêt pour le problème de recherche de presque-collisions à d'autres espaces métriques tels que les espaces métriques de Hamming non-binaires [Hir16, GKH17] ou encore à la métrique rang.

5.2.3 Les modèles de proximité

Dans les problèmes de recherche de voisins proches de la section 5.1, nous ne savons pas comment sont produites la liste $\mathscr{L}$ et les requêtes $\mathbf{x}$. Dans le pire cas, il est difficile de trouver les éléments à distance au plus w (avec $w := \min\{\Delta(\mathbf{x}, \mathbf{y}) : y \in \mathscr{L}\}$ dans le problème 5.1.2) à cause de la présence d'une multitude d'éléments à distance proche de w. Dans les articles auxquels nous faisons référence, ces problèmes réclament de traiter tous les éléments de la liste $\mathscr{L}$; la résolution d'une requête ne peut donc pas être sous-linéaire. C'est pourquoi, dans la littérature, sont introduites des versions plus "souples" des problèmes 5.1.2 et 5.1.3. En anglais, ces problèmes s'intitulent *approximate nearest neighbor* et *approximate near neighbor search problems* que nous traduirons respectivement par le problème approximatif du plus proche voisin et le problème approximatif d'un voisin proche. Dans ces nouveaux problèmes, il ne s'agit plus de rechercher l'élément de $\mathscr{L}$ le plus proche ou distant d'au plus w de la requête $\mathbf{x}$, mais un élément qui en soit au plus $(1+\varepsilon)$ fois plus éloigné où $\varepsilon > 0$ est une constante du problème. Les problèmes 5.2.2 et 5.2.3 les formalisent et le papier [IM98] montre que l'un se réduit à l'autre de la même façon que le problème 5.1.2 se réduit au problème 5.1.3.

Problème 5.2.2 (recherche approximative du plus proche voisin)**.** *Soit $(\mathscr{E}, d)$ un espace métrique. Étant donnés $\mathscr{L} \subseteq \mathscr{E}$ et une constante d'approximation $\varepsilon > 0$, construire une structure de données permettant d'effectuer efficacement l'opération suivante : étant donnée une requête $\mathbf{x} \in \mathscr{E}$, trouver $\mathbf{y} \in \mathscr{L}$ tel que $\Delta(\mathbf{x}, \mathbf{y}) < (1+\varepsilon) \cdot \Delta(\mathbf{x}, \mathscr{L})$.*

Problème 5.2.3 (recherche approximative d'un voisin proche)**.** *Soit $(\mathscr{E}, d)$ un espace métrique. Étant donnés $\mathscr{L} \subseteq \mathscr{E}$, une distance w et une constante d'approximation $\varepsilon > 0$,*

construire une structure de données permettant d'effectuer efficacement l'opération suivante : étant donnée une requête $\mathbf{x} \in \mathscr{E}$*,*

- *s'il existe* $\mathbf{y} \in \mathscr{L}$ *tel que* $\Delta(\mathbf{x}, \mathbf{y}) \leqslant w$ *alors retourner* $\mathbf{y}' \in \mathscr{L}$ *tel que* $\Delta(\mathbf{x}, \mathbf{y}') < (1+\varepsilon)w$ *;*
- *si pour tout* $\mathbf{y} \in \mathscr{L}$*,* $\Delta(\mathbf{x}, \mathbf{y}) \geqslant (1+\varepsilon)w$ *alors retourner* $\varnothing$ *;*
- *s'il existe* $\mathbf{y} \in \mathscr{L}$ *tel que* $\Delta(\mathbf{x}, \mathbf{y}) < (1+\varepsilon)w$ *et si pour tout* $\mathbf{y} \in \mathscr{L}$*,* $\Delta(\mathbf{x}, \mathbf{y}) > w$ *alors retourner* $\mathbf{y}' \in \mathscr{L}$ *tel que* $\Delta(\mathbf{x}, \mathbf{y}') < (1+\varepsilon)w$ *ou bien retourner* $\varnothing$*.*

Considérer les problèmes 5.2.2 et 5.2.3 plutôt que les problèmes 5.1.2 et 5.1.3 permet souvent d'obtenir de meilleures performances en pratique. Toutefois, dans le dernier problème, il est en fait supposé que les requêtes sont "presque" toutes à distance au moins $(1+\varepsilon)w$. Bien que ce modèle soit souvent celui étudié dans la littérature, ce n'est pas toujours le plus pertinent. Le modèle qui nous intéressera davantage par la suite consiste à supposer que les requêtes sont obtenues par un tirage aléatoire dans l'espace ambiant. La difficulté du problème est alors intrinsèquement liée à la loi de probabilité utilisée pour ce tirage. Par exemple, cette loi pourra être une simple loi uniforme sur $\mathscr{E}$ (si celle-ci existe). Posons alors W la distance typique entre un élément de $\mathscr{L}$ et une requête. Dès lors que w est strictement plus petit que W, les éléments proches se distingueront en moyenne des autres éléments.

Nous formalisons les deux modèles sus-cités pour le problème de recherche de presque-collisions :

Définition 5.2.4 (modèle de proximité $\mathcal{M}_{\mathrm{IT}}(\mathscr{E}, L, r)$)**.** *Dans le modèle* $\mathcal{M}_{IT}(\mathscr{E}, L, r)$ *d'informatique théorique, nous supposons deux listes* $\mathscr{L}_1, \mathscr{L}_2 \subseteq \mathscr{E}$ *de taille* $O(L)$ *telles que* $\#\{(\mathbf{x}, \mathbf{y}) \in \mathscr{L}_1 \times \mathscr{L}_2 : \Delta(\mathbf{x}, \mathbf{y}) < r\}$ *est négligeable devant la dimension de* $\mathscr{E}$*.*

Définition 5.2.5 (modèle de proximité $\mathcal{M}_{\mathrm{Aléa}}(\mathscr{E}, L, \mathcal{D})$)**.** *Dans le modèle* $\mathcal{M}_{Aléa}(\mathscr{E}, L, \mathcal{D})$*, nous supposons deux listes* $\mathscr{L}_1, \mathscr{L}_2 \subseteq \mathscr{E}$ *de taille* $O(L)$ *dont chacun des éléments est produit en tirant aléatoirement un élément de* $\mathscr{E}$ *selon la distribution de probabilité* $\mathcal{D}$*.*

Lorsque $\mathscr{E}$ *est un espace probabilisé avec la probabilité uniforme, on notera* $\mathcal{M}_{Aléa}(\mathscr{E}, L)$*, le modèle* $\mathcal{M}_{Aléa}(\mathscr{E}, L, \mathcal{U})$ *où* $\mathcal{U}$ *est la distribution uniforme sur* $\mathscr{E}$*.*

Remarque 5.2.1. Résoudre les problèmes de recherche de proches voisins (resp. les problèmes 5.1.2 et 5.1.4) dans le modèle de proximité $\mathcal{M}_{\mathrm{IT}}(\mathscr{E}, L, (1+\varepsilon)w)$ revient à résoudre les problèmes approximatifs de recherche de voisins proches (resp. les problèmes 5.2.2 et 5.2.3).

Un exemple du modèle de proximité $\mathcal{M}_{\mathrm{Aléa}}(\mathscr{E}, L, \mathcal{D})$ est celui où $\mathscr{E}$ est $\mathbb{F}_q^n$ muni de la métrique de Hamming et où $\mathcal{D}$ est la distribution uniforme sur $\mathbb{F}_q^n$; on note alors ce modèle $\mathcal{M}_{\mathrm{Aléa}}\left(\mathbb{F}_q^n, L\right)$. Remarquons alors que pour toute distance $r > 0$ et pour tout L exponentiel en la dimension de $\mathscr{E}$, nous ne pouvons pas réduire le modèle $\mathcal{M}_{\mathrm{Aléa}}\left(\mathbb{F}_q^n, L\right)$ au modèle $\mathcal{M}_{\mathrm{IT}}\left(\mathbb{F}_q^n, L, r\right)$ puisque $\#\{(\mathbf{x}, \mathbf{y}) \in \mathscr{L}_1 \times \mathscr{L}_2 : \Delta(\mathbf{x}, \mathbf{y}) < r\}$ est exponentiel en n.

Une variante du modèle de proximité aléatoire. Une variante du modèle de proximité $\mathcal{M}_{\mathrm{Aléa}}(\mathscr{E}, L, \mathcal{D})$ est le modèle suivant :

Définition 5.2.6 (modèle de proximité $\mathcal{M}_{\mathrm{Abis}}(\mathscr{E}, L, \mathcal{D}_1, \mathcal{D}_2)$)**.** *Dans le modèle de proximité* $\mathcal{M}_{Abis}(\mathscr{E}, L, \mathcal{D}_1, \mathcal{D}_2)$*, nous supposons deux listes* $\mathscr{L}_1', \mathscr{L}_2' \subseteq \mathscr{E}$ *et un couple* $(\mathbf{x}^*, \mathbf{y}^*) \in \mathscr{E}^2$ *tels que :*

— $\mathscr{L}_1'$ *et* $\mathscr{L}_2'$ *sont produites selon le modèle* $\mathcal{M}_{Aléa}(\mathscr{E}, L, \mathcal{D}_1)$ *;*

— $(\mathbf{x}^*, \mathbf{y}^*)$ *est tiré aléatoirement sur* $\mathscr{E}^2$ *selon la distribution* $\mathcal{D}_2$*.*

Lorsque $\mathscr{E}$ est un espace probabilisé avec la probabilité uniforme, on notera $\mathcal{M}_{Abis}(\mathscr{E}, L, \mathcal{D})$, le modèle $\mathcal{M}_{Abis}(\mathscr{E}, L, \mathcal{U}, \mathcal{D})$ où $\mathcal{U}$ est la distribution uniforme sur $\mathscr{E}$.

Dans le modèle $\mathcal{M}_{\text{Abis}}(\mathscr{E}, L, \mathcal{D}_1, \mathcal{D}_2)$, un problème qui pourrait nous intéresser plutôt que le problème de recherche des presque-collisions est le suivant :

Problème 5.2.7 (recherche de presque-collisions dans le modèle aléatoire)**.** *Soit $(\mathscr{E}, d)$ un espace métrique et soient $\mathcal{D}_1$ une distribution de probabilités sur $\mathscr{E}$ et $\mathcal{D}_2$ une distribution de probabilités sur $\mathscr{E}^2$.*

Étant données deux listes $\mathscr{L}_1 := \mathscr{L}'_1 \cup \{\mathbf{x}^*\}$ et $\mathscr{L}_2 := \mathscr{L}'_2 \cup \{\mathbf{y}^*\}$ suivant le modèle de proximité $\mathcal{M}_{Abis}(\mathscr{E}, L, \mathcal{D}_1, \mathcal{D}_2)$, le but est de retrouver le couple $(\mathbf{x}^*, \mathbf{y}^*)$.

Un exemple de ce modèle de proximité que nous verrons par la suite est le modèle $\mathcal{M}_{\text{Abis}}(\mathbb{F}_2^n, L, \mathcal{B}(\tau))$ où un couple de vecteurs aléatoires $(\mathbf{x}, \mathbf{y}) := ((x_1, \cdots, x_n), (y_1, \cdots, y_n))$ défini sur $\mathbb{F}_2^n \times \mathbb{F}_2^n$ qui suit la distribution $\mathcal{B}(\tau)$ est tel que $\mathbf{x}$ suit la distribution uniforme sur $\mathbb{F}_2^n$ et pour tout $i \in [\![1, n]\!]$, $x_i + y_i$ suit une loi de Bernoulli de paramètre τ. Autrement dit, la distance $\Delta(\mathbf{x}, \mathbf{y})$ suit une loi binomiale de paramètre (n, τ).

Remarquons que dans ce modèle précis, nous pouvons majorer la distance $\Delta(\mathbf{x}^*, \mathbf{y}^*)$ des vecteurs proches recherchés avec une probabilité $(1 - o(1))$. Pour cela on utilise par exemple l'inégalité de Bienaymé-Tchebychev. Ainsi, nous pouvons résoudre le problème 5.2.7 en temps amorti en effectuant une recherche des presque-collisions ; c'est-à-dire en résolvant le problème 5.2.1. Cependant, dans la section 5.6, nous verrons un algorithme itératif probabiliste dont nous étudierons la complexité en moyenne en supposant $(\mathbf{x}^*, \mathbf{y}^*)$ comme source d'aléa.

5.3 L'approche des fonctions localement sensibles

5.3.1 Définition d'une famille de fonctions LSH

Une première idée pour résoudre le problème de recherche de presque-collisions est d'utiliser une approche similaire à la recherche de collisions exactes. Toutefois, s'il est certain que deux vecteurs égaux sont hachés sur une même valeur, il n'en est rien lorsque les vecteurs sont proches. Il nous faut donc choisir différemment notre fonction de hachage : cette fois-ci, elle doit hacher sur une même valeur, deux vecteurs proches. Notons que cette propriété est contraire aux propriétés d'une bonne fonction de hachage cryptographique.

Une manière naturelle de construire de telles fonctions est de partitionner l'espace ambiant et d'indexer chaque partie connexe avec un entier. La fonction de hachage est alors la fonction qui à chaque vecteur de l'espace ambiant, associe l'indice de la partie à laquelle il appartient. En procédant ainsi, nous espérons que deux vecteurs proches se trouvent dans une même partie et donc soient hachés sur une même adresse de la table de hachage. La figure 5.1 schématise la situation.

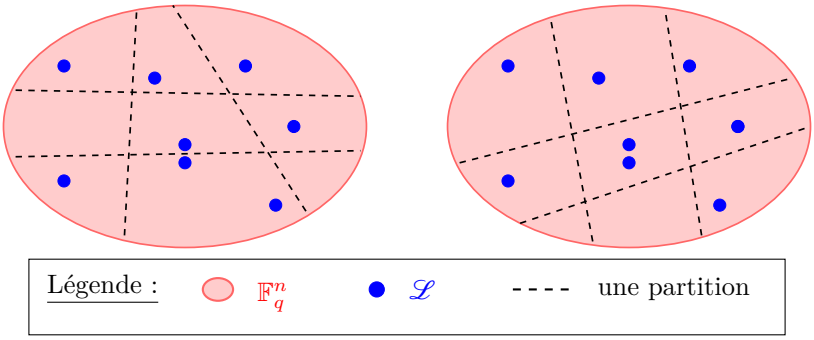


Figure 5.1 – Schématisation de deux partitionnements pour l'approche LSH. À gauche, un mauvais partitionnement et à droite, un bon partitionnement.

La seule solution pour que deux vecteurs proches se retrouvent systématiquement hachés sur une même valeur est de considérer le partitionnement trivial contenant une unique partie qui est l'espace tout entier. La fonction de hachage associée à ce partitionnement est alors une fonction constante. Le problème d'une telle fonction de hachage est qu'elle ne permet pas de distinguer des couples de vecteurs proches des autres couples. L'algorithme qui en découle est alors une simple recherche exhaustive des presque-collisions. Il faut donc partitionner l'espace de telle sorte que non seulement des vecteurs proches se retrouvent dans une même partie mais aussi que des vecteurs non proches se retrouvent dans des parties distinctes. Encore une fois, cette propriété est trop forte et ne peut pas être garantie. C'est pourquoi on se contentera de vouloir construire un partitionnement de l'espace qui unit des vecteurs proches avec une plus grande probabilité que les vecteurs éloignés. Les fonctions de hachage associées à de tels partitionnements sont dites localement sensibles (on parlera de fonctions LSH pour l'appellation anglaise *Locality Sensitive Hashing*).

Pour bien répartir les vecteurs des listes $\mathscr{L}_1$ et $\mathscr{L}_2$ dans notre table de hachage, l'espace est partitionné en un nombre de parties de l'ordre du nombre de vecteurs à hacher ; à savoir $O(L)$. Ce qui signifie que nos fonctions de hachage ont un domaine d'arrivée de taille $O(L)$. Nous verrons plus tard que ce choix n'est pas forcément optimal. Il est parfois plus judicieux de choisir un partitionnement qui contient moins de parties. Pour chaque adresse de la table de hachage, nous devrons alors tester exhaustivement plusieurs couples. Mais dans certains cas, ce surcoût est compensé par le nombre réduit d'adresses à explorer dans la table de hachage.

L'approche LSH fut introduite en 1998 par Indyk et Motwani dans [IM98]. Dans ce papier, les auteurs définissent formellement la propriété LSH d'une famille de fonctions de hachage. Leur définition est particulièrement adaptée à l'analyse de la recherche de voisins proches dans le modèle de proximité $\mathcal{M}_{\text{IT}}(\mathscr{E}, L, r)$. Nous allons généraliser l'approche LSH d'Indyk et Motwani pour que celle-ci s'applique également à l'analyse dans le modèle de proximité $\mathcal{M}_{\text{Aléa}}(\mathscr{E}, L, \mathcal{D})$.

Définition 5.3.1 (famille de fonctions localement sensibles)**.** *Soit $\mathscr{E}$ un espace métrique et soit $\mathscr{F}$ une famille de fonctions définies sur $\mathscr{E}$. On appelle* fonction de collision *de $\mathscr{F}$ la fonction qui, pour tout couple $(\mathbf{x}, \mathbf{y}) \in \mathscr{E}^2$, donne la probabilité de collision de $h(\mathbf{x})$ et $h(\mathbf{y})$ lorsque h est tirée uniformément dans $\mathscr{F}$:*

$$\forall (\mathbf{x}, \mathbf{y}) \in \mathscr{E}^2 \ , \quad p(\Delta(\mathbf{x}, \mathbf{y})) := \mathbb{P}(h(\mathbf{x}) = h(\mathbf{y})) \tag{5.3}$$

De plus, si p est décroissante, alors $\mathscr{F}$ est dite localement sensible *(ou LSH).*

Remarque 5.3.1. Dans la définition 5.3.1, la probabilité $\mathbb{P}(h(\mathbf{x}) = h(\mathbf{y}))$ est fonction de la distance entre $\mathbf{x}$ et $\mathbf{y}$. C'est pourquoi l'équation 5.3 a du sens.

Dasn [IM98], les auteurs montrent comment construire une famille de fonctions LSH en concaténant les fonctions d'une autre famille LSH. Nous rappelons leur construction dans le théorème 5.3.2. C'est cette nouvelle famille que nous allons utiliser pour la recherche de presque-collisions.

Théorème 5.3.2. *Soit $\mathscr{E}$ un espace métrique et soit $\mathscr{F}$ une famille de fonctions définies sur $\mathscr{E}$. Soit t un entier strictement positif. On pose alors $\mathscr{F}_t$ la famille de fonctions suivante :*

$$\mathscr{F}_t := \left\{ g := h_1 | \cdots | h_t \ : \ \forall i \in [\![1, t]\!], h_i \in \mathscr{F} \right\} \tag{5.4}$$

Si p est la fonction de collision de $\mathscr{F}$, alors p^t est celle de $\mathscr{F}_t$. De plus, si $\mathscr{F}$ est LSH, alors $\mathscr{F}_t$ l'est aussi.

Démonstration du théorème 5.3.2.

Soit $g := h_1 | \cdots | h_t$ tirée uniformément dans $\mathscr{F}_t$ et soit $(\mathbf{x}, \mathbf{y}) \in \mathscr{E}^2$.

$$\begin{aligned} \mathbb{P}\left(g(\mathbf{x}) = g(\mathbf{y})\right) &= \mathbb{P}\left(\textstyle\bigwedge_{i=1}^{t} h_i(\mathbf{x}) = h_i(\mathbf{y})\right) \\ &= \textstyle\prod_{i=1}^{t} \mathbb{P}\left(h_i(\mathbf{x}) = h_i(\mathbf{y})\right) \end{aligned}$$

car les h_i sont indépendants. D'où :

$$\mathbb{P}\left(g(\mathbf{x}) = g(\mathbf{y})\right) \leqslant p(\Delta(\mathbf{x}, \mathbf{y}))^t$$

En outre, la propriété LSH est conservée car les variations de p et p^t sont les mêmes. □

5.3.2 L'algorithme LSH générique

Finalement, l'algorithme 5.3.1 décrit la procédure pour trouver les presque-collisions en utilisant des fonctions de hachage localement sensibles.

Algorithme 5.3.1 : La méthode LSH

Entrées : deux listes $\mathscr{L}_1$ et $\mathscr{L}_2 \subseteq \mathscr{E}$;
une distance w.
Paramètres : un nombre d'itérations s ;
un entier t ;
un entier T ;
une famille de fonctions LSH $\mathscr{F} \subseteq \{\mathscr{E} \to [\![1, T]\!]\}$.
Sortie : $\mathscr{L}^* = \{(\mathbf{x}, \mathbf{y}) \in \mathscr{L}_1 \times \mathscr{L}_2 \text{ tel que } \Delta(\mathbf{x}, \mathbf{y}) \leqslant w\}$.

```
1  L* ← ∅ ;
2  répéter s fois
3  |  choisir h_1, ··· , h_t uniformément et indépendamment dans F ;
4  |  initialiser une table de hachage T de taille T^t ;
5  |  pour tout x ∈ L_1 faire
6  |  |  ajouter x dans T[g(x)] ;        /* où g(x) := 1 + Σ_{i=1}^{t} (h_i(x) − 1)T^{i−1} */
7  |  finPour
8  |  pour tout y ∈ L_2 faire
9  |  |  pour tout x ∈ T[g(y)] faire
10 |  |  |  si Δ(x, y) ⩽ w alors
11 |  |  |  |  ajouter (x, y) dans L* ;
12 |  |  |  finSi
13 |  |  finPour
14 |  finPour
15 finRépéter
16 retourner L* ;
```

Remarque 5.3.2. Dans l'algorithme 5.3.1, la fonction g est construite en composant la fonction $\mathbf{x} \mapsto (h_1(\mathbf{x}), \cdots, h_t(\mathbf{x}))$ et une bijection de $[\![1, T]\!]^t \to [\![1, T^t]\!]$.

5.3.3 L'algorithme LSH dans le modèle de proximité $\mathcal{M}_{\mathsf{IT}}(\mathscr{E}, L, (1+\varepsilon)w)$

Soit $\varepsilon > 0$. Dans le modèle de proximité $\mathcal{M}_{\mathrm{IT}}(\mathscr{E}, L, (1+\varepsilon)w)$, les couples de $\mathscr{L}_1 \times \mathscr{L}_2$ sont tous à une distance au moins $(1+\varepsilon)w$ sauf un nombre négligeable de couples ; nous cherchons alors les couples à distance au plus w. Nous considérons le pire cas ; à savoir, celui où les couples proches sont à distance exactement w et les couples éloignés sont à distance exactement $(1+\varepsilon)w$. Indyk et Motwani montrent dans [IM98] que dans ce modèle particulier, la complexité de l'algorithme 5.3.1 peut s'exprimer à l'aide d'une formule très simple que nous rappelons dans le théorème 5.3.3.

Théorème 5.3.3. *Dans le modèle de proximité $\mathcal{M}_{IT}(\mathscr{E}, L, (1+\varepsilon)w)$, l'algorithme 5.3.1 peut résoudre le problème des presque-collisions en utilisant un espace mémoire de l'ordre de $O(L)$ et en évaluant $\tilde{O}(L^\alpha)$ fonctions LSH où :*

$$\alpha := 1 + \frac{\log\left(\frac{1}{P^*}\right)}{\log\left(\frac{1}{P}\right)} \tag{5.5}$$

avec $P^* \leqslant p(w)$, $P \geqslant p\left((1+\varepsilon)w\right)$ et p la fonction de collision de $\mathscr{F}$.

Nous trouvons une démonstration du théorème 5.3.3 dans [IM98]. Il est notamment montré que la complexité en temps énoncée dans ce théorème est atteinte pour $t = \left\lceil \frac{\log(L)}{\log\left(\frac{1}{P}\right)} \right\rceil$ et $s = \lceil (P^*)^{-t} \rceil$.

En outre, puisqu'à la fin de chaque itération la table de hachage contient exactement $\#\mathscr{L}_1$ éléments et que cette table est oubliée dès l'itération suivante, l'algorithme 5.3.1 a une complexité mémoire de l'ordre de $O(L)$. Il faut noter que dans [IM98], le problème considéré n'est pas celui des presque-collisions mais le problème des proches voisins. De ce fait, l'impact de leur version de l'algorithme LSH sur la mémoire est différent : ils ont besoin de stocker toutes les tables de hachage de chaque itération en même temps et donc ils utilisent $O(L^\alpha)$ espace mémoire.

5.3.4 L'algorithme LSH dans le modèle de proximité $\mathcal{M}_{\mathsf{Aléa}}(\mathscr{E}, L, \mathcal{D})$

Dans le modèle de proximité $\mathcal{M}_{\mathrm{Aléa}}(\mathscr{E}, L, \mathcal{D})$, $\mathscr{L}_1$ et $\mathscr{L}_2$ sont produits en tirant des éléments de $\mathscr{E}$ selon une loi de probabilité $\mathcal{D}$. La distribution des distances de $\mathscr{L}_1 \times \mathscr{L}_2$ est alors très différente de celle du modèle de proximité $\mathcal{M}_{\mathrm{IT}}(\mathscr{E}, L, (1+\varepsilon)w)$ et donc l'analyse d'Indyk et Motwani dans [IM98] n'est plus valable ici. De plus, le nombre de couples proches recherchés peut être exponentiel en la dimension de $\mathscr{E}$ et n'est donc pas nécessairement négligeable contrairement au modèle précédent.

Théorème 5.3.4. *Dans le modèle de proximité $\mathcal{M}_{Aléa}(\mathscr{E}, L, \mathcal{D})$, l'algorithme 5.3.1 peut résoudre le problème des presque-collisions en utilisant un espace mémoire de l'ordre de $S_{\mathrm{LSH}}^{\mathrm{Aléa}} = O(L)$ et en évaluant $T_{\mathrm{LSH}}^{\mathrm{Aléa}}(t)$ fonctions LSH où :*

$$T_{\mathrm{LSH}}^{\mathrm{Aléa}}(t) = \tilde{O}\left(L \cdot \frac{1 + L \cdot \int_{\mathbb{R}} f(u) p(u)^t du}{p(w)^t} \right) \tag{5.6}$$

avec p la fonction de collision de $\mathscr{F}$ et f la fonction réelle telle que pour tous $a \leqslant b \in \mathbb{R}$, $\int_a^b f(u)du = \mathbb{P}\left(\Delta(\mathbf{x}, \mathbf{y}) \in [a, b]\right)$ lorsque $\mathbf{x}$ et $\mathbf{y}$ sont tirés indépendamment selon la loi $\mathcal{D}$.

Démonstration du théorème 5.3.4.

La probabilité de trouver un couple proche particulier lors d'une itération est au moins :

$$\inf_{u \leqslant w} (p(u))^t = p(w)^t$$

Nous sommes face à un problème de type "collectionneur de coupons" [FS14] donc l'ensemble des couples proches est trouvé en $s := \tilde{O}\left(\frac{1}{p(w)^t}\right)$ itérations de l'algorithme 5.3.1.

D'autre part, pour chaque itération, l'algorithme hache une seule fois chaque vecteur des listes $\mathscr{L}_1$ et $\mathscr{L}_2$. Le coût pour hacher les listes est donc de l'ordre de $O(L)$. D'autre part, le nombre de couples dont nous devons tester la proximité à chaque itération est de l'ordre de :

$$O\left(L^2 \cdot \int_{\mathbb{R}} f(u)p(u)^t du\right)$$

Nous obtenons alors bien la complexité en temps prétendue dans le théorème.

En outre, la démonstration pour la complexité mémoire est similaire à celle du théorème 5.3.4. □

Remarque 5.3.3. Une alternative à l'approche LSH a été proposée par Andoni, Indyk, Nguyen et Razenshteyn dans [AINR14, AR15]. Dans ces papiers, il est proposé d'utiliser la structure de la donnée pour résoudre le problème des voisins proches dans le modèle de proximité $\mathcal{M}_{\mathrm{IT}}(\mathscr{E}, L, (1+\varepsilon)w)$ (cf. problème 5.2.3). Cette approche peut s'étendre à la résolution du problème de recherche de presque-collisions dans le modèle de proximité $\mathcal{M}_{\mathrm{IT}}(\mathscr{E}, L, (1+\varepsilon)w)$. Les méthodes dites *data-dependent* sont les premières améliorations significatives pour résoudre des problèmes de proximité depuis [IM98]. Cependant, il n'est pas clair qu'elles s'adaptent au modèle de proximité $\mathcal{M}_{\text{Aléa}}(\mathscr{E}, L, \mathcal{D})$.

5.4 Des projections comme fonctions LSH

Intéressons-nous au cas où $\mathscr{E}$ est l'espace $\mathbb{F}_q^n$ muni de la métrique de Hamming. Dans cette section, nous généralisons une méthode bien connue dans le cas binaire au cas non-binaire et nous analysons cette méthode dans différents modèles de proximité.

Pour tout entier positif $k \leqslant n - w$, nous posons la famille de fonctions $\mathscr{F}_{\mathrm{Proj}}(k)$ constituée des projections sur exactement k positions :

$$\mathscr{F}_{\mathrm{Proj}}(k) := \left\{ f_I : \begin{array}{ccc} \mathbb{F}_q^n & \to & [\![1, q^k]\!] \\ (x_1, \cdots, x_n) & \mapsto & 1 + \sum_{s=1}^{k} x_{i_s} q^{s-1} \end{array} : I := \{i_1, \cdots, i_k\} \subseteq [\![1, n]\!] \text{ et } \#I = k \right\} \tag{5.7}$$

La fonction de collision associée à $\mathscr{F}_{\mathrm{Proj}}(k)$ est :

$$\begin{array}{rccc} p_k : & [\![0, n]\!] & \longrightarrow & [0, 1] \\ & u & \longmapsto & \frac{\binom{n-k}{u}}{\binom{n}{u}} \end{array} \tag{5.8}$$

La suite finie $(p_k(u))_{u \in [\![0,n]\!]}$ peut être définie récursivement :

$$p_k(u) := \begin{cases} 1 & \text{si } u = 0 \\ \max\left(0, \frac{n-k-u+1}{n-u+1}\right) \cdot p_k(u-1) & \text{si } u \in [\![1, n]\!] \end{cases} \tag{5.9}$$

Or pour tout $u \in [\![1, n]\!]$, $\max\left(0, \frac{n-k-u+1}{n-u+1}\right) \in [0, 1]$ donc p_k est décroissante et donc la famille $\mathscr{F}_{\mathrm{Proj}}(k)$ est bien LSH.

En outre, dans la version originale du papier [IM98], les auteurs proposent comme fonctions LSH des projections sur une seule position. Cette famille de fonctions correspond dans notre cas au choix $k = 1$. La fonction de collision associée à la famille $\mathscr{F}_{\text{Proj}}(1)$ est :

$$\begin{array}{rccc} p_1 : & [\![0, n]\!] & \longrightarrow & [0, 1] \\ & u & \longmapsto & \dfrac{\binom{n-1}{u}}{\binom{n}{u}} = 1 - \dfrac{u}{n} \end{array} \tag{5.10}$$

5.4.1 Les projections dans le modèle de proximité $\mathcal{M}_{\text{IT}}\left(\mathbb{F}_q^n, L, (1+\varepsilon)w\right)$

Regardons dans un premier temps ce qu'il se passe lorsque les listes $\mathscr{L}_1$ et $\mathscr{L}_2$ suivent le modèle de proximité $\mathcal{M}_{\text{IT}}\left(\mathbb{F}_q^n, L, (1+\varepsilon)w\right)$. On suppose donc que tous les couples de $\mathscr{L}_1 \times \mathscr{L}_2$ ont une distance de Hamming d'au moins $w(1+\varepsilon)$ sauf un nombre négligeable de couples. Par la suite, nous étudions le cas asymptotique où n tend vers l'infini et où $w := \lfloor \omega n \rfloor$ avec $\omega \in [0, 1]$ une constante.

En appliquant le théorème 5.3.3, nous montrons que l'utilisation de la famille de fonctions $\mathscr{F}_{\text{Proj}}(1)$ dans l'algorithme 5.3.1 permet de résoudre le problème des presque-collisions en un temps de l'ordre de :

$$T_{\text{Proj}}^{\text{Aléa}}(1) = O\left(L^{1+\frac{\log(1-w)}{\log(1-w(1+\varepsilon))}}\right) \tag{5.11}$$

Posons une constante $R \in [0, 1]$. Nous pouvons généraliser la méthode des projections en instanciant l'algorithme 5.3.1 non plus avec la famille $\mathscr{F}_{\text{Proj}}(1)$ mais avec la famille $\mathscr{F}_{\text{Proj}}(k)$ où $k := \lfloor Rn \rfloor$. Ainsi, d'après le théorème 5.3.3, la complexité de la méthode obtenue est :

$$T_{\text{Proj}}^{\text{Aléa}}(k) = O\left(L^{1+\frac{(1-R)h_2\left(\frac{\omega}{1-R}\right)-h_2(\omega)}{(1-R)h_2\left(\frac{(1+\varepsilon)\omega}{1-R}\right)-h_2((1+\varepsilon)\omega)}}\right) \tag{5.12}$$

La valeur de k qui minimise la complexité $T_{\text{Proj}}^{\text{Aléa}}(k)$ est $k_{\text{opt}} := n - (1+\varepsilon)w$. Ce choix donne :

$$T_{\text{Proj}}^{\text{Aléa}}(k_{\text{opt}}) = O\left(L^{1-\frac{(1+\varepsilon)\omega h_2\left(\frac{1}{1+\varepsilon}\right)-h_2(\omega)}{h_2((1+\varepsilon)\omega)}}\right) \tag{5.13}$$

La figure 5.2 compare les complexités des méthodes obtenues avec $k = 1$ et k optimal en fonction de ε.

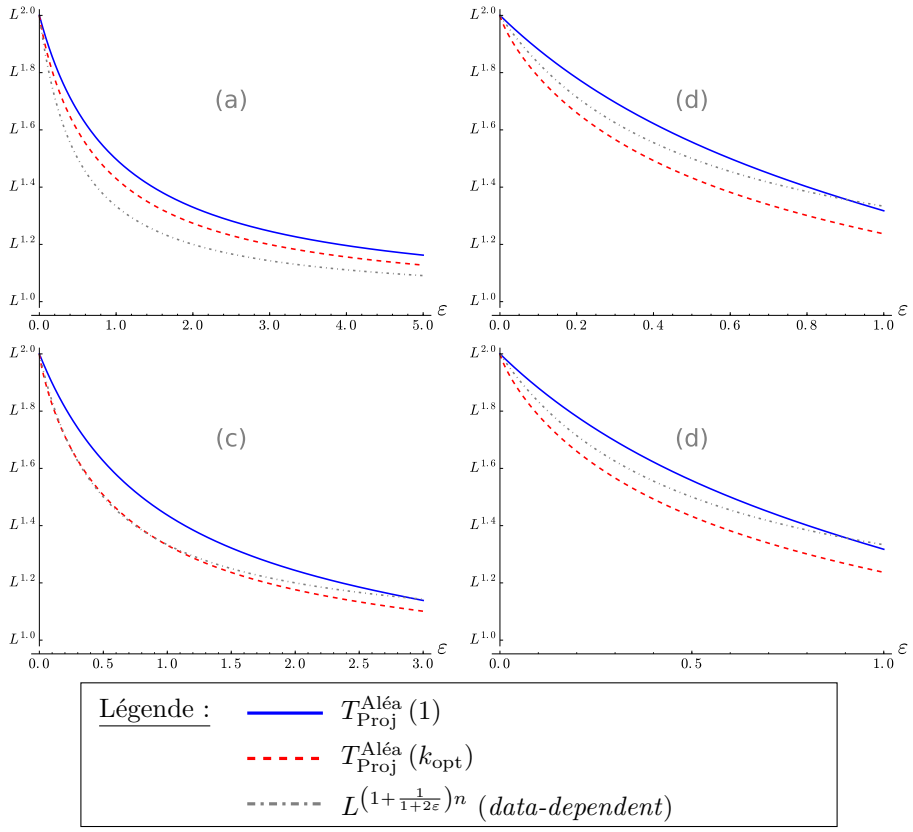


Figure 5.2 – Complexité de la méthode des projections pour la recherche de presque-collisions sur $\mathbb{F}_q^n$ dans le modèle de proximité $\mathcal{M}_{\text{IT}}\left(\mathbb{F}_q^n, L, (1+\varepsilon)w\right)$. (a) $\omega = 0.01$, (b) $\omega = 0.1$, (c) $\omega = 0.2$, (d) $\omega = 0.4$.

Remarquons que lorsque ε tend vers 0, la complexité de la méthode tend vers $O\left(L^2\right)$ qui est la complexité de la recherche exhaustive. En revanche, lorsque ε grandit, la complexité de la méthode décroit. Cela montre bien que plus ε est proche de 0, plus il est difficile de distinguer les presque-collisions tandis qu'au contraire, lorsque ε est grand, les couples d'éléments proches se distinguent mieux des couples d'éléments éloignés.

Remarque 5.4.1. Notons que la combinaison de t fonctions de hachage choisies indépendamment et uniformément dans $\mathscr{F}_{\text{Proj}}(1)$ ne revient pas à choisir une projection dans $\mathscr{F}_{\text{Proj}}(t)$. En effet, lorsqu'une projection de dimension t est choisie, les t positions de la projection ne sont pas choisies indépendamment les unes des autres : notamment, on ne peut pas choisir plusieurs fois la même position.

5.4.2 Les projections dans le modèle de proximité $\mathcal{M}_{\text{Aléa}}\left(\mathbb{F}_q, L\right)$

Considérons à présent le modèle de proximité $\mathcal{M}_{\text{Aléa}}\left(\mathbb{F}_q, L\right)$ où les listes $\mathscr{L}_1$ et $\mathscr{L}_2$ sont produites en tirant des vecteurs uniformément dans $\mathbb{F}_q^n$. La probabilité qu'un mot $\mathbf{x}$ tiré uniformément dans $\mathbb{F}_q^n$ soit de poids de Hamming u est :

$$f(u) := \mathbb{P}_{\mathbf{x}\in\mathbb{F}_q^n}\left(|\mathbf{x}| = u\right) = \frac{\binom{n}{u}(q-1)^u}{q^n} \tag{5.14}$$

Nous étudions le cas asymptotique où n tend vers l'infini et où $w := \lfloor \omega n \rceil$ avec $\omega \in [0,1]$ une constante.

Dans un premier temps, nous instancions l'algorithme 5.3.1 avec la famille LSH $\mathscr{F}_{\mathrm{Proj}}(1)$ des projections sur une position. La fonction de collision de cette famille est donnée par l'équation (5.10). Ainsi, d'après le théorème 5.3.4, les projections $\mathscr{F}_{\mathrm{Proj}}(1)$ permettent de résoudre le problème des presque-collisions sur $\mathbb{F}_q^n$ dans le modèle de proximité $\mathcal{M}_{\text{Aléa}}(\mathbb{F}_q, L)$ en un temps de l'ordre de :

$$T_{\mathrm{Proj}}^{\text{Aléa}}(1) = O\left(L \cdot \inf_{t \in \mathbb{N}^*} \left(\frac{1 + L \sum_{u=0}^{n} f(u) p_1(u)^t}{p_1(w)^t} \right) \right) \tag{5.15}$$

$$= O\left(L \cdot \inf_{t \in \mathbb{N}^*} \left(\frac{1 + L \sup_{\mu \in [0,1]} \left(q^{t \log_q(1-\mu) + n h_q(\mu) - n} \right)}{q^{t \log_q(1-\omega)}} \right) \right) \tag{5.16}$$

Supposons $L = q^{\lambda n}$ pour une constante $\lambda \in [0,1]$. Une autre famille de projections que l'on retrouve souvent dans la littérature est alors $\mathscr{F}_{\mathrm{Proj}}(\ell)$ où $\ell := \lceil \lambda n \rceil$. Pour que la taille des tables de hachage soit du même ordre que celle des listes, nous devons donc choisir $t = 1$. La recherche des presque-collisions coûte ainsi :

$$T_{\mathrm{Proj}}^{\text{Aléa}}(\ell) = O\left(L \cdot \frac{1 + L \sum_{u=0}^{n} f(u) p_\ell(u)}{p_\ell(w)} \right) \tag{5.17}$$

$$= O\left(L \cdot \frac{1 + L q^{-\ell}}{p_\ell(w)} \right) \tag{5.18}$$

$$= O\left(L \cdot q^{n\left(h_q(\omega) - (1-\lambda) h_q\left(\frac{\omega}{1-\lambda} \right) \right)} \right) \tag{5.19}$$

Les deux familles LSH que nous venons d'énoncer produisent des instances particulières de la méthode LSH avec projections. La famille de projections optimale est la famille $\mathscr{F}_{\mathrm{Proj}}(k_{\mathrm{opt}})$ où k_{opt} minimise :

$$T_{\mathrm{Proj}}^{\text{Aléa}}(k) = O\left(L \cdot \inf_{t \in \mathbb{N}^*} \left(\frac{1 + L \sum_{u=0}^{n} f(u) p_k(u)^t}{p_k(w)^t} \right) \right) \tag{5.20}$$

$$= O\left(L \cdot \inf_{t \in \mathbb{N}^*} \left(\frac{1 + L \sup_{\mu \in [0,1]} \left(q^{n\left(t(1-R) h_q\left(\frac{\mu}{1-R} \right) + (1-t) h_q(\mu) - 1 \right)} \right)}{q^{n\left(t(1-R) h_q\left(\frac{\omega}{1-R} \right) - t h_q(\omega) \right)}} \right) \right) \tag{5.21}$$

où $R := \frac{k}{n}$.

La figure 5.3 représente la complexité en fonction de ω de la méthode LSH des projections pour résoudre le problème des presque-collisions dans le modèle de proximité

$\mathcal{M}_{\text{Aléa}}\left(\mathbb{F}_q, q^{\lambda n}\right)$. Nous avons comparé les différentes familles de projections que nous avons vu juste avant ($\mathscr{F}_{\text{Proj}}(1)$, $\mathscr{F}_{\text{Proj}}(\ell)$ et $\mathscr{F}_{\text{Proj}}(k_{\text{opt}})$). Il semblerait que la famille $\mathscr{F}_{\text{Proj}}(k_{\text{opt}})$ optimale soit essentiellement $\mathscr{F}_{\text{Proj}}(1)$ ou $\mathscr{F}_{\text{Proj}}(\ell)$ (où, rappelons-le, $\ell := \lceil \lambda n \rceil$) :

Conjecture 5.4.1.

$$T^{\text{Aléa}}_{\text{Proj}}\left(k_{opt}\right) = \min\left(T^{\text{Aléa}}_{\text{Proj}}\left(1\right)\ ,\ T^{\text{Aléa}}_{\text{Proj}}\left(\ell\right)\right) \tag{5.22}$$

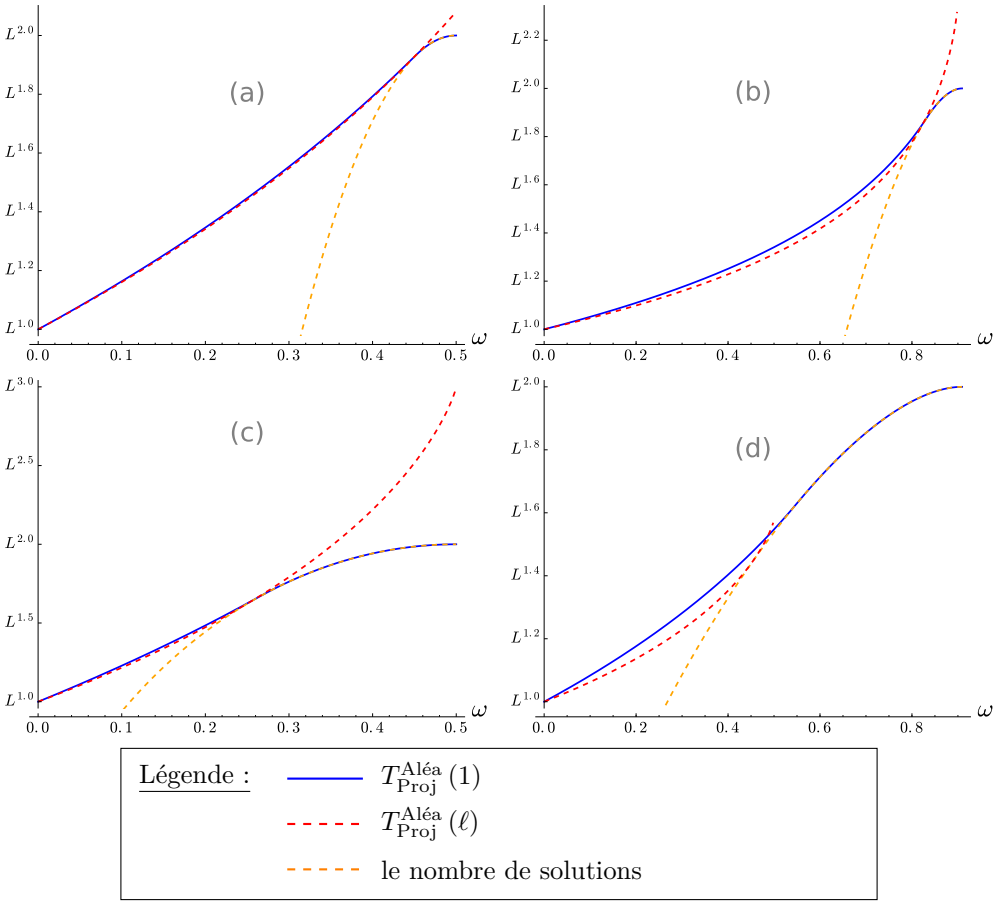


Figure 5.3 – Complexité de la méthode des projections pour la recherche de presque-collisions sur $\mathbb{F}_q^n$ dans le modèle de proximité $\mathcal{M}_{\text{Aléa}}\left(\mathbb{F}_q, q^{\lambda n}\right)$. (a) $q = 2$ et $\lambda = 0.1$, (b) $q = 11$ et $\lambda = 0.1$, (c) $q = 2$ et $\lambda = 0.5$, (d) $q = 11$ et $\lambda = 0.5$.

Remarquons que dans le modèle de proximité $\mathcal{M}_{\text{Aléa}}\left(\mathbb{F}_q, L\right)$, la méthode des projections interpole bien entre la recherche de collisions exactes et la recherche exhaustive. En effet, lorsque ω est proche de 0, la complexité de notre méthode se rapproche de la complexité $O(L)$ de la recherche de collisions exactes et lorsque ω tend vers $1 - \frac{1}{q}$, la complexité de la méthode tend vers la complexité $O\left(L^2\right)$ de la recherche exhaustive ; ce qui est moral car les couples quelconques sont alors typiquement proches et il est donc difficile de faire mieux qu'une recherche exhaustive. En outre, nous remarquons qu'au voisinage de $1 - \frac{1}{q}$,

la méthode des projections a essentiellement une complexité de l'ordre du nombre de couples proches recherchés ; la méthode des projections est donc optimale dans cette zone de paramètres.

5.5 L'approche de May et Ozerov

Dans le cadre de la cryptographie basée sur les codes correcteurs d'erreurs, May et Ozerov se sont penchés sur le problème des presque-collisions dans la métrique binaire de Hamming. Dans leur article [MO15], ils proposent une méthode de résolution du problème de recherche des presque-collisions différente de la méthode des projections. Nous présentons ici succinctement leur algorithme.

Une version non-binaire de la méthode May-Ozerov existe. Elle a été proposée par Hirose dans [Hir16]. Nous ne détaillerons pas son algorithme et nous contenterons essentiellement du cas binaire.

Reposons, dans un premier temps, le problème que nous cherchons à résoudre : soient deux listes $\mathscr{L}_1$ et $\mathscr{L}_2$ incluses dans $\mathbb{F}_2^n$ et une distance $w := \lfloor \omega n \rfloor \in [\![0, n]\!]$ pour une constante $\omega \in [0, 1]$. Le problème des presque-collisions auquel nous nous intéressons consiste alors à rechercher l'ensemble des couples $(\mathbf{x}, \mathbf{y}) \in \mathscr{L}_1 \times \mathscr{L}_2$ tels que $\Delta(\mathbf{x}, \mathbf{y}) \leqslant w$. Nous supposons que les tailles de $\mathscr{L}_1$ et $\mathscr{L}_2$ sont du même ordre $O\left(2^{\lambda n}\right)$. Le modèle de proximité que nous considérons est le modèle de proximité $\mathcal{M}_{\text{Aléa}}\left(\mathbb{F}_2^n, 2^{\lambda n}\right)$; c'est-à-dire que les listes $\mathscr{L}_1$ et $\mathscr{L}_2$ sont produites en tirant uniformément des vecteurs dans $\mathbb{F}_2^n$. Soit $(\mathbf{x}^*, \mathbf{y}^*) \in \mathscr{L}_1 \times \mathscr{L}_2$ une solution particulière de notre problème. Nous considérons le pire cas possible ; à savoir celui où $\Delta(\mathbf{x}^*, \mathbf{y}^*) = w$.

Nous rappelons une notation que nous utiliserons beaucoup dans cette section :

Notation 5.5.1. *Soit* $\mathbf{x} := (x_1, \cdots, x_n) \in \mathbb{F}_2^n$ *et soit* $I := (i_j)_{1 \leqslant j \leqslant t} \subseteq [\![1, n]\!]$. *On note* $\mathbf{x}_I$ *le vecteur suivant :*

$$\mathbf{x}_I := (x_{i_1}, \cdots, x_{i_t}) \in \mathbb{F}_2^t \tag{5.23}$$

5.5.1 L'algorithme de May et Ozerov

Première intuition de l'algorithme May-Ozerov. L'idée de May et Ozerov dans [MO15] consiste à construire un nombre exponentiel N de paires de listes $\left\{\left(\mathscr{L}_1^{(i)}, \mathscr{L}_2^{(i)}\right)\right\}_{i \in [\![1, N]\!]}$ telles que pour tout $j \in \{1, 2\}$ et tout $i \in [\![1, N]\!]$, $\mathscr{L}_j^{(i)} \subseteq \mathscr{L}_j$ et telles qu'au moins une paire contienne la solution $(\mathbf{x}^*, \mathbf{y}^*)$. Cette dernière est alors recherchée de façon exhaustive dans chacun des ensembles $\mathscr{L}_1^{(i)} \times \mathscr{L}_2^{(i)}$. Chacune de ces recherche exhaustive s'effectue en un temps négligeable si les tailles des listes $\mathscr{L}_1^{(i)}$ et $\mathscr{L}_2^{(i)}$ sont suffisamment faibles. May et Ozerov proposent alors de construire les paires de sous-listes de la façon suivante :

$$\mathscr{L}_1^{(i)} := \left\{\mathbf{x} \in \mathscr{L}_1 \ : \ |\mathbf{x}_{A_i}| = \frac{n h_2^{-1}(1-\lambda)}{2}\right\} \tag{5.24}$$

$$\mathscr{L}_2^{(i)} := \left\{\mathbf{y} \in \mathscr{L}_2 \ : \ |\mathbf{y}_{A_i}| = \frac{n h_2^{-1}(1-\lambda)}{2}\right\} \tag{5.25}$$

avec A_i un ensemble de $\left\{A \subseteq [\![1, n]\!] : \#A = \frac{n}{2}\right\}$. On remarque l'intervention ici de la distance de Gilbert-Varshamov $d_{\text{GV}}(n, \lambda n) = n h_2^{-1}(1-\lambda)$ (cf. chapitre 1 section 1.1). Ainsi construites, les sous-listes sont typiquement de tailles polynomiales en n (cf. [MO15, lemme 3]) et le nombre de paires de sous-listes nécessaires pour que le nombre d'entre

elles qui contiennent $(\mathbf{x}^*, \mathbf{y}^*)$ soit typiquement supérieur à 1 est $N = \tilde{O}(2^{yn})$ (cf. [MO15, lemme 2]) avec :

$$y := (1-\omega)\left(1 - h_2\left(\frac{h_2^{-1}(1-\lambda) - \frac{\omega}{2}}{1-\omega}\right)\right) \tag{5.26}$$

Notons que les paramètres λ et ω du problème de recherche de presque-collisions doivent vérifier la contrainte :

$$\lambda < 1 - h_2\left(\frac{\omega}{2}\right) \tag{5.27}$$

pour que la méthode May-Ozerov puisse être appliquée. De plus, chaque paire $\left(\mathscr{L}_1^{(i)}, \mathscr{L}_2^{(i)}\right)$ est construite en parcourant intégralement les listes d'origine $\mathscr{L}_1$ et $\mathscr{L}_2$. Le coût moyen de cette méthode est donc de l'ordre de $\tilde{O}\left(2^{(\lambda+y)n}\right)$. May et Ozerov proposent une solution pour diminuer ce coût.

Un traitement bloc par bloc. Pour produire leurs sous-listes, May et Ozerov procèdent étape par étape en ne traitant pas l'ensemble des n coordonnées d'un seul coup mais plutôt en les traitant bloc par bloc. Plus formellement, soit $t \in [\![1, n]\!]$ et soient t entiers $n_1, \cdots, n_t$ tels que $\sum_{s=1}^{t} n_s = n$. Nous posons également t entiers $N_1, \cdots, N_t$ et nous construisons récursivement les listes suivantes :

$$\mathscr{L}_1^{(\varepsilon)} := \mathscr{L}_1 \tag{5.28}$$

$$\mathscr{L}_2^{(\varepsilon)} := \mathscr{L}_2 \tag{5.29}$$

$$\mathscr{L}_1^{(\mathbf{u},i)} := \left\{\mathbf{x} \in \mathscr{L}_1^{(\mathbf{u})} \ : \ \left|\mathbf{x}_{A_{(\mathbf{u},i)}}\right| = \frac{n_s h_2^{-1}(1-\lambda)}{2}\right\} \tag{5.30}$$

$$\mathscr{L}_2^{(\mathbf{u},i)} := \left\{\mathbf{y} \in \mathscr{L}_2^{(\mathbf{u})} \ : \ \left|\mathbf{y}_{A_{(\mathbf{u},i)}}\right| = \frac{n_s h_2^{-1}(1-\lambda)}{2}\right\} \tag{5.31}$$

où $s \in [\![1, t]\!]$, $\mathbf{u} \in [\![1, N_1]\!] \times \cdots \times [\![1, N_{s-1}]\!]$, $i \in [\![1, N_s]\!]$ et $A_{(\mathbf{u},i)}$ est tiré uniformément dans $\left\{A \subseteq \left[\!\left[1 + \sum_{j=1}^{s-1} n_j, \sum_{j=1}^{s} n_j\right]\!\right] : \#A = \frac{n_s}{2}\right\}$. Notons que le symbole ε représente la chaine vide.

Notation 5.5.2. *Rappelons que pour tout $\mathbf{u} \in \mathbb{N}^{s-1}$ et tout $i \in \mathbb{N}$, nous notons $(\mathbf{u}, i)$ la concaténation du mot $\mathbf{u}$ et de l'entier i. Le mot ainsi produit est une chaine d'entiers de longueur s. Par exemple, si $\mathbf{u} := (1,2,3) \in \mathbb{N}^3$ et $i = 4$, alors $(\mathbf{u}, i) = (1,2,3,4) \in \mathbb{N}^4$.*

Notons $\mathscr{T}$ l'arbre suivant : la chaine vide ε est la racine et pour tout $s \in [\![1, t]\!]$, chaque nœud $\mathbf{u} \in [\![1, N_1]\!] \times \cdots \times [\![1, N_{s-1}]\!]$ à distance $s-1$ de la racine possède les N_s fils $(\mathbf{u}, 1), \cdots, (\mathbf{u}, N_s)$. Les feuilles de $\mathscr{T}$ sont alors tous les éléments de $[\![1, N_1]\!] \times \cdots \times [\![1, N_t]\!]$. Pour tout nœud $\mathbf{u} \in \mathscr{T}$, nous lui associons les listes $\mathscr{L}_1^{(\mathbf{u})}$ et $\mathscr{L}_2^{(\mathbf{u})}$ construites selon les équations (5.28) à (5.31). La figure 5.4 représente l'arbre $\mathscr{T}$ avec les listes associées à chaque nœud. Sur cette figure, l'indice j vaut 1 ou 2 et $\cdot^{\cdot^{\cdot}}$ indique un ensemble de chemins implicites.

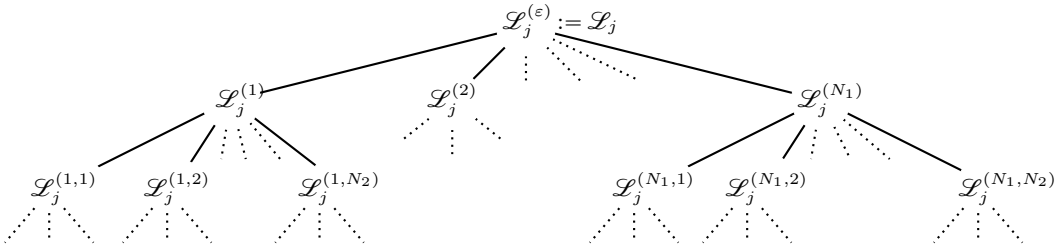


Figure 5.4 – Représentation sous forme d'arbre de la construction des sous-listes de la méthode May-Ozerov.

Notons que pour tout chemin qui va de la racine à une feuille de l'arbre, chaque bifurcation correspond au choix d'un ensemble de positions $A_{\mathbf{u}}$ où $\mathbf{u}$ est le nœud de la bifurcation. Si $\mathbf{u}$ est à l'étage s (la racine étant à l'étage 0 et les feuilles à l'étage t), alors $A_{\mathbf{u}}$ est un ensemble de $\frac{n_s}{2}$ indices choisis aléatoirement dans $\left[\!\left[1+\sum_{j=1}^{s-1} n_j, \sum_{j=1}^{s} n_j\right]\!\right]$. Pour tout nœud $\mathbf{u}$ de $\mathscr{T}$, on note $B_{\mathbf{u}}$ l'ensemble des positions mises en jeu dans les listes $\mathscr{L}_1^{(\mathbf{u})}$ et $\mathscr{L}_2^{(\mathbf{u})}$. En d'autres termes :

$$B_{\mathbf{u}} := \bigcup_{\mathbf{r}\in C(\mathbf{u})} A_{\mathbf{r}} \tag{5.32}$$

où $C(\mathbf{u})$ est l'ensemble des nœuds composant le chemin reliant le nœud $\mathbf{u}$ à la racine ε. Par construction, pour tout $j \in \{1,2\}$, nous avons :

$$\mathscr{L}_j^{(\mathbf{u})} = \left\{ \mathbf{x} \in \mathscr{L}_j \ : \ |\mathbf{x}_{B_{\mathbf{u}}}| = \frac{h_2^{-1}(1-\lambda)}{2} \sum_{i=1}^{s} n_i \right\} \tag{5.33}$$

avec $\#B_{\mathbf{u}} = \sum_{i=1}^{s} \frac{n_i}{2}$. En particulier, lorsque $\mathbf{u}$ est une feuille de l'arbre, nous avons :

$$\mathscr{L}_j^{(\mathbf{u})} = \left\{ \mathbf{x} \in \mathscr{L}_j \ : \ |\mathbf{x}_{B_{\mathbf{u}}}| = \frac{n h_2^{-1}(1-\lambda)}{2} \right\} \tag{5.34}$$

avec $\#B_{\mathbf{u}} = \frac{n}{2}$.

L'algorithme de May et Ozerov ne consiste pas seulement à rechercher les couples proches dans tous les $\mathscr{L}_1^{(\mathbf{u})} \times \mathscr{L}_2^{(\mathbf{u})}$ associés aux feuilles de $\mathscr{T}$. En effet, May et Ozerov imposent également que les couples proches $(\mathbf{x}^*, \mathbf{y}^*)$ recherchés vérifient :

$$\forall s \in [\![1,t]\!], \qquad \Delta(\mathbf{x}_s^*, \mathbf{y}_s^*) = \omega n_s \tag{5.35}$$

$$\forall s \in [\![1,t]\!], \qquad |\mathbf{x}_s^*| = |\mathbf{y}_s^*| = \frac{n_s}{2} \tag{5.36}$$

où $(\mathbf{x}_1^*, \cdots, \mathbf{x}_t^*)$ et $(\mathbf{y}_1^*, \cdots, \mathbf{y}_t^*)$ sont les écritures respectives de $\mathbf{x}^*$ et $\mathbf{y}^*$ dans le produit cartésien $\mathbb{F}_2^{n_1} \times \cdots \times \mathbb{F}_2^{n_t}$. Pour garantir les conditions (5.35) et (5.36), May et Ozerov itèrent leur procédure de construction de sous-listes en ajoutant deux phases de randomisation à chaque itération. La première consiste à appliquer une permutation aléatoire sur les éléments de $\mathscr{L}_1$ et $\mathscr{L}_2$ et la seconde à ajouter un vecteur aléatoire à ces mêmes éléments. Ainsi, nous espérons que les contraintes (5.35) et (5.36) sont vérifiées pour au moins une des itérations.

Finalement, le pseudo-code 5.5.1 rappelle l'algorithme de recherche de presque-collisions de [MO15].

Algorithme 5.5.1 : La méthode May-Ozerov

Entrées : deux listes $\mathscr{L}_1$ et $\mathscr{L}_2 \subseteq \mathbb{F}_2^n$ de taille $O\left(2^{\lambda n}\right)$;
une distance $w \in [\![0, \frac{n}{2}]\!]$.
Paramètres : un nombre d'itérations s ;
un réel $\varepsilon > 0$;
un entier t ;
des entiers $n_1, \cdots, n_t$ tels que $\sum_{s=1}^{t} n_s = n$;
des entiers strictement positifs $N_1, \cdots, N_t$.
Sortie : $\mathscr{L}^* = \{(\mathbf{x}, \mathbf{y}) \in \mathscr{L}_1 \times \mathscr{L}_2 \text{ tel que } \Delta(\mathbf{x}, \mathbf{y}) = w\}$.

$\mathscr{L}^* \leftarrow \emptyset$;
répéter M **fois**
  choisir une permutation π uniformément dans S_n ;
  choisir $\mathbf{v}$ uniformément dans $\mathbb{F}_2^n$;
  $\mathscr{L}'_1 \leftarrow \{\pi(\mathbf{x}) + \mathbf{v} \ : \ \mathbf{x} \in \mathscr{L}_1\}$;
  $\mathscr{L}'_2 \leftarrow \{\pi(\mathbf{x}) + \mathbf{v} \ : \ \mathbf{x} \in \mathscr{L}_2\}$;
  $\mathscr{L}'_1 \leftarrow \left\{\mathbf{x} \in \mathscr{L}'_1 \ : \ \forall s \in [\![1, t]\!], |\mathbf{x}_s| = \frac{n_s}{2}\right\}$;
  $\mathscr{L}'_2 \leftarrow \left\{\mathbf{x} \in \mathscr{L}'_2 \ : \ \forall s \in [\![1, t]\!], |\mathbf{x}_s| = \frac{n_s}{2}\right\}$;
    `/* où` $\mathbf{x} := (\mathbf{x}_1, \cdots, \mathbf{x}_t) \in \mathbb{F}_2^{n_1} \times \cdots \times \mathbb{F}_2^{n_t}$ `*/`
  $\mathscr{L}^*_{\text{tmp}} \leftarrow \text{MayOzerovRec}(\mathscr{L}'_1, \mathscr{L}'_2, 1)$;
  $\mathscr{L}^* \leftarrow \mathscr{L}^* \cup \left\{\left(\pi^{-1}(\mathbf{x}) + \mathbf{v}, \pi^{-1}(\mathbf{y}) + \mathbf{v}\right) \ : \ (\mathbf{x}, \mathbf{y}) \in \mathscr{L}^*_{\text{tmp}}\right\}$;
finRépéter
retourner $\mathscr{L}^*$;

Fonction *MayOzerovRec*$(\mathscr{L}'_1, \mathscr{L}'_2, s)$
  si $s = t + 1$ **alors**
    retourner $\{(\mathbf{x}, \mathbf{y}) \in \mathscr{L}'_1 \times \mathscr{L}'_2 \ : \ \Delta(\mathbf{x}, \mathbf{y}) = w\}$; `/* recherche exhaustive */`
  sinon
    $\mathscr{L}^*_{\text{tmp}} \leftarrow \emptyset$;
    répéter N_s **fois**
      choisir uniformément $A \in \left\{X \subseteq \left[\!\left[1 + \sum_{j=1}^{s-1} n_j, \sum_{j=1}^{s} n_j\right]\!\right] : \#X = \frac{n_s}{2}\right\}$;
      $\mathscr{L}''_1 \leftarrow \left\{\mathbf{x} \in \mathscr{L}'_1 \ : \ |\mathbf{x}_A| = \frac{n_s h_2^{-1}(1-\lambda)}{2}\right\}$; `/* recherche exhaustive */`
      $\mathscr{L}''_2 \leftarrow \left\{\mathbf{y} \in \mathscr{L}'_2 \ : \ |\mathbf{y}_A| = \frac{n_s h_2^{-1}(1-\lambda)}{2}\right\}$; `/* recherche exhaustive */`
      si $\#\mathscr{L}''_1$ *et* $\#\mathscr{L}''_2$ *sont de l'ordre de* $\tilde{O}\left(2^{\lambda\left(n - \sum_{j=1}^{s} n_j\right) + \varepsilon \frac{n}{2}}\right)$ **alors**
        $\mathscr{L}^*_{\text{tmp}} \leftarrow \mathscr{L}^*_{\text{tmp}} \cup \text{MayOzerovRec}(\mathscr{L}''_1, \mathscr{L}''_2, s + 1)$;
      finSi
    finRépéter
    retourner $\mathscr{L}^*_{\text{tmp}}$;
  finSi
finFonction

5.5.2 Analyse de la méthode May-Ozerov

Les tailles des listes et leurs nombres. Tout d'abord, il est montré dans [MO15, Lemma 3] que pour tout $s \in [\![1, t]\!]$, les listes associées aux nœuds de profondeur s dans l'arbre $\mathscr{T}$ sont de taille typiquement $\tilde{O}\left(2^{\lambda\left(n - \sum_{s=1}^{j} n_s\right)}\right)$. Dans l'algorithme May-Ozerov, nous gardons

toutes les listes dont la taille est de l'ordre de $\tilde{O}\left(2^{\lambda\left(n-\sum_{s=1}^{j} n_s\right)+\varepsilon\frac{n}{2}}\right)$; les autres listes sont simplement rejetées. En utilisant l'inégalité de Bienaymé-Tchebychev, May et Ozerov montrent que la probabilité qu'un couple de listes soit rejeté est inférieure à $2t \cdot 2^{-\varepsilon n}$; ce qui est négligeable lorsque n tend vers l'infini.

D'autre part, pour tout $s \in [\![1, t]\!]$, nous choisissons $N_s := \tilde{O}(2^{yn_s})$ où y est donné par l'équation (5.26). Ainsi, l'arbre $\mathscr{T}$ contient $\tilde{O}(2^{yn})$ feuilles. Il est montré dans [MO15, Lemma 2] que si la solution $(\mathbf{x}^*, \mathbf{y}^*)$ que nous recherchons vérifie les conditions (5.35) et (5.36), alors, avec une probabilité de l'ordre $1 - o(1)$, il existe une feuille $\mathbf{u}$ dans $\mathscr{T}$ telle que $\mathscr{L}_1^{(\mathbf{u})} \times \mathscr{L}_2^{(\mathbf{u})}$ contient $(\mathbf{x}^*, \mathbf{y}^*)$.

Notons que les listes associées aux feuilles de l'arbre sont de taille $\tilde{O}\left(2^{\varepsilon\frac{n}{2}}\right)$ et que ces listes sont les seules sur lesquelles nous testons les proximités des couples. Ainsi, dans l'algorithme 5.5.1, la recherche exhaustive des couples proches a un coût de l'ordre de $\tilde{O}\left(2^{(y+\varepsilon)n}\right)$. La construction de toutes les listes intermédiaires ne doit donc pas dépasser cette complexité.

La probabilité de succès d'une itération. Soit un couple solution $(\mathbf{x}^*, \mathbf{y}^*)$. D'après le paragraphe précédent, une itération de l'algorithme May-Ozerov opère avec succès lorsque les équations (5.35) et (5.36) sont vérifiées. Les étapes 3 à 8 permettent d'ajouter de l'aléa dans l'algorithme et ainsi espérer que les conditions exigées soient respectées pour au moins une itération.

Soient une permutation $\pi \in S_n$ et un vecteur $\mathbf{v} \in \mathbb{F}_2^n$ choisis uniformément. Dans [MO15, Lemma 1], il est montré que le couple $(\pi(\mathbf{x}^*) + \mathbf{v}, \pi(\mathbf{y}^*) + \mathbf{v})$ vérifie la contrainte (5.35) avec une probabilité :

$$\mathbb{P}_{(5.35)} = \frac{\prod_{s=1}^{t} \binom{n_s}{\omega n_s}}{\binom{n}{w}} = O\left(n^{-t/2}\right) \tag{5.37}$$

et la contrainte (5.36) avec une probabilité :

$$\mathbb{P}_{(5.36)} = \prod_{s=1}^{t} \frac{\binom{C_{00}^s}{\frac{1}{2}C_{00}^s}\binom{C_{10}^s}{\frac{1}{2}C_{10}^s}\binom{C_{01}^s}{\frac{1}{2}C_{01}^s}\binom{C_{11}^s}{\frac{1}{2}C_{11}^s}}{\binom{n_s}{\frac{1}{2}n_s}} = O\left(n^{-3t/2}\right) \tag{5.38}$$

avec pour tout $s \in [\![1, t]\!]$ et pour tous $\alpha, \beta \in \{0, 1\}$:

$$C_{\alpha\beta}^s := \#\left\{ i \in [\![1, n_s]\!] : x_{s,i} = \alpha \text{ et } y_{s,i} = \beta \right\} \tag{5.39}$$

où $\pi(\mathbf{x}^*) + \mathbf{v} := (\mathbf{x}_1, \cdots, \mathbf{x}_t) \in \mathbb{F}_2^{n_1} \times \cdots \times \mathbb{F}_2^{n_t}$ et pour tout $s \in [\![1, t]\!]$, $\mathbf{x}_s^* := (x_{s,1}, \cdots, x_{s,n_s}) \in \mathbb{F}_2^{n_s}$ (on procède de façon analogue avec $\mathbf{y}^*$).

La probabilité de succès d'une itération de l'algorithme May-Ozerov est donc :

$$\mathbb{P}_{\text{succ}} = \mathbb{P}_{(5.35)} \cdot \mathbb{P}_{(5.36)} = O\left(n^{-2t}\right) \tag{5.40}$$

La complexité de la méthode May-Ozerov. Finalement, le théorème 5.5.3 conclut l'analyse de l'algorithme May-Ozerov en donnant sa complexité asymptotique :

Théorème 5.5.3. *Pour tout $\lambda < 1 - h_2\left(\frac{\omega}{2}\right)$, la méthode May-Ozerov peut résoudre le problème des presque-collisions sur $\mathbb{F}_2^n$ en un temps de l'ordre de :*

$$T_{\text{MO}}^{\text{Aléa}} = 2^{O\left(\log(n)^2\right)} \cdot \tilde{O}(2^{yn}) \quad \textit{avec} \quad y := (1-\omega)\left(1 - h_2\left(\frac{h_2^{-1}(1-\lambda) - \frac{\omega}{2}}{1-\omega}\right)\right) \tag{5.41}$$

Démonstration du théorème 5.5.3.

Dans l'algorithme 5.5.1, $M := \tilde{O}\left(\frac{1}{\mathbb{P}_{\text{succ}}}\right)$ itérations suffisent pour résoudre le problème des presque-collisions. La complexité de la méthode est alors la suivante :

$$T_{\text{MO}}^{\text{Aléa}} = \frac{1}{\mathbb{P}_{\text{succ}}} \cdot \left(\sum_{s=1}^{t} \left(S_{s-1} \prod_{j=1}^{s} N_j \right) + S_t^2 \prod_{s=1}^{t} N_s \right) \tag{5.42}$$

avec $S_s := \tilde{O}\left(2^{\lambda(n-\sum_{j=1}^{s} n_j)+\varepsilon\frac{n}{2}}\right)$ la taille des listes du $s^{\text{ème}}$ étage de l'arbre $\mathscr{T}$ et $N_s := \tilde{O}(2^{yn_s})$ le nombre de nœuds héritant d'un nœud de l'étage $s-1$. Cette complexité est optimale lorsque :

$$S_0 \prod_{j=1}^{1} N_j = S_1 \prod_{j=1}^{2} N_j = \cdots = S_{t-1} \prod_{j=1}^{t} N_j = S_t^2 \prod_{j=1}^{t} N_j = \tilde{O}\left(2^{(y+\varepsilon)n}\right) \tag{5.43}$$

Ce qui nous donne les paramètres optimaux suivants :

$$t = \left\lceil \frac{\log_2(y-\lambda+\frac{\varepsilon}{2}) - \log_2(\frac{\varepsilon}{2})}{\log_2(y) - \log_2(\lambda)} \right\rceil \tag{5.44}$$

$$n_1 = \frac{\left(y-\lambda+\frac{\varepsilon}{2}\right) n}{y} \tag{5.45}$$

$$n_j = \frac{\lambda}{y} n_{j-1} \quad \forall j \in [\![2, t]\!] \tag{5.46}$$

L'équation (5.42) devient alors :

$$\begin{aligned} T_{\text{MO}}^{\text{Aléa}} &= \tilde{O}\left(\frac{2^{(y+\varepsilon)n}}{\mathbb{P}_{\text{succ}}}\right) \\ &= \tilde{O}\left(2^{\varphi(\varepsilon)+yn}\right) \qquad \text{avec } \varphi(\varepsilon) := \varepsilon n + 2t\log_2(n) \end{aligned}$$

En choisissant $\varepsilon := \frac{log_2(n)^2}{n}$, nous obtenons $t = O(\log_2(n))$ et $\varphi(\varepsilon) = O\left(log_2(n)^2\right)$; ce qui nous donne le surcoût super-polynomial du théorème. Nous pouvons montrer avec une analyse différentielle que ce choix est optimal. □

Dans [MO15], le facteur super-polynomial $2^{O\left(\log(n)^2\right)}$ est négligé. Cependant, ce facteur peut s'avérer parfois très handicapant en pratique.

***Meet-in-the-Middle* : une méthode de substitution.** Il faut noter que la méthode May-Ozerov ne peut pas être appliquée lorsque $\lambda \geqslant 1 - h_2^{-1}\left(\frac{\omega}{2}\right)$. De plus, pour certains jeux de paramètres la méthode May-Ozerov est peut être moins efficace que la méthode exhaustive pour résoudre le problème des presque-collisions. Dans [BM18], Both et May proposent un algorithme de secours qu'ils nomment *meet-in-the-middle* et que nous rappelons dans le pseudo-code 5.5.2.

La complexité de l'algorithme 5.5.2 est :

$$T_{\text{MITM}}^{\text{Aléa}} = O\left(\min\left(2^{2\lambda n}, \max\left(2^{\left(\lambda+h_2\left(\frac{\omega}{2}\right)\right)n}, 2^{(2\lambda-1+h_2(\omega))n}\right)\right)\right) \tag{5.47}$$

La figure 5.5 illustre alors les performances théoriques de la méthode May-Ozerov et de la méthode *meet-in-the-middle* en fonction de ω pour différentes tailles de liste. Sur cette figure, nous comparons ces méthodes avec la méthode des projections.

Algorithme 5.5.2 : La méthode *meet-in-the-middle* [BM18, Algo.4]

Entrées : $\mathscr{L}_1$ et $\mathscr{L}_2 \subseteq \mathbb{F}_2^n$;
une distance w paire.
Sortie : $\mathscr{L}^* = \{(\mathbf{x}, \mathbf{y}) \in \mathscr{L}_1 \times \mathscr{L}_2 \text{ tel que } \Delta(\mathbf{x}, \mathbf{y}) \leqslant w\}$.

si $\max\left(\binom{n}{\frac{w}{2}}2^{\lambda n}, \binom{n}{w}2^{(2\lambda-1)n}\right) < 2^{2\lambda n}$ **alors**
  $\mathscr{L}'_1 \leftarrow \varnothing$;
  pour tout $\mathbf{x} \in \mathscr{L}_1$ **faire**
    pour tout $\mathbf{e} \in \mathbb{F}_2^n$ *tel que* $|\mathbf{e}| \leqslant \frac{w}{2}$ **faire**
      ajouter $(\mathbf{x} + \mathbf{e}, \mathbf{x})$ dans $\mathscr{L}'_1$;
    finPour
  finPour
  pour tout $\mathbf{y} \in \mathscr{L}_2$ **faire**
    pour tout $\mathbf{e} \in \mathbb{F}_2^n$ *tel que* $|\mathbf{e}| \leqslant \frac{w}{2}$ **faire**
      si $(\mathbf{x} + \mathbf{e}, \mathbf{y}) \in \mathscr{L}'_1$ **alors**
        ajouter $(\mathbf{x}, \mathbf{y})$ dans $\mathscr{L}^*$;
      finSi
    finPour
  finPour
  retourner $\mathscr{L}^*$;
sinon
  effectuer une recherche exhaustive des solutions ;
finSi

5.6 L'approche de Gordon, Miller et Ostapenko

5.6.1 Le modèle de proximité de Gordon, Miller et Ostapenko

Dans l'article [GMO10], Gordon, Miller et Ostapenko s'intéressent au problème 5.2.7 dans les espaces binaires de Hamming. Rappelons que ce problème est une redéfinition du problème des presque-collisions 5.2.1 dans le modèle de proximité $\mathcal{M}_{\text{Abis}}\left(\mathbb{F}_2^n, 2^{\lambda n}, \mathcal{B}(\omega)\right)$ où $\lambda \in [0,1]$ et $\omega \in \left[0, \frac{1}{2}\right]$ sont deux constantes. Rappelons que dans ce modèle, nous supposons deux listes $\mathscr{L}_1 := \mathscr{L}'_1 \cup \{\mathbf{x}^*\}$ et $\mathscr{L}_2 := \mathscr{L}'_2 \cup \{\mathbf{y}^*\}$ avec :

— $\mathscr{L}'_1$ et $\mathscr{L}'_2$ suivent le modèle $\mathcal{M}_{\text{Aléa}}\left(\mathbb{F}_2^n, 2^{\lambda n}\right)$; c'est-à-dire que ces deux listes sont de tailles $O\left(2^{\lambda n}\right)$ et leurs éléments sont produits en tirant uniformément des vecteurs dans $\mathbb{F}_2^n$;

— $(\mathbf{x}^*, \mathbf{y}^*)$ est tel que $\mathbf{x}^*$ a été choisi uniformément dans $\mathbb{F}_2^n$ et les bits de $\mathbf{x}^* + \mathbf{y}^*$ ont été tirés selon une loi de Bernoulli de paramètre ω.

Le problème 5.2.7 consiste alors à retrouver le couples $(\mathbf{x}^*, \mathbf{y}^*)$. Nous supposons donc qu'il existe un test qui permet de vérifier que l'on est bien en présence du couple recherché.

Remarque 5.6.1. Dans [GMO10], le problème étudié n'est pas exactement le problème ci-dessus. En effet, à chaque instance du problème, nous connaissons un des deux membres du couple proche – disons $\mathbf{x}^*$ – et nous recherchons l'autre membre $\mathbf{y}^*$ qui a été inséré dans une liste $\mathscr{L}$ de vecteurs tirés uniformément dans $\mathbb{F}_2^n$. Toutefois, nous pouvons réduire ce problème au problème 5.2.7 moyennant le facteur multiplicatif $2^{\lambda n}$. Cette réduction s'apparente à celle que nous avons déjà effectuée entre le problème des voisins proches 5.1.4 et le problème des presque-collisions 5.2.1.

Gordon, Miller et Ostapenko utilisent aussi l'approche LSH pour résoudre leur version du problème de presque-collisions. Cependant, étant donné le modèle de proximité considéré, l'algorithme LSH 5.3.1 est légèrement modifié. En effet, il est supposé que celui-ci

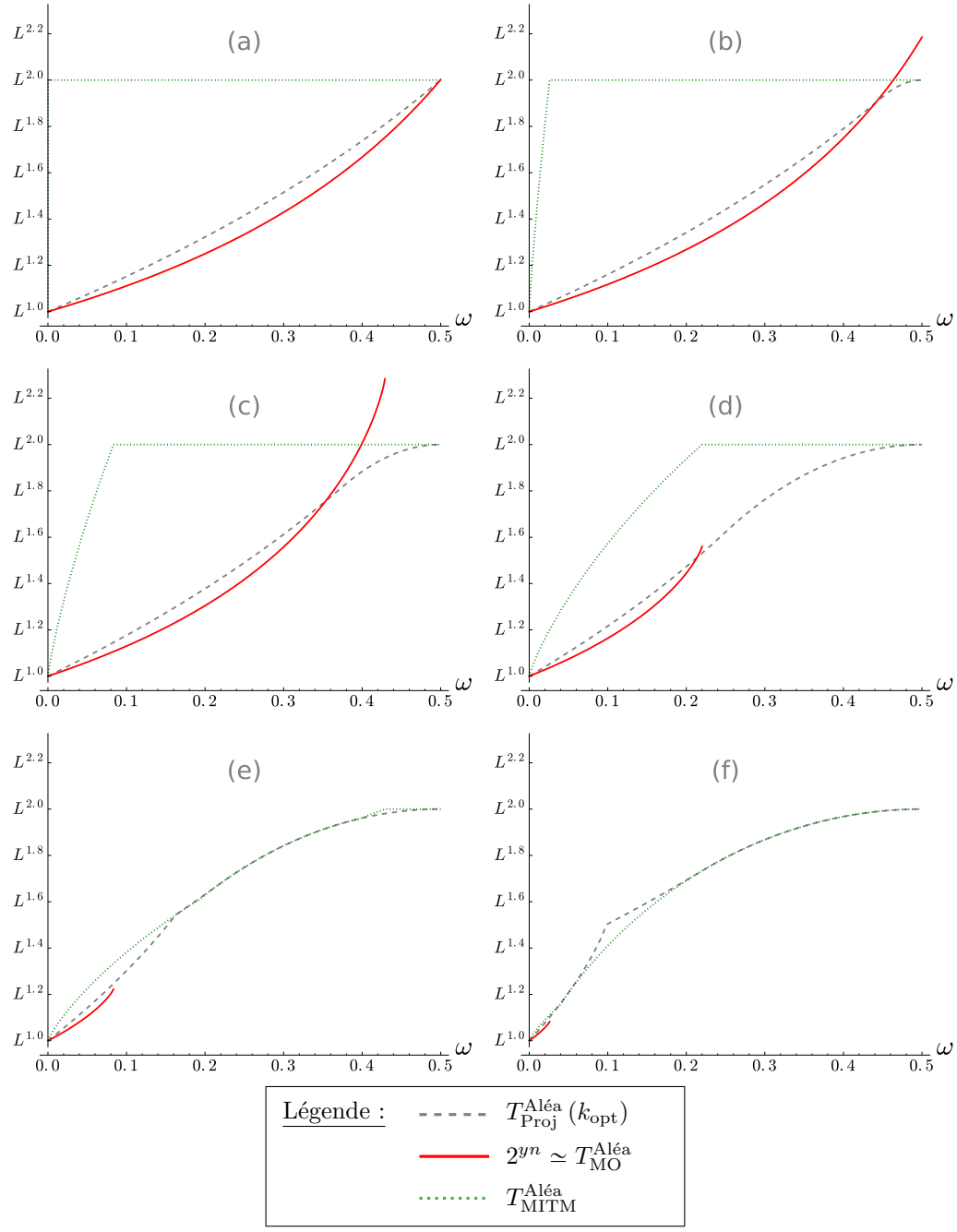


Figure 5.5 – Comparaison de la méthode May-Ozerov et de la méthode des projections pour la recherche de presque-collisions sur $\mathbb{F}_2^n$ dans le modèle de proximité $\mathcal{M}_{\text{Aléa}}\left(\mathbb{F}_2^n, 2^{\lambda n}\right)$. $L = 2^{\lambda n}$ avec : (a) $\lambda = 0.001$, (b) $\lambda = 0.1$, (c) $\lambda = 0.25$, (d) $\lambda = 0.5$, (e) $\lambda = 0.75$, (f) $\lambda = 0.9$.

s'arrête dès qu'il a trouvé le couple $(\mathbf{x}^*, \mathbf{y}^*)$; cela suppose un test qui vérifie qu'un couple quelconque $(\mathbf{x}, \mathbf{y})$ soit le couple recherché sans dévoiler ce dernier. Le pseudo-code 5.6.1 décrit alors leur méthode LSH adaptée au problème 5.2.7.

Algorithme 5.6.1 : La méthode LSH pour le problème 5.2.7

Entrée : une probabilité $\omega \in [0,1]$;
deux listes $\mathscr{L}_1 := \mathscr{L}'_1 \cup \{\mathbf{x}^*\}$ et $\mathscr{L}_2 := \mathscr{L}'_2 \cup \{\mathbf{y}^*\}$ suivant le modèle $\mathcal{M}_{\text{Abis}}\left(\mathbb{F}_2^n, 2^{\lambda n}, \mathcal{B}(\omega)\right)$;
Paramètres : un entier t ;
un entier T ;
une famille de fonctions LSH $\mathscr{F} \subseteq \{\mathscr{E} \to [\![1, T]\!]\}$
Sortie : le couple $(\mathbf{x}^*, \mathbf{y}^*)$.

répéter indéfiniment
  choisir $h_1, \cdots, h_t$ uniformément et indépendamment dans $\mathscr{F}$;
  initialiser une table de hachage $\mathscr{T}$ de taille T^t ;
  pour tout $\mathbf{x} \in \mathscr{L}_1$ **faire**
    ajouter $\mathbf{x}$ dans $\mathscr{T}[g(\mathbf{x})]$; /* où $g(\mathbf{x}) := 1 + \sum_{i=1}^{t}(h_i(x)-1)T^{i-1}$ */
  finPour
  pour tout $\mathbf{y} \in \mathscr{L}_2$ **faire**
    pour tout $\mathbf{x} \in \mathscr{T}[g(\mathbf{y})]$ **faire**
      si $(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^*, \mathbf{y}^*)$ **alors**
        retourner $(\mathbf{x}, \mathbf{y})$;
      finSi
    finPour
  finPour
finRépéter

Remarquons que $\Delta(\mathbf{x}^*, \mathbf{y}^*)$ suit une loi binomiale de paramètre (n, ω). L'inégalité de Bienaymé-Tchevychev nous permet alors de majorer cette distance par une distance $\overline{\omega}$ avec une probabilité de l'ordre de $(1 - o(1))$. Ainsi, une analyse de l'algorithme LSH original 5.3.1 pour rechercher tous les couples à distance $\overline{\omega}$ dans $\mathscr{L}_1 \times \mathscr{L}_2$ nous donne une analyse en temps *amorti* de l'algorithme 5.6.1. Cependant, dans [GMO10], les auteurs proposent une analyse asymptotique *en moyenne* sur les instances du modèle de proximité. Ils se permettent donc de considérer le couple $(\mathbf{x}^*, \mathbf{y}^*)$ comme source d'aléa dans leur analyse.

Remarque 5.6.2. Une analyse en moyenne sur les instances du modèle de proximité de la méthode de [GMO10] donne la complexité en temps amorti dans le contexte où les listes $\mathscr{L}_1$ et $\mathscr{L}_2$ peuvent être renouvelées à n'importe quel instant ; notamment à chaque itération de l'algorithme LSH. Ce contexte très particulier peut être pertinent pour certaines applications telles que la reconnaissance de codes.

Dans leur papier, Gordon, Miller et Ostapenko comparent des familles de fonctions LSH construitent à partir de codes correcteurs d'erreurs avec des familles de projections. Tout d'abord, nous donnons une analyse en moyenne de la méthode des projections qui correspond au modèle de proximité $\mathcal{M}_{\text{Abis}}\left(\mathbb{F}_2^n, 2^{\lambda n}, \mathcal{B}(\omega)\right)$. Soit l'algorithme 5.6.1 instancié avec la famille $\mathscr{F}_{\text{Proj}}(\ell)$ des projections sur $\ell := \lceil \lambda n \rceil$ positions. La complexité de la méthode ainsi obtenue est :

$$T_{\text{Proj}}^{\text{Abis}}(\ell) = O\left(2^{\lambda n(1-\log_2(1-\omega))}\right) \tag{5.48}$$

5.6.2 Les codes parfaits dans l'approche LSH

Gordon, Miller et Ostapenko abordent eux aussi le problème des presque-collisions avec l'approche LSH mais plutôt que d'utiliser des projections comme fonctions de hachage localement sensibles, ils proposent d'utiliser des décodages complets de codes correcteurs d'erreurs. Rappelons qu'une fonction LSH doit partitionner l'espace ambiant ; or les codes

parfaits permettent justement un partitionnement de l'espace ambiant en parties de même taille. C'est pourquoi il est proposé dans [GMO10] de construire des fonctions LSH à partir de ces codes. L'idée d'utiliser des codes parfaits pour résoudre des problèmes de proximité avait déjà été proposée dans [DHL+94] ou [BE98]. Cependant, il n'était pas clair à l'époque que ces méthodes soient plus performantes que les méthodes classiques comme par exemple celle des projections.

Soit $\mathcal{C}$ un code linéaire binaire parfait $[n, k]$ et soit $\mathscr{F}_{\mathcal{C}}$ l'ensemble des décodages complets à maximum de vraisemblance des cosets de $\mathcal{C}$:

$$\mathscr{F}_{\mathcal{C}} := \left\{ \begin{array}{ccc} \mathbb{F}_2^n & \to & \mathcal{C} \sim [\![1, 2^k]\!] \\ \mathbf{x} & \mapsto & D_{\mathcal{C}}(\mathbf{x} + \mathbf{u}) \end{array} \; : \; \mathbf{u} \in \mathbb{F}_2^n \right\} \tag{5.49}$$

où $D_{\mathcal{C}}$ est un décodage complet à maximum de vraisemblance de $\mathcal{C}$.

Dans [GMO10], Gordon, Miller et Ostapenko proposent une méthode pour résoudre leur version du problème des presque-collisions. Leur algorithme est essentiellement l'algorithme 5.3.1 instancié avec la famille $\mathscr{F}_{\mathcal{C}}$ où $\mathcal{C}$ est un code parfait de dimension $k = \ell := \lceil \lambda n \rceil$. Il est aussi fixé $t = 1$. La complexité de la méthode ainsi obtenue est :

$$T_{\text{Codes}}^{\text{Abis}}(\mathcal{C}) = O\left(\frac{2^{\lambda n}}{P_{\mathcal{C}}(\omega)}\right) \tag{5.50}$$

où $P_{\mathcal{C}}(\omega)$ est la probabilité que $h(\mathbf{x}) = h(\mathbf{y})$ pour une fonction h tirée uniformément dans $\mathscr{F}_{\mathcal{C}}$ et un couple $(\mathbf{x}, \mathbf{y})$ tel que les bits de $\mathbf{x} + \mathbf{y}$ suivent une loi de Bernoulli de paramètre ω. Attention, $P_{\mathcal{C}}$ n'est pas exactement la fonction de collision telle que définie dans 5.3.1.

Nous avons vu dans la section 1.2 que les codes linéaires parfaits sont peu nombreux. Ces codes ne permettent donc de traiter que certaines instances très particulières du problème des presque-collisions. Dans [GMO10], les auteurs proposent de construire une famille de fonctions LSH en utilisant tout d'abord le code de Golay binaire puis les codes de Hamming binaires. Le tableau 5.1 donne la probabilité $P_{\mathcal{C}}(\omega)$ de collision de deux mots proches pour une fonction de $\mathscr{F}_{\mathcal{C}}$ où $\mathcal{C}$ est un code parfait. La dernière colonne indique la région de ω pour laquelle la méthode des projections est moins efficace que la méthode des codes parfaits (quelque soit la taille des listes du problème de presque-collisions).

$\mathcal{C}$	$P_{\mathcal{C}}(\omega)$	$T^{\text{Abis}}_{\text{Codes}}(\mathcal{C}) \leqslant T^{\text{Abis}}_{\text{Codes}}(\lambda)$
code de Golay binaire $[23,12]_2$	$\frac{(1-\omega)^{23}}{2^{11}} A(\frac{\omega}{1-\omega})$ où $A(X) := 2048 + 11684X + 128524X^2 + 226688X^3 + 1133440X^4 + 672980X^5 + 2018940X^6$	$\omega \in [0.2555, 0.5]$
code de Hamming binaire $[15,11]_2$	$\frac{(1-\omega)^{15}}{8} \cdot \left(8 + \frac{15\omega}{1-\omega} + \left(\frac{105\omega^2}{(1-\omega)^2}\right)\right)$	$\omega \in [0.2826, 0.5]$
code de Hamming binaire $[31,26]_2$	$\frac{(1-\omega)^{31}}{16} \cdot \left(16 + \frac{31\omega}{1-\omega} + \left(\frac{465\omega^2}{(1-\omega)^2}\right)\right)$	$\omega \in [0.1518, 0.5]$
code de Hamming binaire $[63,57]_2$	$\frac{(1-\omega)^{63}}{32} \cdot \left(32 + \frac{63\omega}{1-\omega} + \left(\frac{1953\omega^2}{(1-\omega)^2}\right)\right)$	$\omega \in [0.0838, 0.5]$
code de Hamming binaire $[127,120]_2$	$\frac{(1-\omega)^{127}}{64} \cdot \left(64 + \frac{127\omega}{1-\omega} + \left(\frac{8001\omega^2}{(1-\omega)^2}\right)\right)$	$\omega \in [0.0468, 0.5]$
code de Hamming binaire $[2^m-1, 2^m-m-1]_2$ $(m>4)$	$\frac{(1-\omega)^{2^m-1}}{2^m} \cdot \left(2^m + 2(2^m-1)\frac{\omega}{1-\omega} + (2^m-1)(2^m-2)\left(\frac{\omega}{1-\omega}\right)^2\right)$	$\omega \in \left[\frac{m}{2^m-m}, \frac{1}{2}\right]$

Tableau 5.1 – Probabilité de collision de deux mots proches pour une fonction de $\mathscr{F}_{\mathcal{C}}$ et critère nécessaire pour que $T^{\text{Abis}}_{\text{Codes}}(\mathcal{C}) \leqslant T^{\text{Abis}}_{\text{Codes}}(\lambda)$.

5.6.3 Les codes aléatoires dans l'approche LSH

Dans [GMO10], Gordon, Miller et Ostapenko généralisent leur approche des codes pour rechercher des presque-collisions. Au lieu d'utiliser des codes linéaires parfaits, ils proposent d'utiliser des codes tirés uniformément dans l'ensemble des codes binaires de longueur n et de cardinalité $2^{\lambda n}$. Ces codes ne sont pas nécessairement linéaires.

Soit $\mathscr{F}_{\mathcal{R}}$ la famille de fonctions suivante :

$$\mathscr{F}_{\mathcal{R}} := \left\{ \begin{array}{ccc} \mathbb{F}_2^n & \to & \mathcal{C} \sim [\![1, 2^{\lambda n}]\!] \\ \mathbf{x} & \mapsto & D_{\mathcal{C}}(\mathbf{x}) \end{array} \quad : \ \mathcal{C} \subseteq \mathbb{F}_2^n \text{ et } \#\mathcal{C} = 2^{\lambda n} \right\} \tag{5.51}$$

où $D_{\mathcal{C}}$ est un décodage complet à maximum de vraisemblance de $\mathcal{C}$; c'est-à-dire que pour tout $\mathbf{x} \in \mathbb{F}_2^n$, $D_{\mathcal{C}}(\mathbf{x})$ est un mot $\mathbf{c} \in \mathcal{C}$ tel que $\Delta(\mathbf{x}, \mathbf{c}) = \min_{\mathbf{c}' \in \mathcal{C}} \left(\Delta\left(\mathbf{x}, \mathbf{c}'\right)\right)$. Pour une fonction de hachage tirée uniformément dans $\mathscr{F}_{\mathcal{R}}$, les mots des listes $\mathscr{L}_1$ et $\mathscr{L}_2$ seront équitablement répartis dans la table de hachage et donc le nombre de comparaisons à effectuer sera $O(1)$. La complexité de la méthode sera donc la même que pour les codes parfaits ; à savoir :

$$T^{\text{Abis}}_{\text{Codes}}(\mathscr{F}_{\mathcal{R}}) = O\left(\frac{2^{\lambda n}}{P_{\mathcal{R}}(\omega)}\right) \tag{5.52}$$

où $P_{\mathcal{R}}(\omega)$ est la probabilité que $h(\mathbf{x}) = h(\mathbf{y})$ pour une fonction h tirée uniformément dans $\mathscr{F}_{\mathcal{R}}$ et un couple $(\mathbf{x}, \mathbf{y})$ tel que les bits de $\mathbf{x} + \mathbf{y}$ suivent une loi de Bernoulli de paramètre ω. De plus, pour tout mot de code $\mathbf{c}$ et $\mathbf{x} \in \mathbb{F}_2^n$, si $\Delta(\mathbf{x}, \mathbf{c})$ est plus petite que la distance de Gilbert-Varshamov, alors on aura typiquement $h(\mathbf{x}) = \mathbf{c}$. Cela permet à Gordon, Miller et Ostapenko de montrer que :

$$P_{\mathcal{R}}(\omega) > \sum_{i=0}^{d_{\text{GV}}(n,\lambda n)} \binom{d_{\text{GV}}(n,\lambda n)}{i} \binom{n - d_{\text{GV}}(n,\lambda n)}{i} \omega^{2i}(1-\omega)^{n-2i} \tag{5.53}$$

où, rappelons le, $d_{\mathrm{GV}}(n, \lambda n) := O\big(nh_2^{-1}(1-\lambda)\big)$. La formule de Stirling et une analyse différentielle nous donne alors :

$$T^{\mathrm{Abis}}_{\mathrm{Codes}}(\mathscr{F}_{\mathcal{R}}) = O\big(2^{\lambda n \alpha}\big) \tag{5.54}$$

avec :

$$\alpha := 1 + \frac{1}{\lambda}\left(\delta h_2\left(\tfrac{\varepsilon}{2\delta}\right) + (1-\delta)h_2\left(\tfrac{\varepsilon}{2(1-\delta)}\right) + \varepsilon\log_2(\omega) + (1-\varepsilon)\log_2(1-\omega)\right)$$

$$\delta := \tfrac{d_{\mathrm{GV}}(n,\lambda n)}{n} \qquad \text{et} \qquad \varepsilon := \min\left(\tfrac{\omega\sqrt{\omega^2+4\delta(1-\delta)(1-2\omega)}-\omega^2}{1-2\omega}\ ,\ 2\delta(1-\delta)\right)$$

Remarque 5.6.3. Lorsque $\lambda = o(1)$, nous avons $\alpha := \frac{1}{1-\omega}$.

La figure 5.6 compare $T^{\mathrm{Abis}}_{\mathrm{Proj}}(\ell)$ et $T^{\mathrm{Abis}}_{\mathrm{Codes}}(\mathscr{F}_{\mathcal{R}})$ pour différentes valeurs de λ.

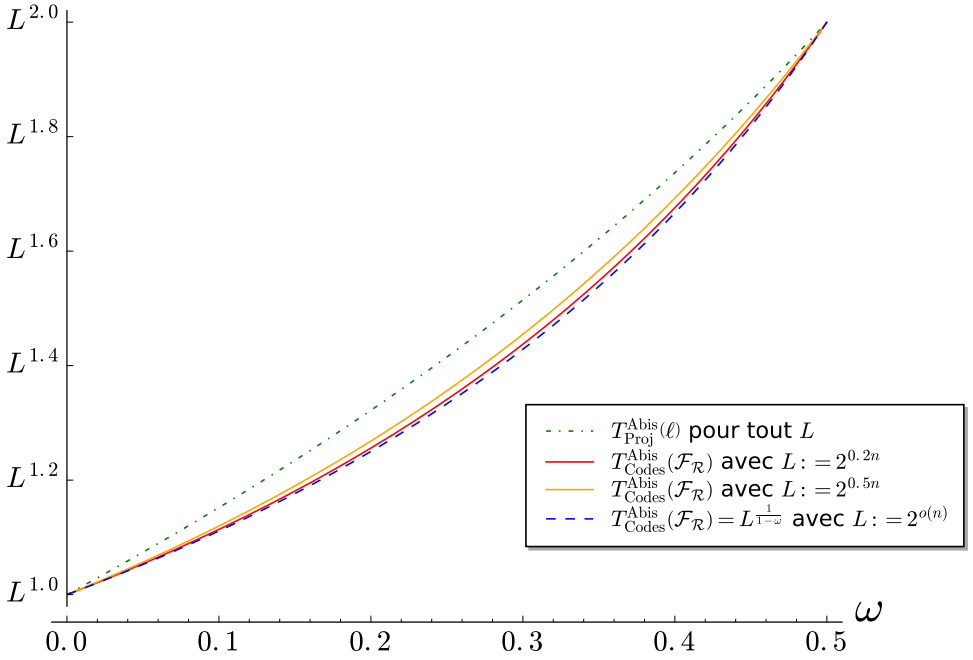


Figure 5.6 – Comparaison de la méthode des projections et de la méthode des codes pour la résolution du problème 5.2.7 dans les espaces binaires de Hamming ($L := 2^{\lambda n} := 2^{\ell}$).

La figure 5.6 semble montrer que la méthode des codes appliquée au modèle de proximité du papier [GMO10] résout le problème des presque-collisions plus efficacement que la méthode des projections. Gordon, Miller et Ostapenko prouvent cette conjecture dans leur papier.

5.6.4 Le produit cartésien de codes

Dans la section précédente, nous avons ignoré un problème majeur. En effet, nous avons implicitement supposé que pour tout code, nous étions en mesure de fournir un algorithme de décodage sous-exponentiel qui décode n'importe quel mot de l'espace à la distance de Gilbert-Varshamov. Malheureusement, un tel décodage n'existe pas. Cependant, Gordon, Miller et Ostapenko proposent d'utiliser une sous-famille de codes pour lesquels nous avons un décodage en temps linéaire ; à savoir, des produits cartésiens de codes.

Théorème 5.6.1. *Soit $(\mathscr{F}_i)_{i\in[\![1,t]\!]}$, t familles de fonctions de hachage. On suppose que pour tout $i \in [\![1,t]\!]$, les fonctions de $\mathscr{F}_i$ sont définies sur $\mathbb{F}_2^{n_i}$ et ont pour image $[\![1,L_i]\!]$. On note $P_i(\omega)$ la probabilité que $h_i(\mathbf{x}_i) = h_i(\mathbf{y}_i)$ pour h_i tiré uniformément dans $\mathscr{F}_i$ et $(\mathbf{x}_i,\mathbf{y}_i) \in \mathbb{F}_2^{n_i}\times\mathbb{F}_2^{n_i}$ tel que chacun des bits de $\mathbf{x}_i+\mathbf{y}_i$ suit une loi de Bernoulli de paramètre ω.*

Soit $\mathscr{F}$ la famille de fonctions suivante :

$$\mathscr{F} := \left\{ \begin{array}{ccc} \mathbb{F}_2^{n_1}\times\cdots\times\mathbb{F}_2^{n_t} & \to & [\![1,L_1]\!]\times\cdots\times[\![1,L_t]\!] \\ (\mathbf{x}_1,\cdots,\mathbf{x}_t) & \mapsto & (h_1(\mathbf{x}_1),\cdots,h_t(\mathbf{x}_t)) \end{array} : \forall i\in[\![1,t]\!], h_i\in\mathscr{F}_i \right\} \tag{5.55}$$

Soit h tirée uniformément dans $\mathscr{F}$ et $\mathbf{x},\mathbf{y}\in\mathbb{F}_2^{n_1}\times\cdots\times\mathbb{F}_2^{n_t}$ tels que les bits de $\mathbf{x}+\mathbf{y}$ suivent une loi de Bernoulli de paramètre ω. La probabilité que $h(\mathbf{x}) = h(\mathbf{y})$ est alors $\prod_{i=1}^{t} P_i(\omega)$.

Soit $t = \left\lceil \frac{\lambda n}{\log(n)} \right\rceil$. Le décodage à maximum de vraisemblance du produit cartésien de t codes de longueur $\frac{n}{t}$ et de taille $\left\lceil 2^{\frac{\lambda n}{t}} \right\rceil$ a un coût linéaire en n. En effet, un tel décodage consiste essentiellement à décoder exhaustivement dans chacun des codes ; or par construction, les codes constituants sont de taille linéaire en n. On peut ainsi construire une famille de fonctions de hachage dont chacune des fonctions sera un décodage à maximum de vraisemblance du produit cartésien de codes aléatoires :

$$\mathscr{F}_{\mathcal{R}(t)} := \left\{ \begin{array}{ccc} \left(\mathbb{F}_2^{\frac{n}{t}}\right)^t & \to & \mathcal{C}_1\times\cdots\times\mathcal{C}_t \\ (\mathbf{x}_1,\cdots,\mathbf{x}_t) & \mapsto & (D_{\mathcal{C}_1}(\mathbf{x}_1),\cdots,D_{\mathcal{C}_t}(\mathbf{x}_t)) \end{array} : \forall i\in[\![1,t]\!], \mathcal{C}_i\subseteq\mathbb{F}_2^{\frac{n}{t}} \text{ et } \#\mathcal{C}_i = \left\lceil 2^{\frac{\lambda n}{t}} \right\rceil \right\} \tag{5.56}$$

où pour tout $i\in[\![1,t]\!]$, $D_{\mathcal{C}_i}$ est un décodage complet à maximum de vraisemblance de $\mathcal{C}_i$. Grâce au théorème 5.6.1, on montre alors que l'on peut atteindre la complexité prétendue par l'équation (5.54) en utilisant la famille $\mathscr{F}_{\mathcal{R}(t)}$ dans l'algorithme 5.6.1.

Dans [Dub10], Dubiner s'intéresse au problème de recherche de presque-collisions dans le même modèle que [GMO10]. Sa méthode est essentiellement celle que nous venons de décrire ; cependant, son analyse n'est valable que pour des tailles de liste sous-exponentielles en n.

Remarque 5.6.4. On peut utiliser le théorème 5.6.1 pour construire des familles de fonctions LSH à partir de produits cartésiens de codes parfaits et ainsi étendre les résultats de la table 5.1 à d'autres longueurs n.

L'approche des codes dans les modèles $\mathcal{M}_{\mathsf{IT}}\left(\mathbb{F}_2^n, 2^{\lambda n}, (1+\varepsilon)w\right)$ et $\mathcal{M}_{\mathsf{Aléa}}\left(\mathbb{F}_2^n, 2^{\lambda n}\right)$. Que ce soit dans le modèle de proximité $\mathcal{M}_{\mathrm{IT}}\left(\mathbb{F}_2^n, 2^{\lambda n}, (1+\varepsilon)w\right)$ ou $\mathcal{M}_{\mathrm{Aléa}}\left(\mathbb{F}_2^n, 2^{\lambda n}\right)$, les couples proches ne suivent pas la même loi que dans le modèle de proximité de Gordon, Miller et Ostapenko. Cependant, la famille de fonctions $\mathscr{F}_{\mathcal{R}(t)}$ composée de décodages complets à maximum de vraisemblance de produits cartésiens de codes aléatoires est une famille LSH. Elle peut donc être utilisée dans l'algorithme 5.3.1 pour résoudre le problème des presque-collisions dans ces deux modèles de proximité. Cependant, il faut bien noter que le théorème 5.6.1 n'est valable que dans le modèle de proximité de [GMO10]. Dans les modèles $\mathcal{M}_{\mathrm{IT}}\left(\mathbb{F}_2^n, 2^{\lambda n}, (1+\varepsilon)w\right)$ ou $\mathcal{M}_{\mathrm{Aléa}}\left(\mathbb{F}_2^n, 2^{\lambda n}\right)$, il ne suffit pas de multiplier les fonctions de collision entre elles pour obtenir la fonction de collision d'un produit cartésien de familles LSH. L'analyse de la méthode est alors très différente de celles de [GMO10] ou [Dub10]. Dans la partie suivante, nous proposerons une méthode qui s'inspire de la méthode des codes. Nous ferons alors une analyse du temps amorti de notre méthode.

Chapitre 6

Décodage en liste et recherche de presque et lointaines collisions

6.1 Une généralisation de l'approche LSH

Le problème auquel nous nous intéressons dans ce chapitre est essentiellement le problème de recherche de presque-collisions 5.2.1 de la section 5.2 dans le modèle de proximité aléatoire $\mathcal{M}_{\text{Aléa}}(\mathscr{E}, L)$ (cf. section 5.2.3) où les listes $\mathscr{L}_1$ et $\mathscr{L}_2$ sont une seule et même liste de taille L. Le choix de ne considérer qu'une seule liste plutôt que deux nous permet de simplifier notre propos mais ne change pas l'essence du problème.

Dans ce chapitre, nous ne précisons pas l'espace métrique $\mathscr{E}$. Toutefois, nous émettons certaines hypothèse sur celui-ci ; par exemple, nous admettons qu'un tirage uniforme a du sens sur $\mathscr{E}$. D'autre part, nous définissons une notion d'admissibilité pour les distances sur $\mathscr{E}$:

Définition 6.1.1 (distance admissible)**.** *Une distance w est dite* admissible *sur l'espace métrique $\mathscr{E}$ s'il existe $\mathbf{x}, \mathbf{y} \in \mathscr{E}$ tels que $\Delta(\mathbf{x}, \mathbf{y}) = w$.*

Finalement, nous reformulons le problème des presque-collisions comme suit :

Problème 6.1.2 (presque-collisions à une liste)**.** *Soit un espace métrique $(\mathscr{E}, \Delta)$ probabilisé avec la probabilité uniforme et soit une distance admissible w et un entier L. Étant donné une liste $\mathscr{L}$ de L éléments tirés uniformément dans $\mathscr{E}$, trouver les couples $(\mathbf{x}, \mathbf{y}) \in \mathscr{L}^2$ tels que $\Delta(\mathbf{x}, \mathbf{y}) \leqslant w$.*

Nous supposons que $\mathscr{E}$ est un espace soumis au *fléau de la dimension* (cf. sous-section 5.1.2) ; nous supposons donc que $\mathscr{E}$ est associé à une notion de dimension que nous notons n.

Dans toute la suite, nous considérons qu'un algorithme résout le problème 6.1.2 dès lors que l'espérance du nombre de couples proches qu'il trouve est de l'ordre de $\Theta\left(L^2 \cdot \mathbb{P}\left(\Delta(\mathbf{u}, \mathbf{v}) \leqslant w\right)\right)$ pour $\mathbf{u}$ et $\mathbf{v}$ tirés uniformément dans $\mathscr{E}$. Cette quantité correspond au nombre de solutions du problème.

Une recherche exhaustive trouve tous les couples proches en un temps de l'ordre de $O\left(L^2\right)$. Nous souhaitons donc construire un algorithme qui résout le problème 6.1.2 en un temps sous-quadratique en L.

Une instance particulière du problème 6.1.2 est celle où $w = 0$. Dans ce cas, la recherche de presque-collisions est la recherche de collisions exactes. Nous avons vu au début du chapitre 5 qu'en utilisant des tables de hachages, nous pouvons trouver toutes les collisions dans $\mathscr{L}^2$ en un temps de l'ordre de $O(L)$. L'algorithme est simple, il consiste à stocker les éléments de $\mathscr{L}$ dans une table de hachage de taille $O(L)$. Si deux éléments sont égaux, alors

ils sont placés à la même adresse dans la table de hachage tandis que s'ils sont différents, ils sont placés dans des adresses différentes avec une bonne probabilité dès lors que la sortie de la fonction de hachage est indistinguable d'une distribution uniforme.

Notation 6.1.3. *Afin de nous référer aisément aux résultats du chapitre précédent, nous uniformisons nos notations pour la complexité de nos algorithmes de recherche de presque-collisions. Nous notons alors nos complexités de la façon suivante :*

$$C^{\text{Mod}}_{\text{Algo}}(\text{params}) \tag{6.1}$$

où :

- $C = S$ *lorsque l'on parle de complexité mémoire et* $C = T$ *lorsque l'on parle de complexité en temps ;*
- Algo *fait référence à la méthode utilisée ;*
- Mod *indique le modèle de proximité suivi par la liste* $\mathscr{L}$ *;*
- params *est une liste de paramètres afférents à la méthode* Algo *(facultatif).*

À titre d'exemple, les complexités mémoire et en temps de la recherche exhaustive pour le problème 6.1.2 sont respectivement :

$$S^{\text{Aléa}}_{\text{Exhaust}} = O(L) \tag{6.2}$$
$$T^{\text{Aléa}}_{\text{Exhaust}} = O\big(L^2\big) \tag{6.3}$$

Nous avons vu dans le chapitre 5 que l'utilisation de tables de hachage peut être étendue à la recherche de presque-collisions grâce aux familles de fonctions LSH (cf. section 5.3.1). Diverses méthodes LSH utilisant des décodages de codes correcteurs ont été proposés dans la littérature. En pratique, ces méthodes sont instanciables si et seulement si nous avons connaissance de codes et de leurs décodages qui soient à la fois efficaces et performants :

- décodages en temps polynomial ;
- distance moyenne de décodage proche de celle d'un décodage à maximum de vraisemblance d'un code aléatoire.

Les avancées récentes dans le domaine des codes correcteurs d'erreurs nous permettent d'aller toujours plus loin dans la qualité du rapport efficacité/performance des décodeurs. Les décodages en liste et plus particulièrement les décodages en liste des codes polaires ont notamment permis de faire un bond dans ce sens. Nous généralisons l'approche des codes pour la recherche de presque-collisions en ne considérant non plus seulement des décodages simples mais aussi des décodages en liste.

Avant de parler de codes, nous commençons par généraliser l'approche LSH. Cette fois-ci, nos fonctions de hachage floues retournent un ensemble d'adresses de la table de hachage. Plus formellement, nous définissons une famille $\mathscr{F}$ de fonctions de hachage définies sur $\mathscr{E}$ et retournant un ensemble d'adresses dans $[\![1, T]\!]$. Remarquons qu'une fonction de $\mathscr{F}$ peut aussi bien retourner un ensemble à un seul élément qu'un ensemble vide ou même un ensemble de taille exponentielle en la dimension n de $\mathscr{E}$.

Le pseudo-code 6.1.1 décrit alors notre algorithme pour résoudre le problème 6.1.2.

Algorithme 6.1.1 : Recherche de presque-collisions avec hachage en liste

Entrées : une liste $\mathscr{L} \subseteq \mathscr{E}$ de taille L ;
une distance admissible w ;
Paramètres : un nombre d'itérations s ;
un entier T ;
une famille de fonctions $\mathscr{F} \subseteq \{\mathscr{E} \to \mathscr{P}([\![1, T]\!])\}$;
`/* où` $\mathscr{P}([\![1,T]\!])$ `est l'ensemble des parties de` $[\![1,T]\!]$
`*/`
Sortie : $\mathscr{L}^* \subseteq \{(\mathbf{x}, \mathbf{y}) \in \mathscr{L}^2 \ : \ \Delta(\mathbf{x}, \mathbf{y}) \leqslant w\}$.

```
1  L* ← ∅ ;
2  répéter s fois
3  |  choisir h uniformément dans F ;
4  |  initialiser une table de hachage chainée T de taille T ;
5  |  pour tout x ∈ L faire
6  |  |  pour tout i ∈ h(x) faire
7  |  |  |  pour tout y ∈ T[i] faire
8  |  |  |  |  si Δ(x, y) ⩽ w alors
9  |  |  |  |  |  ajouter (x, y) dans L* ;
10 |  |  |  |  finSi
11 |  |  |  finPour
12 |  |  |  ajouter x dans la liste T[i] ;
13 |  |  finPour
14 |  finPour
15 finRépéter
16 retourner L* ;
```

Pour toute famille de fonctions $\mathscr{F} \subseteq \{\mathscr{E} \to \mathscr{P}([\![1, T]\!])\}$, nous définissons les trois quantités E_1, E_2 et P^* suivantes :

$$E_1 := \mathbb{E}(\#h(\mathbf{x})) \tag{6.4}$$
$$E_2 := \mathbb{E}(\#(h(\mathbf{x}) \cap h(\mathbf{y}))) \tag{6.5}$$
$$P^* := \mathbb{P}(h(\mathbf{x}^*) \cap h(\mathbf{y}^*) \neq \varnothing) \tag{6.6}$$

où h est choisie uniformément dans $\mathscr{F}$, $\mathbf{x}$ et $\mathbf{y}$ sont choisis uniformément dans $\mathscr{E}$ et $\mathbf{x}^*$ et $\mathbf{y}^*$ sont tels que $\Delta(\mathbf{x}^*, \mathbf{y}^*) = w$.

Lemme 6.1.4. *Soit $\mathscr{F}$ une famille de fonctions de hachage et soient E_1, E_2 et P^* définis respectivement par les équations (6.4), (6.5) et (6.6). Nous supposons que pour tout $h \in \mathscr{F}$ et $\mathbf{x} \in \mathscr{E}$, l'ensemble $h(\mathbf{x})$ se détermine en un temps de l'ordre de $O(\max(1, E_1))$. L'algorithme 6.1.1 peut alors résoudre le problème 6.1.2 en un temps de l'ordre de :*

$$T_{\text{Codes}}^{\text{Aléa}}(T, \mathscr{F}) = \tilde{O}\left(\frac{L \cdot \max(1, E_1) + L^2 \cdot E_2}{P^*}\right) \tag{6.7}$$

Démonstration du lemme 6.1.4.

Premièrement, pour chaque couple proche, la probabilité de le trouver est au pire P^* donc en effectuant $s := \frac{1}{P^*}$ itérations dans l'algorithme 6.1.1, nous pouvons espérer trouver un nombre de couples de l'ordre du nombre de solutions du problème.

Deuxièmement, pour chacune des s itérations, le coût pour hacher tous les éléments de $\mathscr{L}$ est de l'ordre de $O(L \cdot \max(1, E_1))$ et nous devons vérifier la proximité de $O(L^2 \cdot E_2)$ couples. □

6.2 Analyse de décodeurs en liste idéaux

6.2.1 Un hachage flou par décodage en liste

Nous proposons ici une construction idéale de famille de fonctions de hachage floues en utilisant des codes aléatoires. Plus exactement, soit un entier T et soit un code $\mathcal{C}$ constitué d'une liste de T mots tirés uniformément dans $\mathscr{E}$. Pour toute distance admissible d, on note $h_{\mathcal{C}}^{d}$ la fonction suivante :

$$\begin{array}{rccc} h_{\mathcal{C}}^{d}: & \mathscr{E} & \to & \mathscr{P}(\mathcal{C}) \\ & \mathbf{x} & \mapsto & \{\mathbf{c} \in \mathcal{C} : \Delta(\mathbf{x}, \mathbf{c}) \leqslant d\} \end{array} \tag{6.8}$$

où $\mathscr{P}(\mathcal{C})$ est l'ensemble des parties de $\mathcal{C}$. Dans ce chapitre et le suivant, nous faisons l'hypothèse de l'existence d'un décodeur que nous qualifions d'*idéal* qui calcule efficacement la fonction $h_{\mathcal{C}}^{d}$. Plus précisément, sa complexité est supposée de l'ordre de la taille de la liste retournée ; c'est-à-dire de l'ordre de $O(\max(1, E_1))$. Bien sûr cette hypothèse est purement théorique. Nous verrons dans le chapitre 9 comment remédier à cette lacune.

Par abus de notation, nous indexons les adresses de la table de hachage avec les mots du code $\mathcal{C}$. Si $\mathscr{T}$ est notre table de hachage, alors $\{\mathscr{T}[\mathbf{c}]\}_{\mathbf{c}\in\mathcal{C}}$ est l'ensemble des T cellules de la table de hachage.

Finalement, nous instancions l'algorithme 6.1.1 avec la famille de fonctions $\left\{h_{\mathcal{C}}^{d}\right\}_{\mathcal{C}}$ où $\mathcal{C}$ est n'importe quelle liste de T mots de $\mathscr{E}$ (avec répétitions) ; on dira que $\mathcal{C}$ est un code aléatoire de taille T. L'algorithme ainsi obtenu est donné par le pseudo-code 6.2.1.

Algorithme 6.2.1 : Méthode des codes pour la recherche de presque-collisions

Entrées : une liste $\mathscr{L} \subseteq \mathscr{E}$ de taille L ;
une distance admissible w ;
Paramètres : un nombre d'itérations s ;
un entier T ;
une distance de décodage d ;
Sortie : $\mathscr{L}^* \subseteq \left\{(\mathbf{x}, \mathbf{y}) \in \mathscr{L}^2 \ : \ \Delta(\mathbf{x}, \mathbf{y}) \leqslant w\right\}$.

```
1  L* ← ∅ ;
2  répéter s fois
3  |  C ← choisir T mots uniformément dans E ;
4  |  initialiser une table de hachage chainée T de taille T ;
5  |  pour tout x ∈ L faire
6  |  |  pour tout c ∈ h_C^d(x) faire
7  |  |  |  pour tout y ∈ T[c] faire
8  |  |  |  |  si Δ(x, y) ⩽ w alors
9  |  |  |  |  |  ajouter (x, y) dans L* ;
10 |  |  |  |  finSi
11 |  |  |  finPour
12 |  |  |  ajouter x dans la liste T[c] ;
13 |  |  finPour
14 |  finPour
15 finRépéter
16 retourner L* ;
```

Pour étudier nos fonctions de hachage floues, nous utiliserons souvent les notations suivantes :

Notation 6.2.1. *Nous notons $\mathcal{B}(\mathbf{c}, r)$ la boule fermée sur $\mathscr{E}$ de centre $\mathbf{c}$ et de rayon ρ :*

$$\mathcal{B}(\mathbf{c}, r) := \{\mathbf{x} \in \mathscr{E} : \Delta(\mathbf{x}, \mathbf{c}) \leqslant r\} \tag{6.9}$$

Nous posons alors $\mathcal{P}(\leqslant d)$ et $\mathcal{P}(\leqslant d \mid w)$ les deux probabilités suivantes :

$$\mathcal{P}(\leqslant d) := \mathbb{P}(\mathbf{u} \in \mathcal{B}(\mathbf{x}, d)) \tag{6.10}$$

$$\mathcal{P}(\leqslant d \mid w) := \mathbb{P}(\mathbf{u} \in \mathcal{B}(\mathbf{x}, d) \cap \mathcal{B}(\mathbf{y}, d)) \tag{6.11}$$

où $\mathbf{u}$ est choisi uniformément dans $\mathscr{E}$ et $\mathbf{x}, \mathbf{y} \in \mathscr{E}$ sont deux éléments à distance w.

Hypothèse 6.2.2. *Dans toute la suite, il est supposé que l'espace ambiant $\mathscr{E}$ est tel que $\mathcal{P}(\leqslant d)$ ne dépend pas de la valeur de $\mathbf{x}$ et $\mathcal{P}(\leqslant d \mid w)$ ne dépend que de la distance entre $\mathbf{x}$ et $\mathbf{y}$ mais pas de leurs valeurs.*

Nous pouvons alors exprimer la complexité de notre méthode en fonction de $\mathcal{P}(\leqslant d)$ et $\mathcal{P}(\leqslant d \mid w)$:

Théorème 6.2.3. *Supposons que pour tout code aléatoire $\mathcal{C}$ de taille T et pour toute distance admissible d, un décodeur en liste idéal calcule la fonction $h_{\mathcal{C}}^{d}$ en un temps de l'ordre de $O(\max(1, T \cdot \mathcal{P}(\leqslant d)))$. Ainsi, l'algorithme 6.2.1 peut résoudre le problème des presque-collisions avec une complexité mémoire de l'ordre de :*

$$S_{\text{Codes}}^{\text{Aléa}}(d) = O(L) \tag{6.12}$$

et une complexité en temps de l'ordre de :

$$T_{\text{Codes}}^{\text{Aléa}}(d) = \tilde{O}\left(\frac{L \cdot \mathcal{P}(\leqslant d) + (L \cdot \mathcal{P}(\leqslant d))^2}{\mathcal{P}(\leqslant d \mid w)}\right) \tag{6.13}$$

Démonstration du théorème 6.2.3.

Remarquons tout d'abord que pour un code $\mathcal{C}$ constitué de T mots tirés uniformément dans $\mathscr{E}$, nous avons :

$$\begin{aligned} E_1 &= T \cdot \mathbb{P}(\mathbf{c} \in \mathcal{B}(\mathbf{x}, d)) \\ &= T \cdot \mathcal{P}(\leqslant d) \end{aligned}$$

$$\begin{aligned} E_2 &= T \cdot \mathbb{P}(\mathbf{c} \in \mathcal{B}(\mathbf{x}, d) \text{ et } \mathbf{c} \in \mathcal{B}(\mathbf{y}, d)) \\ &= T \cdot \mathbb{P}(\mathbf{c} \in \mathcal{B}(\mathbf{x}, d)) \cdot \mathbb{P}(\mathbf{c} \in \mathcal{B}(\mathbf{y}, d)) \\ &= T \cdot \mathcal{P}(\leqslant d)^2 \end{aligned}$$

où $\mathbf{c}$ est un mot du code $\mathcal{C}$ et $\mathbf{x}$ et $\mathbf{y}$ sont choisis uniformément dans $\mathscr{E}$.

De plus, la probabilité de collision P^* est la probabilité qu'il existe au moins un mot de code dans l'intersection des boules $\mathcal{B}(\mathbf{x}, d)$ et $\mathcal{B}(\mathbf{y}, d)$ où $\mathbf{x}$ et $\mathbf{y}$ sont des vecteurs de $\mathscr{E}$ tels que $\Delta(\mathbf{x}, \mathbf{y}) = w$. La figure 6.1 représente la situation. En utilisant l'indépendance des mots de codes entre eux, nous avons :

$$\begin{aligned} P^* &= \Omega\left(\mathbb{P}\left(\bigcup_{\mathbf{c} \in \mathcal{C}} \mathbf{c} \in \mathcal{B}(\mathbf{x}, d) \cap \mathcal{B}(\mathbf{y}, d)\right)\right) \\ &= \Omega\left(1 - \mathbb{P}\left(\bigcap_{\mathbf{c} \in \mathcal{C}} \mathbf{c} \notin \mathcal{B}(\mathbf{x}, d) \cap \mathcal{B}(\mathbf{y}, d)\right)\right) \\ &= \Omega\left(1 - \prod_{\mathbf{c} \in \mathcal{C}} \mathbb{P}(\mathbf{c} \notin \mathcal{B}(\mathbf{x}, d) \cap \mathcal{B}(\mathbf{y}, d))\right) \\ &= \Omega\left(1 - (1 - \mathcal{P}(\leqslant d \mid w))^T\right) \end{aligned}$$

Un développement de Taylor nous donne alors :

$$P^* = \Omega\left(\min\left(1, T \cdot \mathcal{P}\left(\leqslant d \mid w\right)\right)\right) \tag{6.14}$$

Si nous choisissons $T = O\left(\frac{1}{\mathcal{P}(\leqslant d)}\right)$ alors la table de hachage contient $O(L)$ éléments ; ce qui nous donne la complexité spatiale. Nous avons aussi $T \cdot \mathcal{P}\left(\leqslant d \mid w\right) \leqslant T \cdot \mathcal{P}\left(\leqslant d\right) = 1$ et donc :

$$P^* = \Omega\left(T \cdot \mathcal{P}\left(\leqslant d \mid w\right)\right) \tag{6.15}$$

Finalement, nous terminons la preuve en appliquant le lemme 6.1.4. □

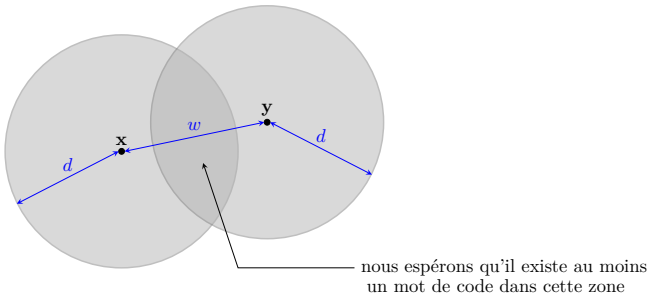


Figure 6.1 – Représentation schématique de l'intersection de boules qui nous intéresse.

6.2.2 À propos du nombre de couples à tester

Dans cette sous-section, nous montrons un lemme important. Celui-ci affirme qu'il existe une distance de décodage pour laquelle le nombre de couples dont la proximité doit être testée dans l'algorithme 6.2.1 est de l'ordre du nombre de couples proches que nous recherchons. Cette distance correspond en fait au rayon d'une sphère de décodage sur laquelle les mots ont une distance typique $\simeq w$.

Lemme 6.2.4. *Soit une distance admissible w que l'on suppose $\leqslant \mathbb{E}\left(\Delta\left(\mathbf{u}, \mathbf{v}\right)\right)$ pour $\mathbf{u}$ et $\mathbf{v}$ choisis uniformément dans $\mathscr{C}$. On note alors :*

$$d_w := \inf\left\{d \text{ admissible } : \ \mathbb{E}\left(\Delta\left(\mathbf{x}, \mathbf{y}\right)\right) \geqslant w\right\} \tag{6.16}$$

où $\mathbf{x}$ et $\mathbf{y}$ sont choisis uniformément dans une même boule de rayon d.

Il est supposé que :

- (hypothèse de densité des distances relatives admissibles) *pour tout $a < b \in [0, 1]$, il existe n_0 tel que pour tout $n > n_0$, il existe une distance admissible d telle que $\frac{d}{d_{\max}} \in [a, b]$ avec $d_{\max}$ la distance admissible maximale ;*
- (hypothèses de concentration) *pour toute distance admissible d et pour $\mathbf{x}$ et $\mathbf{y}$ tirés uniformément dans une boule de rayon d, il existe $\varepsilon(n) = o(1)$ tel que :*
  - *(1)* $\mathbb{P}\left(\Delta\left(\mathbf{x}, \mathbf{y}\right) \leqslant w\right) = \Theta\left(\mathbb{P}\left(\Delta\left(\mathbf{x}, \mathbf{y}\right) \in [w - \varepsilon(n), w]\right)\right)$;*
  - *(2) si* $\mathbb{E}\left(\Delta\left(\mathbf{x}, \mathbf{y}\right)\right) = w$ *alors* $\mathbb{P}\left(\Delta\left(\mathbf{x}, \mathbf{y}\right) \in [w - \varepsilon(n), w]\right) = \Omega\left(1\right)$.*

Ainsi, lorsque la dimension n de $\mathscr{C}$ tend vers l'infini, nous avons :

$$L^2 \cdot \mathcal{P}\left(\leqslant w\right) = \Omega\left(\frac{\left(L \cdot \mathcal{P}\left(\leqslant d_w\right)\right)^2}{\mathcal{P}\left(\leqslant d_w \mid w\right)}\right) \tag{6.17}$$

Démonstration du lemme 6.2.4.

Nous commençons par montrer que la distance d_w du lemme est bien définie. Soient $\mathbf{x}$ et $\mathbf{y}$ tirés uniformément dans une boule de rayon d. Nous remarquons que lorsque d est la distance admissible maximale $d_{\max}$:

$$\mathbb{E}\left(\Delta\left(\mathbf{x},\mathbf{y}\right)\right) \geqslant w$$

Donc l'ensemble $\{d \text{ admissible } : \mathbb{E}\left(\Delta\left(\mathbf{x},\mathbf{y}\right)\right) \geqslant w\}$ n'est pas vide et d_w est bien définie.
D'autre part, lorsque $d = d_{\min} := \inf\{d \text{ admissible } : \mathcal{P}\left(\leqslant d \mid w\right) > 0\}$, nous avons :

$$\mathbb{E}\left(\Delta\left(\mathbf{x},\mathbf{y}\right)\right) \quad \leqslant \quad w$$

Ainsi, d'après l'hypothèse de densité des distances relatives, lorsque n tend vers l'infini et $d = d_w$ nous avons :

$$\mathbb{E}\left(\Delta\left(\mathbf{x},\mathbf{y}\right)\right) = w + o(1)$$

Et donc d'après la seconde hypothèse de concentration, nous avons :

$$\mathbb{P}\left(\Delta\left(\mathbf{x},\mathbf{y}\right) \in [w-\varepsilon(n), w]\right) = \Omega\left(1\right)$$

Nous pouvons alors réécrire cette égalité :

$$\mathbb{P}\left(\Delta\left(\mathbf{x},\mathbf{y}\right) \in [w-\varepsilon(n), w] \;\middle|\; (\mathbf{x},\mathbf{y}) \in \mathcal{B}\left(\mathbf{c}, d_w\right)^2\right) = \Omega\left(1\right)$$

où cette fois, $\mathbf{x}$ et $\mathbf{y}$ sont choisis uniformément dans $\mathscr{E}$ et où $\mathbf{c} \in \mathscr{E}$.
D'autre part, pour tout $\mathbf{a}$, $\mathbf{b}$ et $\mathbf{c} \in \mathscr{E}$ tels que $\Delta\left(\mathbf{a},\mathbf{b}\right) = w$, nous avons avec l'hypothèse 6.2.2 :

$$\begin{array}{rcl} \mathcal{P}\left(\leqslant d_w \mid w\right) & := & \mathbb{P}\left(\mathbf{u} \in \mathcal{B}\left(\mathbf{a}, d_w\right) \cap \mathcal{B}\left(\mathbf{b}, d_w\right)\right) \\ & = & \mathbb{P}\left((\mathbf{a},\mathbf{b}) \in \mathcal{B}\left(\mathbf{u}, d_w\right)^2\right) \\ & = & \mathbb{P}\left((\mathbf{x},\mathbf{y}) \in \mathcal{B}\left(\mathbf{c}, d_w\right)^2 \mid \Delta\left(\mathbf{x},\mathbf{y}\right) = w\right) \end{array}$$

où $\mathbf{u}$, $\mathbf{x}$ et $\mathbf{y}$ sont tirés uniformément dans $\mathscr{E}$.

Finalement, d'après la première hypothèse de concentration, nous avons :

$$\begin{array}{rl} & \dfrac{\mathcal{P}\left(\leqslant d_w \mid w\right) \cdot \mathcal{P}\left(\leqslant w\right)}{\mathcal{P}\left(\leqslant d_w\right)^2} \\ = & \dfrac{\mathbb{P}\left((\mathbf{x},\mathbf{y}) \in \mathcal{B}\left(\mathbf{c}, d_w\right)^2 \mid \Delta\left(\mathbf{x},\mathbf{y}\right) = w\right) \cdot \mathbb{P}\left(\Delta\left(\mathbf{x},\mathbf{y}\right) \leqslant w\right)}{\mathbb{P}\left(\mathbf{x} \in \mathcal{B}\left(\mathbf{c}, d_w\right)\right) \cdot \mathbb{P}\left(\mathbf{y} \in \mathcal{B}\left(\mathbf{c}, d_w\right)\right)} \\ = & \Omega\left(\dfrac{\mathbb{P}\left((\mathbf{x},\mathbf{y}) \in \mathcal{B}\left(\mathbf{c}, d_w\right)^2 \mid \Delta\left(\mathbf{x},\mathbf{y}\right) \in [w-\varepsilon(n), w]\right) \cdot \mathbb{P}\left(\Delta\left(\mathbf{x},\mathbf{y}\right) \in [w-\varepsilon(n), w]\right)}{\mathbb{P}\left((\mathbf{x},\mathbf{y}) \in \mathcal{B}\left(\mathbf{c}, d_w\right)^2\right)}\right) \\ = & \Omega\left(\dfrac{\mathbb{P}\left(\Delta\left(\mathbf{x},\mathbf{y}\right) \in [w-\varepsilon(n), w] \text{ et } (\mathbf{x},\mathbf{y}) \in \mathcal{B}\left(\mathbf{c}, d_w\right)^2\right)}{\mathbb{P}\left((\mathbf{x},\mathbf{y}) \in \mathcal{B}\left(\mathbf{c}, d_w\right)^2\right)}\right) \\ = & \Omega\left(\mathbb{P}\left(\Delta\left(\mathbf{x},\mathbf{y}\right) \in [w-\varepsilon(n), w] \;\middle|\; (\mathbf{x},\mathbf{y}) \in \mathcal{B}\left(\mathbf{c}, d_w\right)^2\right)\right) \\ = & \Omega\left(1\right) \end{array}$$

□

Nous devons aussi regarder ce qu'il se passe lorsque $w > \mathbb{E}\left(\Delta\left(\mathbf{u},\mathbf{v}\right)\right)$ pour $\mathbf{u}$ et $\mathbf{v}$ choisis uniformément dans $\mathscr{E}$. En fait, ces cas sont triviaux car le nombre de couples proches que nous recherchons est alors de l'ordre de $O\left(L^2\right)$. Or, nous avons :

$$\frac{\left(L \cdot \mathcal{P}\left(\leqslant d_{\max}\right)\right)^2}{\mathcal{P}\left(\leqslant d_{\max} \mid w\right)} = O\left(L^2 \cdot \mathcal{P}\left(\leqslant w\right)\right) = O\left(L^2\right) \tag{6.18}$$

En d'autres termes, lorsque $w > \mathbb{E}\left(\Delta\left(\mathbf{u},\mathbf{v}\right)\right)$, notre méthode est asymptotiquement équivalente à la recherche exhaustive. Nous ne nous attarderons donc pas plus longuement sur ces cas particuliers.

6.2.3 Optimisation de la distance de décodage

Dans l'algorithme 6.2.1 nous devons choisir judicieusement le rayon de décodage d des décodeurs idéaux. Pour cela, nous commençons par généraliser la notion de la distance de Gilbert-Varshamov :

Définition 6.2.5. *La distance de Gilbert-Varshamov associée à une taille de code S sur $\mathscr{E}$ est :*

$$d^-_{GV}(S) := \begin{cases} \sup\{d \text{ admissible} \ : \ S \cdot \mathcal{P}(\leqslant d) \leqslant 1\} & \text{si } \mathcal{P}(\leqslant 0) > \frac{1}{S} \\ 0 & \text{sinon} \end{cases} \tag{6.19}$$

Nous pouvons alors énoncer le théorème suivant :

Théorème 6.2.6. *Supposons que pour tout code aléatoire $\mathcal{C} \subseteq \mathscr{E}$ de taille T et toute distance admissible d, il existe un décodeur en liste idéal qui calcule la fonction $h^d_{\mathcal{C}}$ en un temps de l'ordre de $O(\max(1, T \cdot \mathcal{P}(\leqslant d)))$. Ainsi, sous les hypothèses du lemme 6.2.4, l'algorithme 6.2.1 peut résoudre le problème des presque-collisions en un temps de l'ordre de :*

$$T^{\text{Aléa}}_{\text{Codes}} = \begin{cases} \tilde{O}\left(\dfrac{L \cdot \mathcal{P}(\leqslant d^-_{GV}(L))}{\mathcal{P}(\leqslant d^-_{GV}(L) \mid w)}\right) & \text{si } d^-_{GV}(L) \geqslant d_w \\ \tilde{O}\big(L^2 \mathcal{P}(\leqslant w)\big) & \text{sinon} \end{cases} \tag{6.20}$$

Démonstration du théorème 6.2.6.

Lorsque la distance de décodage d est supérieure à la distance de Gilbert-Varshamov $d^-_{\text{GV}}(L)$, la complexité de notre méthode est dominée par le terme $\frac{(L \cdot \mathcal{P}(\leqslant d))^2}{\mathcal{P}(\leqslant d \mid w)}$. Nous choisissons alors $d = \max\left(d^-_{\text{GV}}(L), d_w\right)$ puis nous appliquons le théorème 6.2.3. Nous terminons la preuve en remarquant que, d'après le lemme 6.2.4, on a $\frac{(L \cdot \mathcal{P}(\leqslant d_w))^2}{\mathcal{P}(\leqslant d_w \mid w)} = O\big(L^2 \mathcal{P}(\leqslant w)\big)$.

□

Remarque 6.2.1. La plupart du temps, nous aurons $L \cdot \mathcal{P}(\leqslant d^-_{\text{GV}}(L)) = O(1)$. Cependant, nous pouvons imaginer des espaces $\mathscr{E}$ pour lesquels nous avons $L \cdot \mathcal{P}(\leqslant 0) \leqslant 1$. Dans ce cas là, $d^-_{\text{GV}}(L) = 0$ et $L \cdot \mathcal{P}(\leqslant d^-_{\text{GV}}(L)) \neq O(1)$.

La complexité donnée par le théorème 6.2.6 semble être optimale pour n'importe quel espace métrique vérifiant les hypothèses du lemme 6.2.4. Nous pouvons le montrer dans les espaces métriques particuliers que nous étudierons dans le chapitre suivant à l'aide de simples études de fonctions. Nous allons toutefois donner les éléments qui nous permettent de faire cette conjecture sur des espaces métriques plus généraux.

Soient les deux fonctions suivantes :

$$\begin{array}{rccc} f: & [d_{\min}, d_{\max}] & \longrightarrow & \mathbb{R} \\ & d & \longmapsto & \frac{L \cdot \mathcal{P}(\leqslant d)}{\mathcal{P}(\leqslant d \mid w)} \end{array} \tag{6.21}$$

$$\begin{array}{rccc} g: & [d_{\min}, d_{\max}] & \longrightarrow & \mathbb{R} \\ & d & \longmapsto & \frac{(L \cdot \mathcal{P}(\leqslant d))^2}{\mathcal{P}(\leqslant d \mid w)} \end{array} \tag{6.22}$$

où $d_{\max}$ est la distance maximale dans $\mathscr{E}$ et $d_{\min}$ est la distance minimale telle que $\mathcal{P}(\leqslant d \mid w) > 0$ ($d_{\min}$ dépend donc de w). Pour toute distance de décodage $d \in [d_{\min}, d_{\max}]$, la complexité de notre méthode est $T^{\text{Aléa}}_{\text{Codes}}(d) = \tilde{O}(\max(f(d), g(d)))$ (cf. théorème 6.2.3). Or nous pouvons décrire assez précisément le comportement des fonctions f et g. Tout d'abord, nous remarquons que pour tout $d \geqslant d^-_{\text{GV}}(L)$, on a $g(d) \geqslant f(d)$ et pour tout $d \leqslant d^-_{\text{GV}}(L)$, on a $g(d) \leqslant f(d)$. De plus, lorsque d tend vers $d_{\max}$, on a $\mathcal{P}(\leqslant d)$ et $\mathcal{P}(\leqslant d \mid w)$ qui tendent tous les deux vers 1 donc :

$$\lim_{d \to W} f(d) = L \qquad \text{et} \qquad \lim_{d \to W} g(d) = L^2 \tag{6.23}$$

Nous pouvons ensuite vérifier que f est décroissante sur $[d_{\min}, d_{\max}]$. Pour cela, il suffit de montrer que pour toutes distances admissibles d' et d, si $d' > d$ alors le rapport du volume de $\mathcal{B}(\mathbf{x}, d')$ sur celui de $\mathcal{B}(\mathbf{x}, d)$ est plus petit que le rapport du volume de $\mathcal{B}(\mathbf{x}, d') \cap \mathcal{B}(\mathbf{y}, d')$ sur celui de $\mathcal{B}(\mathbf{x}, d) \cap \mathcal{B}(\mathbf{y}, d)$ où $\Delta(\mathbf{x}, \mathbf{y})$ est fixé. La figure 6.2 illustre de façon schématique ce phénomène.

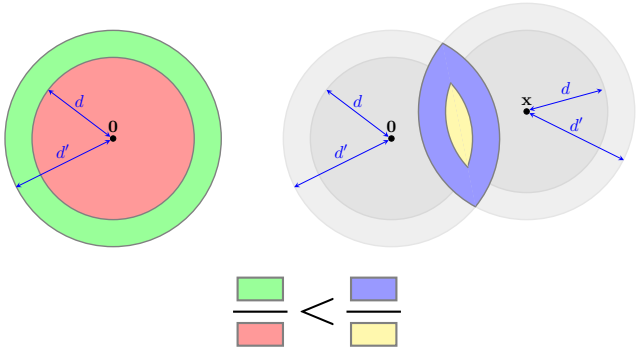


Figure 6.2 – Représentation schématique de la variation de f.

Les variations de g peuvent aussi être décrites. Tout d'abord, $L^2\mathcal{P}(\leqslant w)$ est un minimum de la fonction g sur $[d_{\min}, d_{\max}]$. En effet, dans le problème de presque-collisions, nous recherchons en moyenne $L^2\mathcal{P}(\leqslant w)$ couples proches. Donc un algorithme qui résout le problème des presque-collisions devra nécessairement vérifier au moins $L^2\mathcal{P}(\leqslant w)$ couples et donc $g(d) \geqslant L^2\mathcal{P}(\leqslant w)$. Or, la distance $d_w \in [d_{\min}, d_{\max}]$ donnée par le lemme 6.2.4, est telle que $g(d_w) = L^2\mathcal{P}(\leqslant w)$. De plus, au vu des variations de la fonction f, on peut supposer que g est décroissante sur $[d_{\min}, d_w]$ et croissante sur $[d_w, d_{\max}]$.

Finalement, la figure 6.3 résume les différentes situations possibles pour les variations de f et g. Nous avons alors :

$$\inf_{d\in[d_{\min},d_{\max}]}\left(T^{\text{Aléa}}_{\text{Codes}}(d)\right) := \tilde{O}\left(\inf_{d\in[d_{\min},d_{\max}]}\left(\max\left(f(d), g(d)\right)\right)\right) = \tilde{O}\left(g\left(\max\left(d^-_{\text{GV}}(L), d_w\right)\right)\right) \tag{6.24}$$

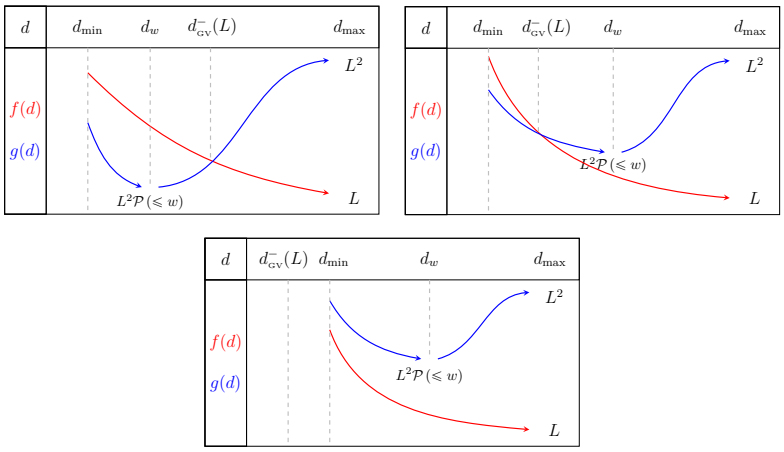


Figure 6.3 – Les différents cas possibles pour les variations de f et g.

6.3 Trouver les couples éloignés avec des anti-décodeurs

6.3.1 Anti-décodages en liste idéaux de codes aléatoires

Soient $\mathbf{u}$ et $\mathbf{v}$ tirés uniformément dans $\mathscr{E}$. Le problème de recherche de presque-collisions 6.1.2 n'a réellement de sens que lorsque $w < \mathbb{E}(\Delta(\mathbf{u}, \mathbf{v}))$. En effet, si $w \geqslant \mathbb{E}(\Delta(\mathbf{u}, \mathbf{v}))$ alors tous les couples de $\mathscr{L}^2$ sont typiquement proches et nous ne pouvons pas faire mieux qu'une recherche exhaustive. Toutefois, plutôt que de rechercher des couples à distance au plus w, nous pourrions être intéressés par la recherche des couples à distance *au moins* w. On définit alors le problème de recherche des lointaines-collisions :

Problème 6.3.1 (lointaines-collisions)**.** *Soit un espace métrique $(\mathscr{E}, \Delta)$ probabilisé avec la probabilité uniforme et soit une distance admissible w et un entier L. Étant donné une liste $\mathscr{L}$ de L éléments tirés uniformément dans $\mathscr{E}$, trouver les couples $(\mathbf{x}, \mathbf{y}) \in \mathscr{L}^2$ tels que $\Delta(\mathbf{x}, \mathbf{y}) \geqslant w$.*

Commençons par regarder ce qu'il se passe dans le cas où $\mathscr{E}$ est l'espace de Hamming binaire $\mathbb{F}_2^n$. Le problème des lointaines-collisions se réduit alors trivialement au problème des presque-collisions. En effet, pour trouver les couples à distance au moins w dans $\mathscr{L}^2$, il suffit de rechercher les couples à distance au plus $n - w$ dans $\mathscr{L} \times \overline{\mathscr{L}}$ où $\overline{\mathscr{L}} := \{\mathbf{x} + \mathbf{1} \in \mathbb{F}_2^n \ : \ \mathbf{x} \in \mathscr{L}\}$ (le mot $\mathbf{1}$ est le vecteur tout à 1 dans $\mathbb{F}_2^n$). Notre méthode de recherche de lointaines-collisions consiste alors à décoder les mots de $\mathscr{L}$ et $\overline{\mathscr{L}}$. Or décoder les éléments de $\overline{\mathscr{L}}$ consiste en fait à rechercher des mots de codes éloignés des mots de $\mathscr{L}$. Nous appelons *anti-décodeur*, un algorithme qui recherche des mots de codes éloignés d'un mot donné. Nous allons généraliser l'utilisation d'anti-décodeurs à d'autres espaces métriques $\mathscr{E}$ de dimension n.

Nous proposons de changer légèrement notre algorithme 6.2.1 pour traiter le problème des lointaines-collisions dans $\mathscr{E}$. Plus spécifiquement, à chaque itération, nous hachons les éléments de $\mathscr{L}$ deux fois : une première fois avec un décodeur en liste idéal d'un code aléatoire (cf. section 6.2) et une seconde fois avec un anti-décodeur en liste idéal du même code qui, au lieu de trouver les mots de codes proches, trouve tous les mots de codes éloignés. Plus formellement, pour tout code aléatoire $\mathcal{C}$ et tout couple de distances admissibles (d_1, d_2), on note $h_{\mathcal{C}}^{d_1}$ et $\overline{h}_{\mathcal{C}}^{d_2}$ les deux fonctions suivantes :

$$\begin{array}{rccc} h_{\mathcal{C}}^{d_1} : & \mathscr{E} & \to & \mathscr{P}(\mathcal{C}) \\ & \mathbf{x} & \mapsto & \{\mathbf{c} \in \mathcal{C} : \Delta(\mathbf{x}, \mathbf{c}) \leqslant d_1\} \end{array} \tag{6.25}$$

$$\begin{array}{rccc} \overline{h}_{\mathcal{C}}^{d_2} : & \mathscr{E} & \to & \mathscr{P}(\mathcal{C}) \\ & \mathbf{x} & \mapsto & \{\mathbf{c} \in \mathcal{C} : \Delta(\mathbf{x}, \mathbf{c}) \geqslant d_2\} \end{array} \tag{6.26}$$

où $\mathscr{P}(\mathcal{C})$ est l'ensemble des parties de $\mathcal{C}$. Nous rappelons que par abus de notation, nous indexons les adresses de la table de hachage avec les mots du code $\mathcal{C}$. De plus, nous faisons encore une fois l'hypothèse de l'existence d'un décodeur et d'un anti-décodeur que nous qualifions d'*idéaux* qui calculent efficacement les fonctions $h_{\mathcal{C}}^{d_1}$ et $\overline{h}_{\mathcal{C}}^{d_2}$. Plus précisément, leurs complexités sont supposées de l'ordre des tailles des listes retournées.

Notre méthode pour résoudre le problème des lointaines-collisions est finalement décrite

par l'algorithme 6.3.1.

Algorithme 6.3.1 : Recherche de lointaines-collisions

Entrées : une liste $\mathscr{L} \subseteq \mathscr{E}$ de taille L ;
une distance admissible w ;
Paramètres : un nombre d'itérations s ;
un entier T ;
deux distances de décodage d_1 et d_2 ;
Sortie : $\mathscr{L}^* = \left\{(\mathbf{x},\mathbf{y}) \in \mathscr{L}^2 \ : \ \Delta(\mathbf{x},\mathbf{y}) \geqslant w\right\}$.

```
1  L* ← ∅ ;
2  répéter s fois
3  |  C ← choisir T mots uniformément dans E ;
4  |  initialiser une table de hachage chainée T de taille T ;
5  |  pour tout x ∈ L faire
6  |  |  pour tout c ∈ h_C^{d1}(x) faire
7  |  |  |  ajouter x dans la liste T[c] ;
8  |  |  finPour
9  |  finPour
10 |  pour tout y ∈ L faire
11 |  |  pour tout c ∈ h̄_C^{d2}(y) faire
12 |  |  |  pour tout x ∈ T[c] faire
13 |  |  |  |  si Δ(x,y) ⩾ w alors
14 |  |  |  |  |  ajouter (x,y) dans L* ;
15 |  |  |  |  finSi
16 |  |  |  finPour
17 |  |  finPour
18 |  finPour
19 finRépéter
20 retourner L* ;
```

Remarque 6.3.1. Notons que pour rechercher des couples éloignés dans $\mathscr{L}_1 \times \mathscr{L}_2$ où $\mathscr{L}_1$ et $\mathscr{L}_2$ sont deux listes de vecteurs de $\mathscr{E}$, nous devons simplement remplacer les instructions **5** et **10** de l'algorithme 6.3.1 par :

```
5   |  pour tout x ∈ L1 faire
[...]
10  |  pour tout y ∈ L2 faire
```

Pour analyser l'algorithme 6.3.1, nous avons besoin de compléter nos notations :

Notation 6.3.2. *Nous notons $\overline{\mathcal{B}}(\mathbf{c},\rho)$ le complémentaire sur $\mathscr{E}$ de la boule ouverte de centre $\mathbf{c}$ et de rayon ρ :*

$$\overline{\mathcal{B}}(\mathbf{c},\rho) := \{\mathbf{x} \in \mathscr{E} \ : \ \Delta(\mathbf{c},\mathbf{x}) \geqslant \rho\} \tag{6.27}$$

Nous définissons alors les deux probabilités suivantes :

$$\mathcal{P}(\geqslant d) \ := \ \mathbb{P}\left(\mathbf{u} \in \overline{\mathcal{B}}(\mathbf{x},d)\right) \tag{6.28}$$

$$\mathcal{P}(\leqslant d_1 \ , \ \geqslant d_2 \mid w) \ := \ \mathbb{P}\left(\mathbf{u} \in \mathcal{B}(\mathbf{x},d_1) \cap \overline{\mathcal{B}}(\mathbf{y},d_2)\right) \tag{6.29}$$

où $\mathbf{u}$ est choisi uniformément dans $\mathscr{E}$ et $\mathbf{x},\mathbf{y} \in \mathscr{E}$ sont deux éléments à distance w.

Nous pouvons alors énoncer le théorème suivant pour la recherche de lointaines-collisions :

Théorème 6.3.3. *Nous supposons que pour tout code aléatoire $\mathcal{C}$ de taille T et pour tout couple de distances admissibles (d_1, d_2), il existe un décodeur et un anti-décodeur en liste idéaux qui calculent respectivement les fonctions $h_{\mathcal{C}}^{d_1}$ et $\overline{h}_{\mathcal{C}}^{d_2}$ en un temps de l'ordre de $O(\max(1, T \cdot \mathcal{P}(\leqslant d_1)))$ et $O(\max(1, T \cdot \mathcal{P}(\geqslant d_2)))$. Ainsi, l'algorithme 6.3.1 peut résoudre le problème des lointaines-collisions avec une complexité mémoire de l'ordre de :*

$$S_{\text{Codes}}^{\text{Aléa}}(d_1, d_2) = O(L) \tag{6.30}$$

et une complexité en temps de l'ordre de :

$$T_{\text{Codes}}^{\text{Aléa}}(d_1, d_2) = \tilde{O}\left(\frac{L \cdot \mathcal{P}(\leqslant d_1) + L \cdot \mathcal{P}(\geqslant d_2) + L^2 \cdot \mathcal{P}(\leqslant d_1) \cdot \mathcal{P}(\geqslant d_2)}{\mathcal{P}(\leqslant d_1\ ,\ \geqslant d_2 \mid w)}\right) \tag{6.31}$$

Démonstration du théorème 6.3.3.

La preuve suit essentiellement la même stratégie que pour le théorème 6.2.3. Dans l'équation (6.31), les quantités $L \cdot \mathcal{P}(\leqslant d_1)$ et $L \cdot \mathcal{P}(\geqslant d_2)$ sont respectivement les coûts pour décoder et anti-décoder les éléments de $\mathscr{L}$ lors d'une itération. Le nombre de comparaisons à effectuer pendant cette itération est $L^2 \cdot \mathcal{P}(\leqslant d_1) \cdot \mathcal{P}(\geqslant d_2)$. Et enfin, sa probabilité de succès est $\mathcal{P}(\leqslant d_1\ ,\ \geqslant d_2 \mid w)$.

La figure 6.4 représente la nouvelle situation. □

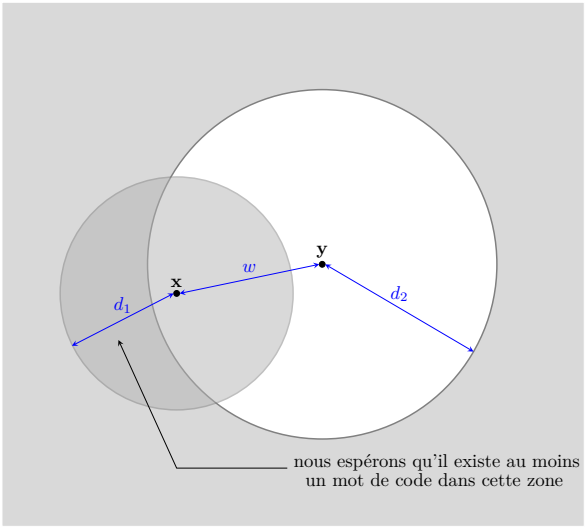


Figure 6.4 – Représentation schématique de l'intersection qui nous intéresse pour la recherche de lointaines-collisions.

6.3.2 À propos du nombre de couples à tester

On retrouve un lemme analogue au lemme 6.2.4 qui affirme que dans l'algorithme 6.3.1, nous pouvons toujours choisir des distances de décodage d_1 et d_2 tels que le nombre de couples qui apparaitront dans la table de hachage sera de l'ordre du nombre de couples éloignés que nous recherchons.

Lemme 6.3.4. *Soit une distance admissible w que l'on suppose $\geqslant \mathbb{E}(\Delta(\mathbf{u}, \mathbf{v}))$ pour $\mathbf{u}$ et $\mathbf{v}$ choisis uniformément dans $\mathscr{E}$. Pour toute distance admissible $d_2^w \geqslant w$, on note :*

$$d_1^w := \sup\{d_1 \text{ admissible } :\ \mathbb{E}(\Delta(\mathbf{x}, \mathbf{y})) \geqslant w\} \tag{6.32}$$

où $\mathbf{x}$ et $\mathbf{y}$ sont respectivement choisis uniformément dans $\mathcal{B}(\mathbf{c}, d_1)$ et $\overline{\mathcal{B}}(\mathbf{c}, d_2^w)$ avec $\mathbf{c} \in \mathscr{E}$.

Il est supposé que :

— (hypothèse de densité des distances relatives) *pour tout* $a < b \in [0,1]$*, il existe* n_0 *tel que pour tout* $n > n_0$*, il existe une distance admissible* d *telle que* $\frac{d}{d_{\max}} \in [a,b]$ *avec* $d_{\max}$ *la distance admissible maximale ;*

— (hypothèses de concentration) *pour toute distance admissible* d_1 *et pour* $\mathbf{x}$ *et* $\mathbf{y}$ *tirés respectivement uniformément dans* $\mathcal{B}(\mathbf{c}, d_1)$ *et* $\overline{\mathcal{B}}(\mathbf{c}, d_2^w)$ *avec* $\mathbf{c} \in \mathscr{C}$*, il existe* $\varepsilon(n) = o(1)$ *tel que :*

(1) $\mathbb{P}(\Delta(\mathbf{x},\mathbf{y}) \geqslant w) = \Theta(\mathbb{P}(\Delta(\mathbf{x},\mathbf{y}) \in [w, w+\varepsilon(n)]))$ *;*

(2) si $\mathbb{E}(\Delta(\mathbf{x},\mathbf{y})) = w$ *alors* $\mathbb{P}(\Delta(\mathbf{x},\mathbf{y}) \in [w, w+\varepsilon(n)]) = \Omega(1)$*.*

Ainsi, lorsque la dimension n *de* $\mathscr{C}$ *tend vers l'infini, nous avons :*

$$L^2 \cdot \mathcal{P}(\geqslant w) = \Omega\left(\frac{L^2 \cdot \mathcal{P}(\leqslant d_1^w) \cdot \mathcal{P}(\geqslant d_2^w)}{\mathcal{P}(\leqslant d_1^w \ , \ \geqslant d_2^w \mid w)}\right) \tag{6.33}$$

Démonstration du lemme 6.3.4.

La preuve est analogue à celle du lemme 6.2.4 avec toutefois quelques ajustements.

Soient $\mathbf{x}$ et $\mathbf{y}$ tirés respectivement uniformément dans $\mathcal{B}(\mathbf{c}, d_1)$ et $\overline{\mathcal{B}}(\mathbf{c}, d_2^w)$ avec $\mathbf{c} \in \mathscr{C}$. Pour montrer l'existence de d_1^w, nous remarquons simplement que lorsque $d_1 = d_1^{\min} := \inf\{d_1 \text{ admissible } : \mathcal{P}(\leqslant d_1^w \ , \ \geqslant d_2^w \mid w) > 0\}$, nous avons :

$$\mathbb{E}(\Delta(\mathbf{x},\mathbf{y})) \geqslant w$$

Donc l'ensemble $\{d_1 \text{ admissible } : \mathbb{E}(\Delta(\mathbf{x},\mathbf{y}) \geqslant w)\}$ n'est pas vide et d_1^w est bien définie.

D'autre part, lorsque $d_1 = d_{\max}$, nous avons :

$$\mathbb{E}(\Delta(\mathbf{x},\mathbf{y})) = \mathbb{E}(\Delta(\mathbf{u},\mathbf{v})) \leqslant w$$

Ainsi, d'après l'hypothèse de densité des distances relatives, lorsque n tend vers l'infini et $d_1 = d_1^w$ nous avons :

$$\mathbb{E}(\Delta(\mathbf{x},\mathbf{y})) = w + o(1)$$

Et donc d'après la seconde hypothèse de concentration, nous avons :

$$\mathbb{P}(\Delta(\mathbf{x},\mathbf{y}) \in [w, w+\varepsilon(n)]) = \Omega(1)$$

Nous pouvons alors réécrire cette égalité :

$$\mathbb{P}\left(\Delta(\mathbf{x},\mathbf{y}) \in [w, w+\varepsilon(n)] \;\middle|\; (\mathbf{x},\mathbf{y}) \in \mathcal{B}(\mathbf{c}, d_1^w) \times \overline{\mathcal{B}}(\mathbf{c}, d_2^w)\right) = \Omega(1)$$

où cette fois, $\mathbf{x}$ et $\mathbf{y}$ sont choisis uniformément dans $\mathscr{C}$ et où $\mathbf{c} \in \mathscr{C}$.

D'autre part, pour tout $\mathbf{a}$, $\mathbf{b}$ et $\mathbf{c} \in \mathscr{C}$ tels que $\Delta(\mathbf{a},\mathbf{b}) = w$, nous avons avec l'hypothèse 6.2.2 :

$$\begin{aligned}
\mathcal{P}(\leqslant d_1 \ , \ \geqslant d_2 \mid w) &:= \mathbb{P}\left(\mathbf{u} \in \mathcal{B}(\mathbf{a}, d_1) \cap \overline{\mathcal{B}}(\mathbf{b}, d_2)\right) \\
&= \mathbb{P}\left((\mathbf{a},\mathbf{b}) \in \mathcal{B}(\mathbf{u}, d_1) \times \mathcal{B}(\mathbf{u}, d_2)\right) \\
&= \mathbb{P}\left((\mathbf{x},\mathbf{y}) \in \mathcal{B}(\mathbf{c}, d_1) \times \overline{\mathcal{B}}(\mathbf{c}, d_2) \;\middle|\; \Delta(\mathbf{x},\mathbf{y}) = w\right)
\end{aligned}$$

où $\mathbf{u}$, $\mathbf{x}$ et $\mathbf{y}$ sont tirés uniformément dans $\mathscr{C}$.

Finalement, d'après la première hypothèse de concentration, nous avons :

$$\begin{aligned}
&\frac{\mathcal{P}\left(\leqslant d_1^w\ ,\ \geqslant d_2^w \mid w\right)\cdot\mathcal{P}\left(\geqslant w\right)}{\mathcal{P}\left(\leqslant d_1^w\right)\cdot\mathcal{P}\left(\geqslant d_2^w\right)}\\
=\ &\frac{\mathbb{P}\left((\mathbf{x},\mathbf{y})\in\mathcal{B}\left(\mathbf{c},d_1^w\right)\times\overline{\mathcal{B}}\left(\mathbf{c},d_2^w\right)\ \middle|\ \Delta\left(\mathbf{x},\mathbf{y}\right)=w\right)\cdot\mathbb{P}\left(\Delta\left(\mathbf{x},\mathbf{y}\right)\geqslant w\right)}{\mathbb{P}\left(\mathbf{x}\in\mathcal{B}\left(\mathbf{c},d_1^w\right)\right)\cdot\mathbb{P}\left(\mathbf{x}\in\overline{\mathcal{B}}\left(\mathbf{c},d_2^w\right)\right)}\\
=\ &\Omega\left(\frac{\mathbb{P}\left((\mathbf{x},\mathbf{y})\in\mathcal{B}\left(\mathbf{c},d_1^w\right)\times\overline{\mathcal{B}}\left(\mathbf{c},d_2^w\right)\ \middle|\ \Delta\left(\mathbf{x},\mathbf{y}\right)\in[w,w+\varepsilon(n)]\right)\cdot\mathbb{P}\left(\Delta\left(\mathbf{x},\mathbf{y}\right)=w+o(1)\right)}{\mathbb{P}\left((\mathbf{x},\mathbf{y})\in\mathcal{B}\left(\mathbf{c},d_1^w\right)\times\overline{\mathcal{B}}\left(\mathbf{c},d_2^w\right)\right)}\right)\\
=\ &\Omega\left(\frac{\mathbb{P}\left(\Delta\left(\mathbf{x},\mathbf{y}\right)\in[w,w+\varepsilon(n)]\text{ et }(\mathbf{x},\mathbf{y})\in\mathcal{B}\left(\mathbf{c},d_1^w\right)\times\overline{\mathcal{B}}\left(\mathbf{c},d_2^w\right)\right)}{\mathbb{P}\left((\mathbf{x},\mathbf{y})\in\mathcal{B}\left(\mathbf{c},d_1^w\right)\times\overline{\mathcal{B}}\left(\mathbf{c},d_2^w\right)\right)}\right)\\
=\ &\Omega\left(\mathbb{P}\left(\Delta\left(\mathbf{x},\mathbf{y}\right)\in[w,w+\varepsilon(n)]\ \middle|\ (\mathbf{x},\mathbf{y})\in\mathcal{B}\left(\mathbf{c},d_1^w\right)\times\overline{\mathcal{B}}\left(\mathbf{c},d_2^w\right)\right)\right)\\
=\ &\Omega\left(1\right)
\end{aligned}$$

□

Nous devons aussi regarder le problème des lointaines-collisions pour $w<\mathbb{E}\left(\Delta\left(\mathbf{u},\mathbf{v}\right)\right)$ pour $\mathbf{u}$ et $\mathbf{v}$ choisis uniformément dans $\mathscr{E}$. En fait, ce cas est trivial car le nombre de couples proches que nous recherchons est alors de l'ordre de $O\big(L^2\big)$. Ainsi, nous avons :

$$\frac{L^2\cdot\mathcal{P}\left(\leqslant d_{\max}\right)\cdot\mathcal{P}\left(\geqslant d_2^w\right)}{\mathcal{P}\left(\leqslant d_{\max}\ ,\ \geqslant d_2^w\mid w\right)}=O\big(L^2\cdot\mathcal{P}\left(\geqslant w\right)\big)=O\big(L^2\big)\tag{6.34}$$

En d'autres termes, lorsque $w<\mathbb{E}\left(\Delta\left(\mathbf{u},\mathbf{v}\right)\right)$, notre méthode est asymptotiquement équivalente à la recherche exhaustive pour la recherche de lointaines-collisions. Nous ne nous attarderons donc pas plus longuement sur ces cas particuliers.

6.3.3 Optimisation des distances de décodage

Il nous reste encore à choisir judicieusement les distances de décodages d_1 et d_2 dans l'algorithme 6.3.1. Pour cela, nous avons besoin d'adapter notre notion précédente de distance de Gilbert-Varshamov pour les anti-décodeurs :

Définition 6.3.5. *La distance supérieure de Gilbert-Varshamov associée à une taille de code S sur $\mathscr{E}$ est :*

$$d_{GV}^{+}(S):=\begin{cases}\inf\left\{d\ \textit{admissible}\ :\ S\cdot\mathcal{P}\left(\geqslant d\right)\leqslant 1\right\} & \textit{si } \mathcal{P}\left(\geqslant d_{\max}\right)>\frac{1}{S}\\ d_{\max} & \textit{sinon}\end{cases}\tag{6.35}$$

où $d_{\max}$ est la distance admissible maximale.

Nous avons alors un théorème analogue au théorème 6.2.6 :

Théorème 6.3.6. *Nous supposons que pour tout code aléatoire $\mathcal{C}$ de taille T et tout couple de distances admissibles (d_1,d_2), il existe un décodeur et un anti-décodeur en liste idéaux qui calculent respectivement les fonctions $h_{\mathcal{C}}^{d_1}$ et $\overline{h}_{\mathcal{C}}^{d_2}$ en un temps de l'ordre de $O(\max\left(1,T\cdot\mathcal{P}\left(\leqslant d_1\right)\right))$ et $O(\max\left(1,T\cdot\mathcal{P}\left(\geqslant d_2\right)\right))$. Ainsi, sous les hypothèses du lemme 6.3.4 avec $d_2^w:=d_{GV}^{+}(L)$, l'algorithme 6.3.1 peut résoudre le problème des lointaines-collisions en un temps de l'ordre de :*

$$T_{\text{Codes}}^{\text{Aléa}}=\begin{cases}\tilde{O}\left(\dfrac{L^2\cdot\mathcal{P}\left(\leqslant d_{GV}^{-}(L)\right)\cdot\mathcal{P}\left(\geqslant d_{GV}^{+}(L)\right)}{\mathcal{P}\left(\leqslant d_{GV}^{-}(L)\ ,\ \geqslant d_{GV}^{+}(L)\mid w\right)}\right) & \textit{si}\quad d_2^w<w\\ & \quad\textit{ou}\ \left[d_2^w\geqslant w\ \textit{et}\ d_1^w\leqslant d_{GV}^{-}(L)\right]\\ \tilde{O}\big(L^2\mathcal{P}\left(\geqslant w\right)\big) & \textit{sinon}\end{cases}\tag{6.36}$$

Démonstration du théorème 6.3.6.

Pour toutes distances de décodages $d_1 \geqslant d^-_{\mathrm{GV}}(L)$ et $d_2 \leqslant d^+_{\mathrm{GV}}(L)$, la complexité de notre méthode est dominée par le terme $\frac{L^2 \cdot \mathcal{P}(\leqslant d_1) \cdot \mathcal{P}(\geqslant d_2)}{\mathcal{P}(\leqslant d_1 \ , \ \geqslant d_2 \mid w)}$. Nous choisissons $d_2 := d^+_{\mathrm{GV}}(L)$. Si $d_2 \geqslant w$ alors nous pouvons appliquer le lemme 6.3.4 avec $d_2^w = d_2$ et obtenir ainsi d_1^w ; dans ce cas là, nous prenons $d_1 := \max\left(d^-_{\mathrm{GV}}(L), d_1^w\right)$. En revanche, si $d_2 < w$ alors nous choisissons $d_1 = d^-_{\mathrm{GV}}(L)$. □

Remarque 6.3.2. Si $L \cdot \mathcal{P}(\leqslant 0) \leqslant 1$ alors nous avons $L \cdot \mathcal{P}\left(\leqslant d^-_{\mathrm{GV}}(L)\right) = O(1)$, sinon, $L \cdot \mathcal{P}\left(\leqslant d^-_{\mathrm{GV}}(L)\right) = L \cdot \mathcal{P}(\leqslant 0)$. De la même façon, si $L \cdot \mathcal{P}(\geqslant d_{\max}) \leqslant 1$ alors nous avons $L \cdot \mathcal{P}\left(\geqslant d^+_{\mathrm{GV}}(L)\right) = O(1)$, sinon, $L \cdot \mathcal{P}\left(\geqslant d^+_{\mathrm{GV}}(L)\right) = L \cdot \mathcal{P}(\geqslant d_{\max})$.

Remarque 6.3.3. Pour $\mathbf{u}$ et $\mathbf{v}$ choisis uniformément dans $\mathscr{C}$, nous avons vu dans la section précédente que lorsque $w \geqslant \mathbb{E}\left(\Delta(\mathbf{u}, \mathbf{v})\right)$, notre méthode pour rechercher les presque-collisions est équivalente à une recherche exhaustive. De la même façon, lorsque $w \leqslant \mathbb{E}\left(\Delta(\mathbf{u}, \mathbf{v})\right)$, notre méthode pour rechercher les lointaines-collisions est aussi équivalente à une recherche exhaustive.

Chapitre 7

Presque et lointaines collisions dans différents espaces métriques

Dans ce chapitre, nous appliquons nos résultats du chapitre 6 à différents espaces métriques. Dans la section 7.1, nous nous concentrons particulièrement sur le cas de la sphère euclidienne. Cette instance du problème des presque-collisions trouve de nombreuses applications ; notamment dans le domaine de la cryptologie fondée sur les réseaux euclidiens. Dans les sections suivantes, nous nous intéressons aux problèmes des presque et lointaines collisions dans les espaces de Hamming binaires et non-binaires.

7.1 La sphère euclidienne

7.1.1 La résolution du problème SVP dans les réseaux euclidiens

Un problème important dans le domaine des réseaux euclidiens est le problème SVP (*Shortest Vector Problem*). Nous notons $\mathcal{L}(B)$ le réseau euclidien de dimension n et de base $B := \{\mathbf{b}_1, \cdots, \mathbf{b}_n\} \subseteq \mathbb{R}^n$ et rappelons sa définition :

$$\mathcal{L}(B) := \left\{ \sum_{i=1}^{n} \alpha_i \mathbf{b}_i \ : \ \forall i \in [\![1, n]\!], \alpha_i \in \mathbb{Z} \right\} \tag{7.1}$$

Le problème SVP se formalise alors ainsi :

Problème 7.1.1 (SVP)**.** *Étant donnée une base B de n vecteurs de $\mathbb{R}^n$, trouver un vecteur $\mathbf{x} \in \mathcal{L}(B)$ tel que $||\mathbf{x}|| = \min_{\mathbf{u} \in \mathcal{L}(B) \setminus \{\mathbf{0}\}} ||\mathbf{u}||$ où $||\cdot||$ est la norme ℓ_2 sur $\mathbb{R}^n$.*

La difficulté de ce problème croît avec la dimension n du réseau. Nous nous intéressons alors particulièrement aux cas où n est élevé ; et même au cas asymptotique où n tend vers l'infini.

Les techniques dites de "tamisage" (en anglais *sieving*) ont rencontré un fort succès ces dernières années [BGJ15, BL16, Laa15, LdW15, BDGL16]. Cette approche est la plus efficace connue à ce jour pour résoudre le problème SVP. Le principe est simple : étant donnée une liste $\mathscr{L}_0 \subseteq \mathcal{L}(B)$, des listes $\mathscr{L}_1, \cdots, \mathscr{L}_s \subseteq \mathcal{L}(B)$ sont produites récursivement de la façon suivante :

$$\forall i \in [\![0, s-1]\!], \mathscr{L}_{i+1} := \{\mathbf{x} \pm \mathbf{y} \ : \ \mathbf{x} \neq \mathbf{y} \in \mathscr{L}_i \text{ et } ||\mathbf{x} \pm \mathbf{y}|| \leqslant \max(||\mathbf{x}||, ||\mathbf{y}||)\} \tag{7.2}$$

Le nombre de listes s est alors choisi tel que la probabilité d'avoir un vecteur de $\mathcal{L}(B)$ de norme minimale dans $\mathscr{L}_s$ soit de l'ordre de $\Omega(1)$.

Des heuristiques montrent que l'analyse de telles méthodes sont plus proches de la réalité lorsque nous supposons que pour chaque $i \in [\![0, s]\!]$, les éléments de la liste $\mathscr{L}_i$ vivent sur une sphère de $\mathbb{R}^n$. Pour chacune de ces sphères, on note r_i son rayon. Sous cette hypothèse, les couples que nous conservons d'une liste à l'autre sont ceux formant un angle d'au plus $\frac{\pi}{3}$ ou au moins $\frac{2\pi}{3}$; c'est-à-dire :

$$\mathscr{L}_{i+1} := \left\{ \mathbf{x} \pm \mathbf{y} \ : \ \mathbf{x} \neq \mathbf{y} \in \mathscr{L}_i \ \text{ et } \ \arccos\left(\frac{|\langle \mathbf{x}, \mathbf{y} \rangle|}{r_i^2} \right) \leqslant \frac{\pi}{3} \right\} \tag{7.3}$$

où $\langle \cdot, \cdot \rangle$ est le produit scalaire canonique sur $\mathbb{R}^n$. La liste $\mathscr{L}_{i+1}$ est calculée en effectuant une recherche de presque-collisions dans la liste $\mathscr{L}_i$. Ces recherches récursives de presque-collisions sont les étapes essentielles dans la technique de tamisage.

Pour simplifier les calculs, on normalise chaque liste $\mathscr{L}_i$ par le rayon r_i de la sphère sur laquelle ses éléments vivent. Ainsi, nous considérerons que pour tout $i \in [\![0, s]\!]$, les éléments de $\mathscr{L}_i$ sont des vecteurs de la sphère euclidienne unitaire $\mathcal{S}_1^n$.

Dans [NV08], il est montré que si $\mathscr{L}_0$ est de taille de l'ordre de $\left(\frac{4}{3}\right)^{\frac{n}{2}+o(n)}$, alors pour tout $i \in [\![0, s]\!]$, nous aurons $\#\mathscr{L}_i = \left(\frac{4}{3}\right)^{\frac{n}{2}+o(n)}$. De plus, le nombre de listes s nécessaires en moyenne pour résoudre le problème SVP est alors polynomial en n. Ainsi, la complexité asymptotique de la résolution du problème SVP par tamisage est dominée par la recherche de presque-collisions sur une liste $\mathscr{L} \subseteq \mathcal{S}_1^n$ de taille $\left(\frac{4}{3}\right)^{\frac{n}{2}+o(n)}$. Finalement, effectuer les recherches de presque-collisions de façon exhaustive entraine une complexité globale de l'ordre de $2^{0.415n+o(n)}$. Dans [BDGL16], il est proposé une méthode de recherche de presque-collisions qui permet de résoudre le problème SVP par tamisage avec une complexité en temps et en mémoire de l'ordre de $2^{0.292n+o(n)}$.

Notre méthode ne permettra pas d'améliorer la complexité en temps de la résolution du problème SVP car la taille des listes et la distance des couples proches recherchés correspondent à une instance du problème des presque-collisions où nous n'améliorons pas l'exposant asymptotique de la méthode de [BDGL16]. Cependant, notre impact sur la mémoire est moindre par rapport à la leur puisque la complexité mémoire de notre méthode est seulement de l'ordre de $2^{0.208n+o(n)}$.

7.1.2 La recherche de presque-collisions sur la sphère euclidienne

Nous nous intéressons à présent au problème de recherche de presque-collisions 6.1.2 sur la sphère unité de dimension n :

$$\mathcal{S}_1^n := \{ \mathbf{x} \in \mathbb{R}^n : ||\mathbf{x}|| = 1 \} \tag{7.4}$$

munie de la distance suivante :

$$\forall \mathbf{x}, \mathbf{y} \in \mathcal{S}_1^n \ , \quad \Delta(\mathbf{x}, \mathbf{y}) := \arccos(\langle \mathbf{x}, \mathbf{y} \rangle) \tag{7.5}$$

où $||\cdot||$ et $\langle \cdot, \cdot \rangle$ sont respectivement la norme ℓ_2 et le produit scalaire canonique sur $\mathbb{R}^n$. Les distances admissibles sont donc tous les réels dans $[0, \pi]$.

Remarque 7.1.1. Nous prenons cette distance plutôt que la distance euclidienne car les formules s'expriment alors plus simplement.

-

Dans l'exemple du problème SVP de la sous-section précédente, nous cherchons non seulement des couples à distance au plus $\frac{\pi}{3}$ mais aussi des couples à distance au moins $\frac{2\pi}{3}$. Nous devons donc résoudre un problème de presque-collisions et un problème de lointaines-collisions. Cependant, sur la sphère unité $\mathcal{S}_1^n$, ces deux problèmes sont équivalents. En effet, remarquons tout d'abord que la fermeture du complémentaire d'une boule fermée centrée en $\mathbf{c} \in \mathcal{S}_1^n$ et de rayon $r \in [0, \pi]$ est encore une boule fermée :

$$\overline{\mathcal{B}}(\mathbf{c}, r) = \mathcal{B}(-\mathbf{c}, \pi - r) \tag{7.6}$$

De ce fait, rechercher les couples à distance au plus w dans une liste $\mathscr{L} \subseteq \mathcal{S}_1^n$ revient à rechercher les couples à distance au moins $\pi - w$ dans $\mathscr{L} \times \overline{\mathscr{L}}$ où $\overline{\mathscr{L}} := \{\mathbf{x} \in \mathcal{S}_1^n : -\mathbf{x} \in \mathscr{L}\}$. Ainsi, nous pouvons limiter notre étude à la recherche de presque-collisions où la distance maximale w des éléments proches est dans $\left[0, \frac{\pi}{2}\right]$.

Pour résoudre le problème des presque-collisions sur $\mathcal{S}_1^n$, nous appliquons l'algorithme 6.2.1 avec des décodeurs en liste idéaux de rayon $d \in \left[0, \frac{\pi}{2}\right]$ comme fonctions de hachage floues. Dans [BDGL16], on trouve les outils nécessaires pour étudier cette méthode. En fait, leur méthode est équivalente à la notre sauf qu'au lieu de choisir la distance de décodage qui optimise la complexité, ils choisissent toujours de décoder à la distance de Gilbert-Varshamov.

Lemme 7.1.2 ([BDGL16, Lemme 2.1])**.** *Pour tout $d \in \left[0, \frac{\pi}{2}\right]$, on a :*

$$\mathcal{P}\left(\leqslant d\right) = \tilde{\Theta}\left(\sin^n(d)\right) \tag{7.7}$$

Lemme 7.1.3 ([BDGL16, Lemme 2.2])**.** *Pour tout $w \in \left[0, \frac{\pi}{2}\right]$ et tout $d \in \left[\frac{w}{2}, \frac{\pi}{2}\right]$, on a :*

$$\mathcal{P}\left(\leqslant d \mid w\right) = \tilde{\Theta}\left(\left(1 - \frac{2\cos^2(d)}{1+\cos(w)}\right)^{\frac{n}{2}}\right) \tag{7.8}$$

Corollaire 7.1.4 (du théorème 6.2.6)**.** *Nous supposons que pour tout code aléatoire $\mathcal{C} \subseteq \mathcal{S}_1^n$ de taille T et toute distance $d \in [0,1]$, il existe un décodeur en liste idéal qui s'exécute en un temps de l'ordre de $O(\max(1, T \cdot \mathcal{P}(\leqslant d)))$. L'algorithme 6.2.1 peut alors résoudre le problème des presque-collisions sur $\mathcal{S}_1^n$ en un temps de l'ordre de :*

$$T_{\text{Codes}}^{\text{Aléa}} = \begin{cases} \tilde{O}\left(\left(\dfrac{1+\cos(w)}{2L^{\frac{-2}{n}} + \cos(w) - 1}\right)^{\frac{n}{2}}\right) & \text{si } \ 1 - \cos(w) < L^{\frac{-2}{n}} \\ \tilde{O}\left(L^2 \sin^n(w)\right) & \text{sinon} \end{cases} \tag{7.9}$$

Démonstration du corollaire 7.1.4.

Les hypothèses du lemme 6.2.4 sont bien vérifiées dans $\mathcal{S}_1^n$. Nous pouvons donc appliquer le théorème 6.2.6. Pour cela, il nous faut déterminer les valeurs de d_w et $d_{\text{GV}}^-(L)$. Tout d'abord, le lemme 7.1.2 nous donne :

$$d_{\text{GV}}^-(L) = \arcsin\left(L^{\frac{-1}{n}}\right)$$

De plus, on a :

$$d_w \quad := \quad \inf\left\{d \in \left[\tfrac{w}{2}, \tfrac{\pi}{2}\right] \ : \ \mathbb{E}\left(\Delta\left(\mathbf{x}, \mathbf{y}\right)\right) \geqslant w\right\}$$

où $\mathbf{x}$ et $\mathbf{y}$ sont choisis uniformément sur une boule de $\mathcal{S}_1^n$ de rayon d. Or nous pouvons déterminer précisément $\mathbb{E}\left(\Delta\left(\mathbf{x}, \mathbf{y}\right)\right)$. Pour cela, nous allons jongler entre l'espace métrique $\mathbb{R}^n$ muni de la distance euclidienne canonique et notre espace $\mathcal{S}_1^n$. Remarquons alors que deux éléments tirés uniformément dans une boule de rayon d sur $\mathcal{S}_1^n$ sont typiquement localisés sur la sphère et forment un angle de $\frac{\pi}{2}$ avec son centre. Leur distance euclidienne est donc typiquement $\sqrt{2}\sin(d)$; sur la sphère unité $\mathcal{S}_1^n$ munie de la métrique des angles, cette distance correspond à la distance r telle que :

$$\sin^2(r) + (1 - \cos(r))^2 = \left(\sqrt{2}\sin(d)\right)^2$$

ce qui est équivalent à :

$$r = \arccos\left(\cos^2(d)\right)$$

On a finalement :

$$d_w := \arccos\left(\sqrt{\cos(w)}\right)$$

En effet, pour $d = d_w$, on a $\mathbb{E}\left(\Delta\left(\mathbf{x}, \mathbf{y}\right)\right) = w$. □

Nous pouvons montrer que la complexité donnée dans le corollaire 7.1.4 est la meilleure complexité que l'on peut obtenir sur $\mathcal{S}_1^n$ avec notre méthode. Pour cela, nous suivons la stratégie de démonstration évoquée dans la sous-section 6.2.3 du chapitre précédent. Soient f et g, les deux fonctions suivantes :

$$f(d) := \frac{L \cdot \mathcal{P}(\leqslant d)}{\mathcal{P}(\leqslant d \mid w)} \tag{7.10}$$

$$g(d) := \frac{(L \cdot \mathcal{P}(\leqslant d))^2}{\mathcal{P}(\leqslant d \mid w)} \tag{7.11}$$

La complexité optimale de notre méthode est alors $\inf_{d \in \left[\frac{w}{2}, \frac{\pi}{2}\right]} (\max(f(d), g(d)))$. Une analyse différentielle de f et g nous permet de montrer que cette complexité est minimale pour $d = \max(d_w, d^-_{\mathrm{GV}}(L))$.

La figure 7.1 donne des exemples pour illustrer les variations de f et g.

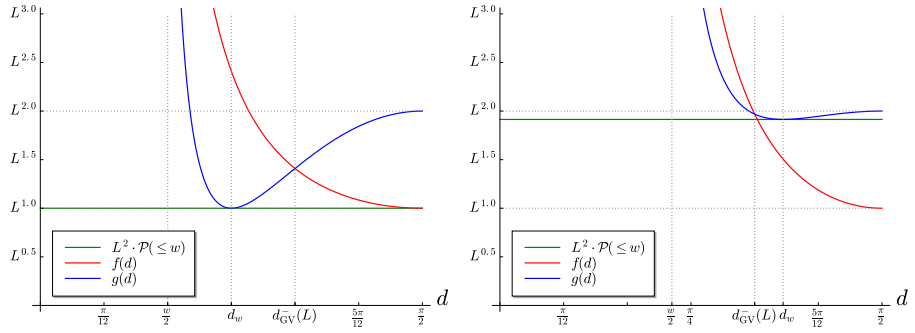


Figure 7.1 – Représentations graphiques des fonctions f et g pour $L = \left(\frac{4}{3}\right)^{\frac{n}{2}+o(n)}$. À gauche $w = 0.2$ et à droite $w = 0.48\pi$.

Finalement, la figure 7.2 représente la complexité de notre méthode pour trouver les presque-collisions dans une liste d'éléments de la sphère euclidienne.

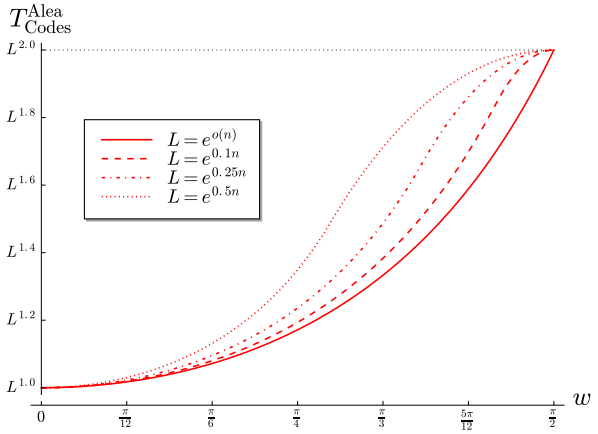


Figure 7.2 – Complexité de notre méthode pour la recherche de presque-collisions sur la sphère euclidienne.

Remarque 7.1.2. Lorsque L est sous-exponentiel en n, la complexité de notre méthode est $\tilde{O}\left(L^{\frac{2}{1+\cos(w)}}\right)$.

Rappelons toutefois que notre algorithme est théorique puisque nous ne connaissons pas de décodeur en liste de codes aléatoires aussi efficace que nous le prétendons. Pour obtenir une méthode pratique, les auteurs de [BDGL16] proposent de remplacer les codes aléatoires par des produits cartésiens de codes. Cette astuce est la même que celle utilisée dans [GMO10] et [Dub10]. Nous avons vu dans la section 5.6 que dans le modèle de proximité particulier considéré dans [GMO10], l'utilisation de produits cartésiens de codes n'engendrait pas de surcoût sur la complexité. Cependant, dans le chapitre 9, nous verrons que dans le modèle de proximité qui nous intéresse, cette méthode produit un surcoût super-polynomial. Nous proposerons alors d'autres astuces qui réduiront drastiquement ce surcoût.

Nous avons vu dans la sous-section précédente que la résolution du problème SVP par tamisage se réduit avec un facteur polynomial au problème des presque-collisions où $L = \left(\frac{4}{3}\right)^{\frac{n}{2}+o(n)}$ et $w = \frac{\pi}{3}$. Ainsi, en appliquant le théorème 6.2.3 et le corollaire 7.1.4 à cette instance du problème, on obtient une complexité mémoire pour la résolution du problème SVP de l'ordre de :

$$S_{\mathrm{SVP}} = \left(\frac{4}{3}\right)^{\frac{n}{2}+o(n)} = 2^{0.2075n+o(n)} \tag{7.12}$$

et une complexité en temps de l'ordre de :

$$T_{\mathrm{SVP}} = \left(\frac{3}{2}\right)^{\frac{n}{2}+o(n)} = 2^{0.2925n+o(n)} \tag{7.13}$$

7.2 Les espaces de Hamming binaires

Le cas qui nous intéresse à présent est celui où l'espace métrique ambiant $\mathscr{E}$ est $\mathbb{F}_2^n$ muni de la métrique de Hamming. Les applications de cette instance sont nombreuses et nous en aborderons certaines dans la partie suivante.

Nous rappelons tout d'abord la définition de la distance de Hamming :

$$\forall \mathbf{x}, \mathbf{y} \in \mathbb{F}_2^n \ , \quad \Delta(\mathbf{x}, \mathbf{y}) := \#\,\mathrm{supp}(\mathbf{x} + \mathbf{y}) \tag{7.14}$$

où $\mathrm{supp}(\mathbf{x})$ est le support de $\mathbf{x}$; c'est-à-dire l'ensemble des positions non-nulles de $\mathbf{x}$. On note $|\mathbf{x}| := \Delta(\mathbf{0}, \mathbf{x})$ le poids de Hamming de $\mathbf{x}$.

Comme pour la sphère euclidienne, la fermeture du complémentaire d'une boule fermée est encore une boule fermée. Plus exactement, pour tout $\mathbf{c} \in \mathbb{F}_2^n$ et $r \in [\![0, n]\!]$, nous avons :

$$\overline{\mathcal{B}}(\mathbf{c}, r) = \mathcal{B}(\mathbf{1} + \mathbf{c}, n - r) \tag{7.15}$$

De ce fait, rechercher les couples à distance au plus w dans une liste $\mathscr{L} \subseteq \mathbb{F}_2^n$ revient à rechercher les couples à distance au moins $n-w$ dans $\mathscr{L} \times \overline{\mathscr{L}}$ où $\overline{\mathscr{L}} := \{\mathbf{x} \in \mathbb{F}_2^n : \mathbf{1} + \mathbf{x} \in \mathscr{L}\}$. Ainsi, nous pouvons limiter notre étude à la recherche de presque-collisions où la distance maximale w des éléments proches est dans $[\![0, \frac{n}{2}]\!]$.

Pour résoudre le problème des presque-collisions dans $\mathbb{F}_2^n$, nous appliquons l'algorithme 6.2.1 avec des décodeurs en liste idéaux de rayon $d \in [\![0, n]\!]$ comme fonctions de hachage floues. Nous supposons que nos décodeurs s'exécutent en un temps de l'ordre de la taille des listes qu'ils retournent. Dans le chapitre 9, nous verrons comment remédier au fait que nous ne connaissons pas de tels décodeurs.

L'analyse de notre méthode de recherche de presque-collisions dans les espaces binaires de Hamming repose essentiellement sur les deux lemmes suivants :

Lemme 7.2.1. *Soit $d := \lceil \delta n \rceil$ pour une constante $\delta \in \left[0, \frac{1}{2}\right]$. On a :*

$$\mathcal{P}(\leqslant d) = \tilde{\Theta}\left(2^{-n(1-h_2(\delta))}\right) \tag{7.16}$$

Démonstration du lemme 7.2.1.

Nous rappelons tout d'abord que :

$$\begin{aligned}\mathcal{P}(\leqslant d) &:= \mathbb{P}(\mathbf{u} \in \mathcal{B}(\mathbf{x}, d)) \\ &= \mathbb{P}(\mathbf{u} \in \mathcal{B}(\mathbf{0}, d)) \\ &= \frac{\#\mathcal{B}(\mathbf{0}, d)}{\#\mathbb{F}_2^n}\end{aligned}$$

où $\mathbf{u}$ est choisi uniformément dans $\mathbb{F}_2^n$ et $\mathbf{x}$ est un vecteur quelconque de $\mathbb{F}_2^n$.
Or nous avons $\#\mathcal{B}(\mathbf{0}, d) = \sum_{u=0}^{d} \binom{n}{u}$. Puisque d est plus petit que $\frac{n}{2}$, la somme précédente est dominée par le terme $\binom{n}{d}$. La formule de Stirling (cf. théorème 1.1.4) nous donne alors :

$$\binom{n}{d} = \tilde{\Theta}\left(2^{nh_2(\delta)}\right)$$

Nous terminons la preuve en remarquant que $\#\mathbb{F}_2^n = 2^n$. □

Lemme 7.2.2. *Soient $w := \lfloor \omega n \rceil$ et $d := \lceil \delta n \rceil$ pour des constantes $\omega \in \left[0, \frac{1}{2}\right]$ et $\delta \in \left[\frac{\omega}{2}, \frac{1}{2}\right]$. On a :*

$$\mathcal{P}(\leqslant d \mid w) = \tilde{\Theta}\left(2^{-n(1-\omega)\left(1-h_2\left(\frac{\delta-\omega/2}{1-\omega}\right)\right)}\right) \tag{7.17}$$

Démonstration du lemme 7.2.2.

Nous rappelons tout d'abord que :

$$\begin{aligned}\mathcal{P}(\leqslant d \mid w) &:= \mathbb{P}(\mathbf{u} \in \mathcal{B}(\mathbf{x}, d) \cap \mathcal{B}(\mathbf{y}, d)) \\ &= \mathbb{P}(\mathbf{u} \in \mathcal{B}(\mathbf{0}, d) \cap \mathcal{B}(\mathbf{z}, d)) \\ &= \frac{\#\mathcal{B}(\mathbf{0}, d) \cap \mathcal{B}(\mathbf{z}, d)}{\#\mathbb{F}_2^n}\end{aligned}$$

où $\mathbf{u}$ est choisi uniformément dans $\mathbb{F}_2^n$ et $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{F}_2^n$ sont tels que $\Delta(\mathbf{x}, \mathbf{y}) = |\mathbf{z}| = w$.
Soit $\mathbf{c} \in \mathcal{B}(\mathbf{0}, d) \cap \mathcal{B}(\mathbf{z}, d)$ et soient α et β les deux quantités suivantes :

$$\begin{aligned}\alpha &:= \frac{\#(\mathrm{supp}(\mathbf{z}) \cap \mathrm{supp}(\mathbf{c}))}{n} \\ \beta &:= \frac{\#(\mathrm{supp}(\mathbf{c}) \setminus \mathrm{supp}(\mathbf{z}))}{n}\end{aligned}$$

La figure 7.3 représente alors la situation.

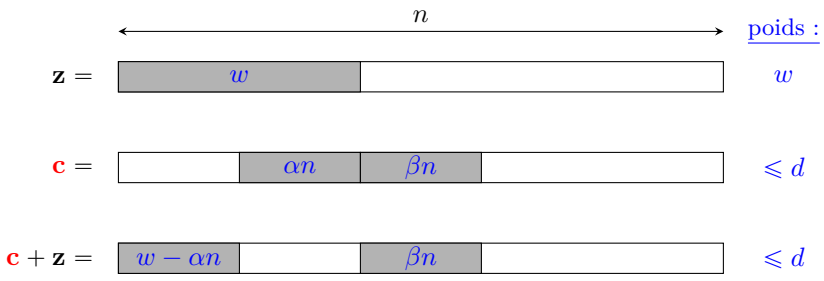


Figure 7.3 – Configuration du vecteur $\mathbf{c} \in \mathcal{B}(\mathbf{0}, d) \cap \mathcal{B}(\mathbf{z}, d)$ à permutation près des positions.

Les poids relatifs α et β sont nécessairement tels que $|\mathbf{c}| = \alpha n + \beta n \leqslant d$ et $\Delta(\mathbf{c}, \mathbf{z}) = w - \alpha n + \beta n \leqslant d$. Donc nous avons finalement :

$$\#\mathcal{B}(\mathbf{0}, d) \cap \mathcal{B}(\mathbf{z}, d) = \sum_{\alpha n=0}^{w} \sum_{\beta n=0}^{d-\max(\alpha n, w-\alpha n)} \binom{w}{\alpha n}\binom{n-w}{\beta n}$$

Puisque $d \in [\![0, \frac{n}{2}]\!]$, la double somme précédente est dominée par le terme qui correspond à $|\mathbf{c}| = d$ et $\Delta(\mathbf{c}, \mathbf{z}) = d$; c'est-à-dire par le terme correspondant à $\alpha n = \frac{w}{2}$ et $\beta n = d - \frac{w}{2}$ (sans perte de généralité, on suppose n tel que w soit pair). Cela signifie que les éléments de $\mathcal{B}(\mathbf{0}, d) \cap \mathcal{B}(\mathbf{z}, d)$ sont concentrés sur $\mathcal{S}(\mathbf{0}, d) \cap \mathcal{S}(\mathbf{z}, d)$ où $\mathcal{S}(\mathbf{u}, \rho) := \{\mathbf{x} \in \mathbb{F}_2^n \ : \ \Delta(\mathbf{x}, \mathbf{u}) = \rho\}$. Nous avons ainsi :

$$\begin{aligned}\#\mathcal{B}(\mathbf{0}, d) \cap \mathcal{B}(\mathbf{z}, d) &= \tilde{\Theta}\left(\binom{w}{\frac{w}{2}}\binom{n-w}{d-\frac{w}{2}}\right) \\ &= \tilde{\Theta}\left(2^{n\left(\omega + (1-\omega) h_2\left(\frac{\delta - \omega/2}{1-\omega}\right)\right)}\right)\end{aligned}$$

Nous terminons finalement la preuve en remarquant que $\#\mathbb{F}_2^n = 2^n$. □

Corollaire 7.2.3 (du théorème 6.2.6)**.** *Supposons que pour tout code aléatoire $\mathcal{C} \subseteq \mathbb{F}_2^n$ de taille T et pour toute distance $d \in [\![0, \frac{n}{2}]\!]$, il existe un décodeur en liste idéal qui s'exécute en un temps de l'ordre de $O(\max(1, T \cdot \mathcal{P}(\leqslant d)))$. L'algorithme 6.2.1 peut alors résoudre le problème des presque-collisions sur $\mathbb{F}_2^n$ en un temps de l'ordre de :*

$$T_{\text{Codes}}^{\text{Aléa}} = \begin{cases} \tilde{O}\left(2^{n(1-\omega)\left(1 - h_2\left(\frac{h_2^{-1}(1-\lambda) - \omega/2}{1-\omega}\right)\right)}\right) & \text{si } \frac{1-\sqrt{1-2\omega}}{2} < h_2^{-1}(1-\lambda) \\ \tilde{O}\left(2^{n(2\lambda - 1 + h_2(\omega))}\right) & \text{sinon} \end{cases} \tag{7.18}$$

où $L := 2^{\lambda n}$ et $w := \lfloor \omega n \rfloor$ sont les paramètres du problème de presque-collisions pour des constantes $\lambda \in [0, 1]$ et $\omega \in [0, \frac{1}{2}]$.

Démonstration du corollaire 7.2.3.

Les hypothèses du lemme 6.2.4 sont bien vérifiées dans $\mathbb{F}_2^n$. Nous pouvons donc appliquer le théorème 6.2.6. Pour cela, nous devons déterminer les valeurs de $d_{\text{GV}}^-(L)$ et d_w. Nous utilisons tout d'abord le lemme 7.2.1 pour montrer que :

$$d_{\text{GV}}^-(L) = \left\lceil \delta_{\text{GV}}^-(L) n \right\rceil \qquad \text{avec} \qquad \delta_{\text{GV}}^-(L) := h_2^{-1}(1-\lambda)$$

D'autre part, pour $\mathbf{x}$ et $\mathbf{y}$ tirés uniformément dans la boule $\mathcal{B}(\mathbf{0}, d)$, nous savons que $\mathbf{x}$ et $\mathbf{y}$ sont typiquement de poids d. Or, en moyenne, le support de $\mathbf{y}$ est équitablement réparti sur le support de $\mathbf{x}$ et son complémentaire donc la taille moyenne de $\text{supp}(\mathbf{x}) \cap \text{supp}(\mathbf{y})$ est $\frac{d^2}{n}$. Nous avons ainsi $\mathbb{E}(\Delta(\mathbf{x}, \mathbf{y})) = 2\left(d - \frac{d^2}{n}\right)$. Et finalement :

$$\begin{aligned} d_w &:= \inf\left\{d \in [\![\tfrac{w}{2}, n]\!] \ : \ \mathbb{E}(\Delta(\mathbf{x}, \mathbf{y})) \geqslant w\right\} \\ &= \lceil \delta_w n \rceil \qquad \text{avec} \qquad \delta_w := \tfrac{1-\sqrt{1-2\omega}}{2} \end{aligned}$$

□

Comme pour la sphère euclidienne, nous pouvons montrer que la complexité donnée par le corollaire 7.2.3 est optimale. Pour cela, on procède de façon analogue à la section précédente en effectuant une analyse différentielle des fonctions f et g définies comme suit :

$$f(d) := \frac{L \cdot \mathcal{P}(\leqslant d)}{\mathcal{P}(\leqslant d \mid w)} \tag{7.19}$$

$$g(d) := \frac{(L \cdot \mathcal{P}(\leqslant d))^2}{\mathcal{P}(\leqslant d \mid w)} \tag{7.20}$$

La figure 7.4 donne des exemples pour illustrer les variations de f et g dans le modèle binaire de Hamming. Sur cette figure, on pose $d := \lceil \delta n \rceil$.

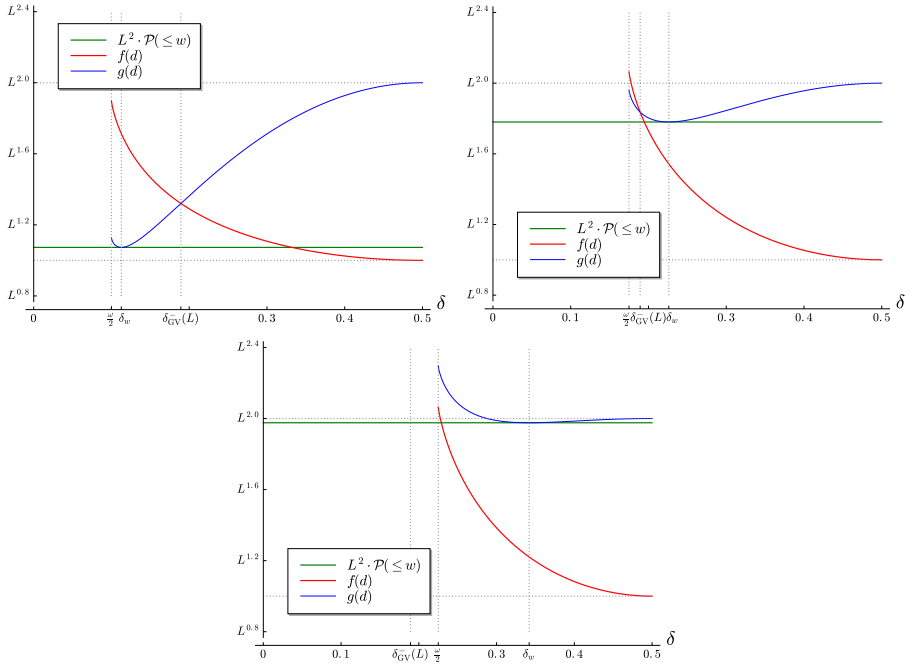


Figure 7.4 – Représentations graphiques des fonctions f et g pour $L = 2^{0.3n}$.
À gauche $w = 0.2$, à droite $w = 0.35$ et en bas $w = 0.45$.

Finalement, la figure 7.5 représente la complexité de notre méthode pour trouver les presque-collisions dans une liste d'éléments d'un espace binaire de Hamming.

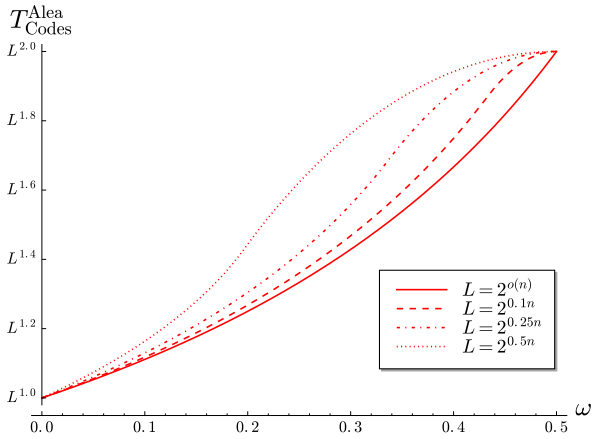


Figure 7.5 – Complexité de notre méthode pour la recherche de presque-collisions dans les espaces binaires.

Remarque 7.2.1. Lorsque L est sous-exponentiel en n, la complexité de notre méthode est $\tilde{O}\left(L^{\frac{1}{1-w}}\right)$.

7.3 Les espaces de Hamming non-binaires

Dans cette section, nous généralisons l'étude de notre méthode de recherche de presque/lointaines-collisions aux espaces de Hamming non-binaires. Dans les espaces métriques que nous avons explorés précédemment, la recherche de presque-collisions s'avérait être équivalente à la recherche de lointaines-collisions. Ce phénomène ne se vérifie pas dans les espaces de Hamming non-binaires ; il nous faudra donc distinguer les deux problèmes.

Nous nous intéressons donc aux cas où l'espace ambiant $\mathscr{E}$ est un espace vectoriel de dimension n sur le corps fini $\mathbb{F}_q$. Pour être en accord avec le titre de cette section, nous devrions supposer $q \neq 2$ mais en fait, tout ce que nous allons dire par la suite est aussi valable pour les espaces binaires ; la section précédente est donc un cas particulier de l'étude que nous faisons ici. En outre, nous pouvons aussi généraliser notre propos aux espaces $\mathbb{Z}_q^n$ où $\mathbb{Z}_q$ est l'anneau $\mathbb{Z}/q\mathbb{Z}$.

Nous munissons $\mathbb{F}_q^n$ de la distance de Hamming :

$$\forall \mathbf{x}, \mathbf{y} \in \mathbb{F}_q^n \ , \quad \Delta(\mathbf{x}, \mathbf{y}) := \# \operatorname{supp}(\mathbf{x} - \mathbf{y}) \tag{7.21}$$

où $\operatorname{supp}(\mathbf{x})$ est le support de $\mathbf{x}$; c'est-à-dire l'ensemble des positions non-nulles de $\mathbf{x}$. On note $|\mathbf{x}| := \Delta(\mathbf{0}, \mathbf{x})$ le poids de Hamming de $\mathbf{x}$.

Contrairement aux espaces que nous avons étudiés jusqu'à présent, dans les espaces de Hamming non-binaires la fermeture du complémentaire d'une boule fermée n'est pas une boule fermée :

$$\overline{\mathcal{B}}(\mathbf{c}, r) = \bigcup_{\{\mathbf{e}:|\mathbf{e}|=n\}} \mathcal{B}(\mathbf{c} + \mathbf{e}, n - r) \tag{7.22}$$

De ce fait, la réduction naïve du problème des lointaines-collisions à celui des presque-collisions est exponentielle en n dès lors que $q > 2$. C'est pourquoi nous devons traiter les deux problèmes séparément.

Notation 7.3.1. *Dans la suite, nous utiliserons fréquemment la fonction entropie q-aire de Shannon. Nous aurons notamment besoin de l'inverser. Nous notons alors g_q^- (resp. g_q^+) la fonction inverse de h_q lorsque celle-ci est restreinte au domaine $\left[0, 1 - \frac{1}{q}\right]$ (resp. $\left[1 - \frac{1}{q}, 1\right]$).*

7.3.1 La recherche de presque-collisions dans les espaces de Hamming non-binaires

Dans $\mathbb{F}_q^n$, le problème de recherche de presque-collisions n'est pertinent que pour $w \leqslant n - \frac{n}{q}$. En effet, au delà de cette borne, tous les couples de $\mathscr{L}$ sont typiquement proches. Aussi, pour $w \geqslant n - \frac{n}{q}$, nous nous intéresserons essentiellement au problème des lointaines-collisions.

Nous résolvons le problème des presque-collisions sur $\mathbb{F}_q^n$ en appliquant l'algorithme 6.2.1 avec des décodeurs en liste idéaux de rayon $d \in [\![0, n]\!]$ comme fonctions de hachage floues. Nous supposons que nos décodeurs s'exécutent en un temps de l'ordre de la taille des listes qu'ils retournent. Dans le chapitre 9 nous discuterons plus en détail de la construction de tels décodeurs pour certaines familles de codes.

Nous commençons l'analyse de notre méthode de recherche de presque-collisions sur $\mathbb{F}_q^n$ en montrant les deux lemmes suivants :

Lemme 7.3.2. *Soit $d := \lceil \delta n \rceil$ pour une constante $\delta \in \left[0, 1 - \frac{1}{q}\right]$. On a :*

$$\mathcal{P}(\leqslant d) = \tilde{\Theta}\left(q^{-n(1-h_q(\delta))}\right) \tag{7.23}$$

Démonstration du lemme 7.3.2.

Nous rappelons tout d'abord que :

$$\begin{aligned}\mathcal{P}(\leqslant d) &:= \mathbb{P}(\mathbf{u} \in \mathcal{B}(\mathbf{x}, d)) \\ &= \mathbb{P}(\mathbf{u} \in \mathcal{B}(\mathbf{0}, d)) \\ &= \frac{\#\mathcal{B}(\mathbf{x}, d)}{\#\mathbb{F}_q^n}\end{aligned}$$

où $\mathbf{u}$ est choisi uniformément dans $\mathbb{F}_q^n$ et $\mathbf{x}$ est un vecteur quelconque de $\mathbb{F}_q^n$.
Or $\#\mathcal{B}(\mathbf{0}, d) = \sum_{u=0}^{d} \binom{n}{u}(q-1)^u$. Puisque d est plus petit que $n - \frac{n}{q}$, la somme précédente est dominée par le terme $\binom{n}{d}$. Nous avons alors :

$$\#\mathcal{B}(\mathbf{0}, d) = \tilde{\Theta}\left(q^{nh_q(\delta)}\right)$$

Nous terminons la preuve en remarquant que $\#\mathbb{F}_q^n = q^n$. □

Lemme 7.3.3. *Soient $w := \lfloor \omega n \rceil$ et $d := \lceil \delta n \rceil$ pour des constantes $\omega \in \left[0, 1 - \frac{1}{q}\right]$ et $\delta \in \left[\frac{\omega}{2}, 1 - \frac{1}{q}\right]$. On a :*

$$\mathcal{P}(\leqslant d \mid w) = \tilde{\Theta}\left(\left(\frac{\left(1 - \frac{1}{q-1}\right)^{2(\delta-\alpha_0)-\omega}\left(\frac{1}{q-1}\right)^{(\delta-\alpha_0)}}{q^{1-(1-\omega)h_q\left(\frac{\alpha_0}{1-\omega}\right)-\omega h_q\left(\frac{\delta-\alpha_0}{\omega}\right)-(\delta-\alpha_0)h_q\left(2-\frac{\omega}{\delta-\alpha_0}\right)}}\right)^n\right) \tag{7.24}$$

où $\alpha_0 \in \left[\max(0, \delta - \omega), \min\left(1 - \omega, \delta - \frac{\omega}{2}\right)\right]$ maximise l'expression.

Démonstration du lemme 7.3.3.

Nous commençons par rappeler que :

$$\begin{aligned}\mathcal{P}(\leqslant d \mid w) &:= \mathbb{P}(\mathbf{u} \in \mathcal{B}(\mathbf{x}, d) \cap \mathcal{B}(\mathbf{y}, d)) \\ &= \mathbb{P}(\mathbf{u} \in \mathcal{B}(\mathbf{0}, d) \cap \mathcal{B}(\mathbf{z}, d)) \\ &= \frac{\#\mathcal{B}(\mathbf{0}, d) \cap \mathcal{B}(\mathbf{z}, d)}{\#\mathbb{F}_q^n}\end{aligned}$$

où $\mathbf{u}$ est choisi uniformément dans $\mathbb{F}_q^n$ et $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{F}_q^n$ sont tels que $\Delta(\mathbf{x}, \mathbf{y}) = |\mathbf{z}| = w$.
Soit $\mathbf{c} \in \mathcal{B}(\mathbf{0}, d) \cap \mathcal{B}(\mathbf{z}, d)$ et soient α, β et γ les trois quantités suivantes :

$$\begin{aligned}\alpha &:= \frac{\#\operatorname{supp}(\mathbf{c}) \setminus \operatorname{supp}(\mathbf{z})}{n} \\ \beta &:= \frac{\#\operatorname{supp}(\mathbf{c} - \mathbf{z}) \cap \operatorname{supp}(\mathbf{z})}{n} \\ \gamma &:= \frac{\#\operatorname{supp}(\mathbf{c}) \cap \operatorname{supp}(\mathbf{z}) \cap \operatorname{supp}(\mathbf{c} - \mathbf{z})}{n}\end{aligned}$$

Nous représentons la situation à l'aide de la figure 7.6.

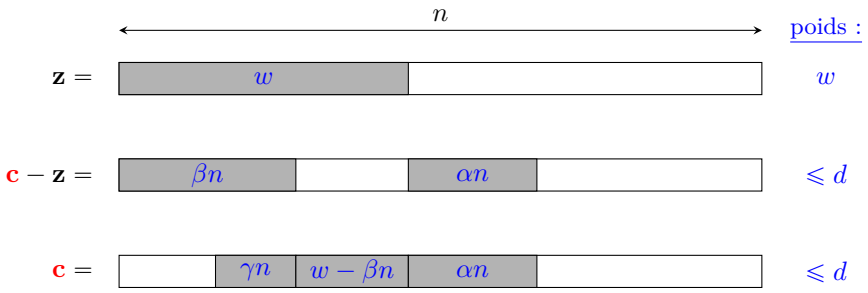


Figure 7.6 – Configuration du vecteur $\mathbf{c} \in \mathcal{B}(\mathbf{0}, d) \cap \mathcal{B}(\mathbf{z}, d)$ à permutation près des positions.

Les poids relatifs α, β et γ sont nécessairement tels que $\Delta(\mathbf{c}, \mathbf{z}) = \alpha n + \beta n \leqslant d$ et $|\mathbf{c}| = w - \beta n + \alpha n + \gamma n \leqslant d$. Nous avons donc :

$$\#\mathcal{B}(\mathbf{0}, d) \cap \mathcal{B}(\mathbf{z}, d) =$$

$$\sum_{\alpha n=0}^{d} \sum_{\beta n=0}^{d-\alpha n} \sum_{\gamma n=0}^{d+\beta n-w-\alpha n} \binom{n-w}{\alpha n} \binom{w}{\beta n} \binom{\beta n}{\gamma n} (q-2)^{\gamma n} (q-1)^{\alpha n}$$

Puisque $d \leqslant n - \frac{n}{q}$, la somme précédente est dominée par les termes vérifiant $|\mathbf{c}| = \Delta(\mathbf{c}, \mathbf{z}) = d$; c'est-à-dire les termes vérifiant $\beta n = d - \alpha n$ et $\gamma n = 2(d - \alpha n) - w$. Nous avons ainsi :

$$\#\mathcal{B}(\mathbf{0}, d) \cap \mathcal{B}(\mathbf{z}, d) =$$

$$\tilde{\Theta}\left(\sum_{\alpha n=0}^{d} \binom{n-w}{\alpha n} \binom{w}{d-\alpha n} \binom{d-\alpha n}{2(d-\alpha n)-w} (q-2)^{2(d-\alpha n)-w} (q-1)^{\alpha n} \right)$$

En nous restreignant exclusivement aux termes non-nuls, nous obtenons :

$$\#\mathcal{B}(\mathbf{0}, d) \cap \mathcal{B}(\mathbf{z}, d) =$$

$$\tilde{\Theta}\left(\sum_{\alpha n \in \overline{A}} \binom{n-w}{\alpha n} \binom{w}{d-\alpha n} \binom{d-\alpha n}{(2(d-\alpha n)-w} (q-2)^{2(d-\alpha n)-w} (q-1)^{\alpha n} \right)$$

où $\overline{A} := [\![0, n-w]\!] \cap [\![d-w, d-\frac{w}{2}]\!]$.
Nous avons donc finalement :

$$\#\mathcal{B}(\mathbf{0}, d) \cap \mathcal{B}(\mathbf{z}, d) =$$

$$\tilde{\Theta}\left(\sup_{\alpha \in A} \left(q^{(1-\omega)h_q\left(\frac{\alpha}{1-\omega}\right) + \omega h_q\left(\frac{\delta-\alpha}{\omega}\right) + (\delta-\alpha)h_q\left(2-\frac{\omega}{\delta-\alpha}\right)} \cdot \frac{(q-2)^{2(\delta-\alpha)-\omega}}{(q-1)^{3(\delta-\alpha)-\omega}} \right)^n \right)$$

où $A := \left[\max(0, \delta - \omega), \min\left(1-\omega, \delta - \frac{\omega}{2}\right)\right]$.
Nous terminons la preuve en remarquant que $\#\mathbb{F}_q^n = q^n$. □

À propos de la valeur de α_0 dans le lemme 7.3.3. Nous pouvons être un peu plus précis quant à la valeur de α_0 dans le lemme 7.3.3. En effet, le problème d'optimisation dont il est issu se résout par une analyse différentielle de la fonction φ définie comme suit :

$$\begin{aligned} \varphi(\alpha) \quad := \quad & (1-\omega)h_q\left(\tfrac{\alpha}{1-\omega}\right) + \omega h_q\left(\tfrac{\delta-\alpha}{\omega}\right) + (\delta-\alpha)h_q\left(2-\tfrac{\omega}{\delta-\alpha}\right) \\ & +(2(\delta-\alpha)-\omega)\log_q(q-2) - (3(\delta-\alpha)-\omega)\log_q(q-1) \end{aligned} \tag{7.25}$$

où $\alpha \in \left[\max\left(0, \delta - \omega\right), \min\left(1-\omega, \delta - \frac{\omega}{2}\right)\right]$.

Une analyse de la dérivée de φ nous permet de montrer que son maximum est atteint sur une solution de l'équation suivante :

$$(q-1)\left(1-\omega-\alpha\right)\left(2(\delta-\alpha)-\omega\right)^2 = \alpha(q-2)^2\left(\omega-\delta+\alpha\right)^2 \tag{7.26}$$

Nous pouvons alors utiliser la méthode de Cardan pour déterminer la valeur de α_0 (ce sera l'unique racine réelle dans A).

Corollaire 7.3.4 (du théorème 6.2.6). *Supposons que pour tout code aléatoire $\mathcal{C} \subseteq \mathbb{F}_q^n$ de taille T et pour toute distance $d \in \left[\!\left[0, n - \frac{n}{q}\right]\!\right]$, il existe un décodeur en liste idéal qui s'exécute en un temps de l'ordre de $O(\max\left(1, T \cdot \mathcal{P}\left(\leqslant d\right)\right))$. L'algorithme 6.2.1 peut alors résoudre le problème des presque-collisions sur $\mathbb{F}_q^n$ en un temps de l'ordre de :*

$$T_{\text{Codes}}^{\text{Aléa}} = \begin{cases} \tilde{O}\left(\dfrac{1}{\mathcal{P}\left(\leqslant \lceil n g_q^-(1-\lambda)\rceil \mid w\right)}\right) & \text{si} \quad \left(1-\frac{1}{q}\right)\left(1-\sqrt{1-\frac{q\omega}{q-1}}\right) < g_q^-(1-\lambda) \\ \tilde{O}\left(q^{n(2\lambda-1+h_q(\omega))}\right) & \text{sinon} \end{cases} \tag{7.27}$$

où $L := q^{\lambda n}$ et $w := \lceil \omega n \rceil$ sont les paramètres du problème de presque-collisions pour des constantes $\lambda \in [0,1]$ et $\omega \in \left[0, 1-\frac{1}{q}\right]$.

Démonstration du corollaire 7.2.3.

Les hypothèses du lemme 6.2.4 sont bien vérifiées dans $\mathbb{F}_q^n$. Nous pouvons donc appliquer le théorème 6.2.6. Pour cela, nous devons déterminer les valeurs de $d_{\text{GV}}^-(L)$ et d_w. Le lemme 7.3.2 nous permet de montrer que :

$$d_{\text{GV}}^-(L) = \left\lceil \delta_{\text{GV}}^-(L) n \right\rceil \qquad \text{avec} \qquad \delta_{\text{GV}}^-(L) := g_q^-(1-\lambda)$$

D'autre part, pour toute distance $d \in \left[\!\left[0, n - \frac{n}{q}\right]\!\right]$ et pour $\mathbf{x}$ et $\mathbf{y}$ tirés uniformément dans la boule $\mathcal{B}\left(\mathbf{0}, d\right)$, nous savons que $\mathbf{x}$ et $\mathbf{y}$ sont typiquement de poids d. Or, en moyenne, le support de $\mathbf{y}$ est équitablement réparti sur le support de $\mathbf{x}$ et son complémentaire donc la taille moyenne de $\text{supp}\left(\mathbf{x}\right) \cap \text{supp}\left(\mathbf{y}\right)$ est $\frac{d^2}{n}$. De plus, pour tout $i \in \text{supp}\left(\mathbf{x}\right) \cap \text{supp}\left(\mathbf{y}\right)$, la probabilité $\mathbb{P}\left(x_i \neq y_i\right) = \frac{q-2}{q-1}$ où $\mathbf{x} := (x_1, \cdots, x_n)$ et $\mathbf{y} := (y_1, \cdots, y_n)$. Nous résumons la situation avec la figure 7.7.

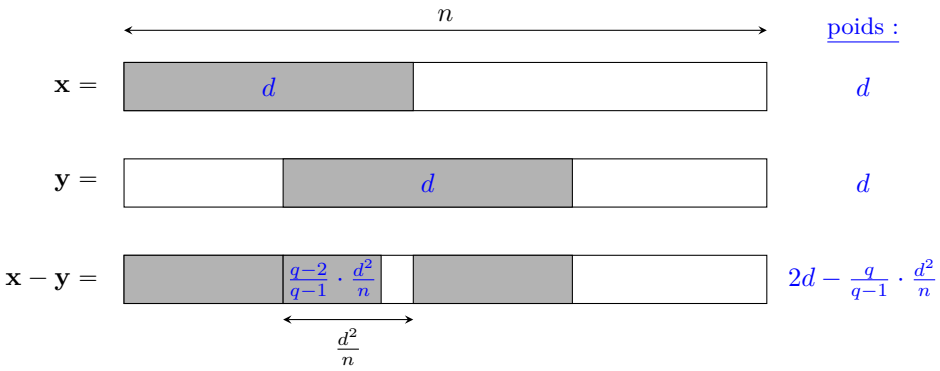


Figure 7.7 – Configuration typique de $\mathbf{x}$ et $\mathbf{y}$ à permutation près des positions.

Nous avons ainsi $\mathbb{E}\left(\Delta\left(\mathbf{x},\mathbf{y}\right)\right) = 2d - \frac{q}{q-1}\cdot\frac{d^2}{n}$ et donc finalement :

$$\begin{aligned} d_w &:= \inf\left\{d \in \left[\!\left[\tfrac{w}{2}, n\right]\!\right] \;:\; \mathbb{E}\left(\Delta\left(\mathbf{x},\mathbf{y}\right)\right) \geqslant w\right\} \\ &= \lceil \delta_w n \rceil \qquad \text{avec} \qquad \delta_w := \left(1-\tfrac{1}{q}\right)\left(1-\sqrt{1-\tfrac{q\omega}{q-1}}\right) \end{aligned}$$

□

Nous pouvons montrer que la complexité donnée par le corollaire 7.3.4 est optimale. Pour cela, on procède de façon analogue à la section précédente en effectuant une analyse différentielle des fonctions f et g définies comme suit :

$$f(d) := \frac{L\cdot\mathcal{P}\left(\leqslant d\right)}{\mathcal{P}\left(\leqslant d \mid w\right)} \tag{7.28}$$

$$g(d) := \frac{\left(L\cdot\mathcal{P}\left(\leqslant d\right)\right)^2}{\mathcal{P}\left(\leqslant d \mid w\right)} \tag{7.29}$$

La figure 7.8 donne des exemples pour illustrer les variations de f et g dans le modèle q-aire de Hamming. Sur cette figure, on pose $d := \lceil \delta n \rceil$.

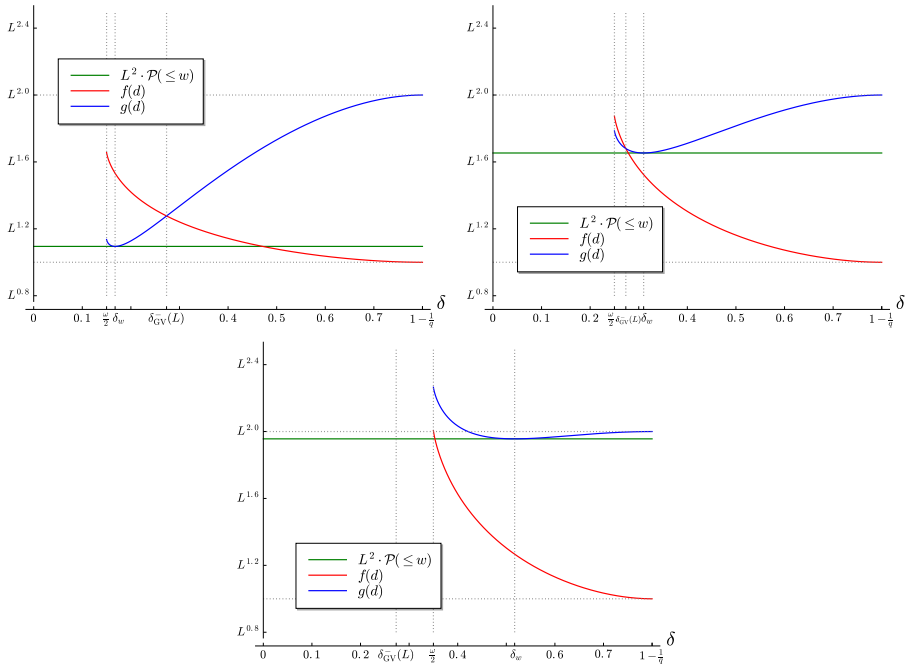


Figure 7.8 – Représentations graphiques des fonctions f et g pour $q = 5$ et $L = q^{0.4n}$. À gauche $w = 0.3$, à droite $w = 0.5$ et en bas $w = 0.7$.

Finalement, la figure 7.9 représente la complexité de notre méthode pour trouver les presque-collisions dans une liste d'éléments d'un espace q-aire de Hamming.

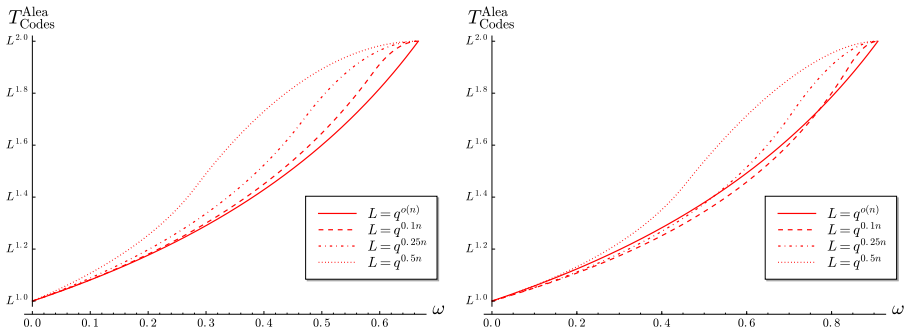


Figure 7.9 – Complexité de notre méthode pour la recherche de presque-collisions dans les espaces q-aire de Hamming. À gauche $q = 3$ et à droite $q = 11$.

Remarque 7.3.1. Lorsque L est sous-exponentiel en n, la complexité de notre méthode est $O\left(L^{\frac{1}{1-\frac{qw}{2(q-1)}}}\right)$.

Remarque 7.3.2. Pour q et w fixés, la complexité de notre méthode est croissante en fonction de L. Sur la figure 7.9, nous avons représenté cette complexité avec une échelle logarithmique en base L ; c'est pourquoi l'affirmation précédente peut paraître erronée mais il n'en est rien.

7.3.2 La recherche de lointaines-collisions dans les espaces de Hamming non-binaires

Nous avons vu précédemment qu'il n'y a pas de réduction triviale du problème des lointaines-collisions au problème des presque-collisions dans les espaces de Hamming non-binaires. Dans ce modèle, nous allons donc utiliser l'algorithme 6.3.1 pour trouver les lointaines-collisions. Pour analyser celui-ci, nous devons d'abord montrer les deux lemmes suivants :

Lemme 7.3.5. *Soit* $d := \lceil \delta n \rceil$ *pour une constante* $\delta \in \left[1 - \frac{1}{q}, 1\right]$. *On a :*

$$\mathcal{P}(\geqslant d) = \tilde{\Theta}\left(q^{-n(1-h_q(\delta))}\right) \tag{7.30}$$

Démonstration du lemme 7.3.5.

Nous rappelons tout d'abord que :

$$\begin{aligned} \mathcal{P}(\geqslant d) &:= \mathbb{P}\left(\mathbf{u} \in \overline{\mathcal{B}}(\mathbf{x}, d)\right) \\ &= \mathbb{P}\left(\mathbf{u} \in \overline{\mathcal{B}}(\mathbf{0}, d)\right) \\ &= \frac{\#\overline{\mathcal{B}}(\mathbf{x}, d)}{\#\mathbb{F}_q^n} \end{aligned}$$

où $\mathbf{u}$ est choisi uniformément dans $\mathbb{F}_q^n$ et $\mathbf{x}$ est un vecteur quelconque dans $\mathbb{F}_q^n$.
Or $\#\overline{\mathcal{B}}(\mathbf{0}, d) = \sum_{u=d}^{n} \binom{n}{u}(q-1)^u$. Puisque d est plus grand que $n - \frac{n}{q}$, la somme précédente est dominée par le terme $\binom{n}{d}$. Nous avons alors :

$$\#\overline{\mathcal{B}}(\mathbf{0}, d) = \tilde{\Theta}\left(q^{nh_q(\delta)}\right)$$

Nous terminons la preuve en remarquant que $\#\mathbb{F}_q^n = q^n$. □

Lemme 7.3.6. *Soient* $w := \lceil \omega n \rceil$, $d_1 = \lceil \delta_1 n \rceil$ *et* $d_2 = \lfloor \delta_2 n \rfloor$ *pour des constantes* $\omega \in \left[1 - \frac{1}{q}, 1\right]$, $\delta_1 \in \left[0, 1 - \frac{1}{q}\right]$ *et* $\delta_2 \in \left[1 - \frac{1}{q}, 1\right]$ *telles que* $\delta_2 - \delta_1 \leqslant \omega \leqslant \delta_1 + \delta_2$. *On a :*

$$\mathcal{P}\left(\leqslant d_1 \ , \ \geqslant d_2 \mid w\right) = \tilde{\Theta}\left(\left(\frac{\left(1-\frac{1}{q-1}\right)^{\delta_1+\delta_2-2\alpha_0-\omega}\left(\frac{1}{q-1}\right)^{\delta_2-\alpha_0}}{q^{1-(1-\omega)h_q\left(\frac{\alpha_0}{1-\omega}\right)-\omega h_q\left(\frac{\delta_2-\alpha_0}{\omega}\right)-(\delta_2-\alpha_0)h_q\left(1+\frac{\delta_1-\alpha_0-\omega}{\delta_2-\alpha_0}\right)}}\right)^n\right) \tag{7.31}$$

où $\alpha_0 \in \left[\max(0, \delta_2 - \omega), \min\left(1-\omega, \frac{\delta_1+\delta_2-\omega}{2}\right)\right]$ *maximise l'expression.*

Démonstration du lemme 7.3.6.

Nous rappelons tout d'abord que :

$$\begin{aligned}\mathcal{P}\left(\leqslant d_1 \ , \ \geqslant d_2 \mid w\right) &:= \mathbb{P}\left(\mathbf{u} \in \mathcal{B}\left(\mathbf{x}, d_1\right) \cap \overline{\mathcal{B}}\left(\mathbf{y}, d_2\right)\right) \\ &= \mathbb{P}\left(\mathbf{u} \in \mathcal{B}\left(\mathbf{0}, d_1\right) \cap \overline{\mathcal{B}}\left(\mathbf{z}, d_2\right)\right) \\ &= \frac{\#\mathcal{B}\left(\mathbf{0}, d_1\right) \cap \overline{\mathcal{B}}\left(\mathbf{z}, d_2\right)}{\#\mathbb{F}_q^n}\end{aligned}$$

où $\mathbf{u}$ est choisi uniformément dans $\mathbb{F}_q^n$ et $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{F}_q^n$ sont tels que $\Delta\left(\mathbf{x}, \mathbf{y}\right) = |\mathbf{x}| = w$. Soit $\mathbf{c} \in \mathcal{B}\left(\mathbf{0}, d_1\right) \cap \overline{\mathcal{B}}\left(\mathbf{z}, d_2\right)$ et soient α, β et γ les trois quantités suivantes :

$$\begin{aligned}\alpha &:= \frac{\#\left(\operatorname{supp}\left(\mathbf{c}\right) \setminus \operatorname{supp}\left(\mathbf{z}\right)\right)}{n} \\ \beta &:= \frac{\#\left(\operatorname{supp}\left(\mathbf{c}-\mathbf{z}\right) \cap \operatorname{supp}\left(\mathbf{z}\right)\right)}{n} \\ \gamma &:= \frac{\#\left(\operatorname{supp}\left(\mathbf{c}\right) \cap \operatorname{supp}\left(\mathbf{z}\right) \cap \operatorname{supp}\left(\mathbf{c}-\mathbf{z}\right)\right)}{n}\end{aligned}$$

Nous représentons la situation à l'aide de la figure 7.10.

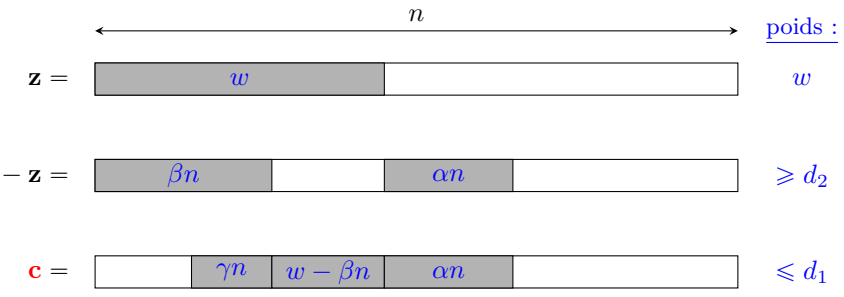


Figure 7.10 – Configuration du vecteur $\mathbf{c} \in \mathcal{B}\left(\mathbf{0}, d_1\right) \cap \overline{\mathcal{B}}\left(\mathbf{z}, d_2\right)$ à permutation près des positions.

Les poids relatifs α, β et γ sont nécessairement tels que $\Delta\left(\mathbf{c}, \mathbf{z}\right) = \alpha n + \beta n \geqslant d_2$ et $|\mathbf{c}| = w - \beta n + \alpha n + \gamma n \leqslant d_1$. Nous avons donc :

$$\#\mathcal{B}\left(\mathbf{0}, d_1\right) \cap \overline{\mathcal{B}}\left(\mathbf{z}, d_2\right) =$$

$$\sum_{\alpha n=0}^{d} \sum_{\beta n=d_2-\alpha n}^{n} \sum_{\gamma n=0}^{d_1+\beta n-w-\alpha n} \binom{n-w}{\alpha n}\binom{w}{\beta n}\binom{\beta n}{\gamma n}(q-2)^{\gamma n}(q-1)^{\alpha n}$$

Puisque $d_1 \leqslant n - \frac{n}{q} \leqslant d_2$, la somme précédente est dominée par les termes vérifiant $|\mathbf{c}| = d_1$ et $\Delta(\mathbf{c}, \mathbf{z}) = d_2$; c'est-à-dire les termes vérifiant $\beta n = d_2 - \alpha n$ et $\gamma n = d_1 + d_2 - 2\alpha n - w$. Nous avons ainsi :

$$\#\mathcal{B}(\mathbf{0}, d_1) \cap \overline{\mathcal{B}}(\mathbf{z}, d_2) =$$

$$\tilde{\Theta}\left(\sum_{\alpha n=0}^{d} \binom{n-w}{\alpha n}\binom{w}{d_2-\alpha n}\binom{d_2-\alpha n}{d_1+d_2-2\alpha n-w}(q-2)^{d_1+d_2-2\alpha n-w}(q-1)^{\alpha n}\right)$$

En nous restreignant exclusivement aux termes non-nuls, nous obtenons :

$$\#\mathcal{B}(\mathbf{0}, d_1) \cap \overline{\mathcal{B}}(\mathbf{z}, d_2) =$$

$$\tilde{\Theta}\left(\sum_{\alpha n\in\overline{A}} \binom{n-w}{\alpha n}\binom{w}{d_2-\alpha n}\binom{d_2-\alpha n}{d_1+d_2-2\alpha n-w}(q-2)^{d_1+d_2-2\alpha n-w}(q-1)^{\alpha n}\right)$$

où $\overline{A} := [\![0, n-w]\!] \cap [\![d_2 - w, d_2]\!] \cap [\![d_1 - w, \frac{d_1+d_2-w}{2}]\!]$. Nous avons alors :

$$\#\mathcal{B}(\mathbf{0}, d) \cap \mathcal{B}(\mathbf{z}, d) =$$
$$\tilde{\Theta}\left(\sup_{\alpha\in A}\left(q^{(1-\omega)h_q\left(\frac{\alpha}{1-\omega}\right)+\omega h_q\left(\frac{\delta_2-\alpha}{\omega}\right)+(\delta_2-\alpha)h_q\left(1+\frac{\delta_1-\alpha-\omega}{\delta_2-\alpha}\right)}\cdot\frac{(q-2)^{\delta_1+\delta_2-2\alpha-\omega}}{(q-1)^{\delta_1+2\delta_2-3\alpha-\omega}}\right)^n\right)$$

où $A := \left[\max(0, \delta_2 - \omega), \min\left(1-\omega, \frac{\delta_1+\delta_2-\omega}{2}\right)\right]$. Notons que A n'est pas vide lorsque l'hypothèse $\delta_2 - \delta_1 \leqslant \omega \leqslant \delta_1 + \delta_2$ est vérifiée.
Nous terminons la preuve en remarquant que $\#\mathbb{F}_q^n = q^n$. □

À propos de la valeur de α_0 dans le lemme 7.3.6. Nous remarquons tout d'abord que si $\delta_2 = 1$ alors nous avons nécessairement $\alpha_0 = 1 - \omega$. Et donc :

$$\mathcal{P}(\leqslant d_1\ ,\ \geqslant n \mid w) = \tilde{\Theta}\left(\left(\frac{(q-2)^{\delta_1-1+\omega}(q-1)^{2-2\omega-\delta_1}}{q^{1-\omega h_q\left(1-\frac{1-\delta_1}{\omega}\right)}}\right)^n\right) \tag{7.32}$$

En revanche, si $\delta_2 < 1$ alors, de la même façon que pour le lemme 7.3.3, nous pouvons montrer que α_0 est l'unique racine réelle dans A de l'équation suivante :

$$(q-1)(1-\omega-\alpha)(\delta_1+\delta_2-2\alpha-\omega)^2 = \alpha(q-2)^2(\omega-\delta_1+\alpha)(\omega-\delta_2+\alpha) \tag{7.33}$$

Nous pouvons finalement utiliser la méthode de Cardan pour déterminer une formule close pour α_0.

Corollaire 7.3.7 (du théorème 6.3.6)**.** *Supposons que pour tout code aléatoire $\mathcal{C} \subseteq \mathbb{F}_q^n$ de taille T et pour tout couple de distance $(d_1, d_2) \in \left[\!\left[0, n-\frac{n}{q}\right]\!\right] \times \left[\!\left[n-\frac{n}{q}, n\right]\!\right]$, il existe un décodeur et un anti-decodeur en liste idéaux qui s'exécutent respectivement en un temps de l'ordre de $O(\max(1, T\cdot\mathcal{P}(\leqslant d_1)))$ et $O(\max(1, T\cdot\mathcal{P}(\geqslant d_2)))$. L'algorithme 6.3.1 peut alors résoudre le problème des lointaines-collisions sur $\mathbb{F}_q^n$ en un temps de l'ordre de :*

$$T_{\text{Codes}}^{\text{Aléa}} = \begin{cases} \tilde{O}\left(\dfrac{\max\left(L\left(1-\frac{1}{q}\right)^n, 1\right)}{\mathcal{P}\left(\leqslant d_{GV}^-(L)\ ,\ \geqslant d_{GV}^+(L) \mid w\right)}\right) & \text{si } \delta_{GV}^+(L) < \omega \\ & \text{ou } 0 \leqslant \dfrac{\omega - \delta_{GV}^+(L)}{1 - \frac{q}{q-1}\cdot\delta_{GV}^+(L)} \leqslant d_{GV}^-(L) \\ \tilde{O}\left(L^2\mathcal{P}(\geqslant w)\right) & \text{sinon} \end{cases} \tag{7.34}$$

où $L := q^{\lambda n}$ *et* $w := \lceil \omega n \rceil$ *sont les paramètres du problème des lointaines-collisions pour des constantes* $\lambda \in [0,1]$ *et* $\omega \in \left[0, \frac{1}{2}\right]$ *et où :*

$$d_{GV}^{-}(L) = \left\lceil \delta_{GV}^{-}(L) n \right\rceil \qquad \text{avec} \qquad \delta_{GV}^{-}(L) := g_q^{-}\left(1-\lambda\right) \tag{7.35}$$

$$d_{GV}^{+}(L) = \left\lfloor \delta_{GV}^{+}(L) n \right\rfloor \qquad \text{avec} \qquad \delta_{GV}^{+}(L) := g_q^{+}\left(\max\left(\log_q(q-1), 1-\lambda\right)\right) \tag{7.36}$$

Démonstration du corollaire 7.3.7.

Les hypothèses du lemme 6.3.4 sont bien vérifiées dans $\mathbb{F}_q^n$. Nous pouvons donc appliquer le théorème 6.3.6. Pour cela, nous commençons par utiliser les lemmes 7.3.2 et 7.3.5 pour montrer les équations (7.35) et (7.36).

D'autre part, soit $\mathbf{x}$ tiré uniformément dans $\mathcal{B}(\mathbf{0}, d_1)$ avec $d_1 \in \left[\!\left[0, n - \frac{n}{q}\right]\!\right]$ et soit $\mathbf{y}$ tiré uniformément dans $\overline{\mathcal{B}}(\mathbf{0}, d_2)$ avec $d_2 := d_{\mathrm{GV}}^{+}(L) \in \left[\!\left[n - \frac{n}{q}, n\right]\!\right]$. Nous savons que $\mathbf{x}$ et $\mathbf{y}$ sont typiquement de poids de Hamming respectifs d_1 et d_2. De plus, en moyenne, le support de $\mathbf{y}$ est équitablement réparti sur le support de $\mathbf{x}$ et son complémentaire donc la taille moyenne de $\mathrm{supp}(\mathbf{x}) \cap \mathrm{supp}(\mathbf{y})$ est $\frac{d_1 d_2}{n}$. En outre, pour tout $i \in \mathrm{supp}(\mathbf{x}) \cap \mathrm{supp}(\mathbf{y})$, la probabilité $\mathbb{P}(x_i \neq y_i) = \frac{q-2}{q-1}$ où $\mathbf{x} := (x_1, \cdots, x_n)$ et $\mathbf{y} := (y_1, \cdots, y_n)$. Nous résumons la situation avec la figure 7.11.

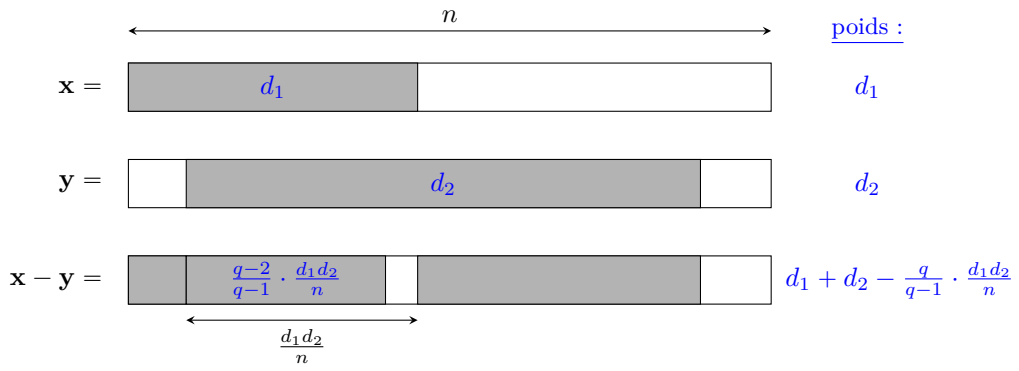


Figure 7.11 – Configuration typique de $\mathbf{x}$ et $\mathbf{y}$ à permutation près des positions.

Nous avons ainsi $\mathbb{E}(\Delta(\mathbf{x}, \mathbf{y})) = d_1 + d_2 - \frac{q}{q-1} \cdot \frac{d_1 d_2}{n}$ et donc finalement, lorsque $d_2 \geqslant w$, on a :

$$d_1^w := \sup\left\{d_1 \in [\![d_2 - w, n]\!] \ : \ \mathbb{E}(\Delta(\mathbf{x}, \mathbf{y})) \geqslant w\right\}$$

Nous rappelons alors que $d_2 := d_{\mathrm{GV}}^{+}(L) := \left\lfloor \delta_{\mathrm{GV}}^{+}(L) n \right\rfloor$; d'où :

$$d_1^w = \lceil \delta_1^w n \rceil \qquad \text{avec} \qquad \delta_1^w := \frac{\omega - \delta_{\mathrm{GV}}^{+}(L)}{1 - \frac{q}{q-1} \cdot \delta_{\mathrm{GV}}^{+}(L)}$$

□

On peut montrer que la complexité donnée par le corollaire 7.3.7 est optimale. Pour cela, on pose les fonctions f, g et h suivantes :

$$f(\delta_1, \delta_2) := \frac{L \cdot \mathcal{P}(\leqslant \lceil \delta_1 n \rceil)}{\mathcal{P}(\leqslant \lceil \delta_1 n \rceil \ , \ \geqslant \lfloor \delta_2 n \rfloor \mid w)} \tag{7.37}$$

$$g(\delta_1, \delta_2) := \frac{L \cdot \mathcal{P}(\geqslant \lfloor \delta_2 n \rfloor)}{\mathcal{P}(\leqslant \lceil \delta_1 n \rceil \ , \ \geqslant \lfloor \delta_2 n \rfloor \mid w)} \tag{7.38}$$

$$h(\delta_1, \delta_2) := \frac{L^2 \cdot \mathcal{P}(\leqslant \lceil \delta_1 n \rceil) \cdot \mathcal{P}(\geqslant \lfloor \delta_2 n \rfloor)}{\mathcal{P}(\leqslant \lceil \delta_1 n \rceil \ , \ \geqslant \lfloor \delta_2 n \rfloor \mid w)} \tag{7.39}$$

La complexité de la méthode est alors de l'ordre de :

$$T^{\text{Aléa}}_{\text{Codes}}(d_1, d_2) = O(\max(f(d_1, d_2), g(d_1, d_2), h(d_1, d_2))) \tag{7.40}$$

Une analyse différentielle de f et g montre que $f(\delta_1, \cdot)$ est croissante sur $\left[1 - \frac{1}{q}, 1\right]$ et que $g(\cdot, \delta_2)$ est décroissante sur $\left[0, 1 - \frac{1}{q}\right]$. De plus, on a trivialement $h(\delta_1, \delta_2) \geqslant \max(f(\delta_1, \delta_2), g(\delta_1, \delta_2))$ pour tout $(\delta_1, \delta_2) \in \left[\delta^-_{\text{GV}}(L), 1 - \frac{1}{q}\right] \times \left[1 - \frac{1}{q}, \delta^+_{\text{GV}}(L)\right]$ (par définition des distances de Gilbert-Varshamov inférieure et supérieure). Ainsi, $T^{\text{Aléa}}_{\text{Codes}}(d_1, d_2)$ atteint son minimum dans $\left[\delta^-_{\text{GV}}(L), 1 - \frac{1}{q}\right] \times \left[1 - \frac{1}{q}, \delta^+_{\text{GV}}(L)\right]$. De plus, sur cet espace des paramètres, la complexité est de l'ordre de $O(h(\delta_1, \delta_2))$.

Une analyse des dérivées partielles de h montre que s'il existe un couple $(\delta_1, \delta_2) \in \left[\delta^-_{\text{GV}}(L), 1 - \frac{1}{q}\right] \times \left[1 - \frac{1}{q}, \delta^+_{\text{GV}}(L)\right]$ tel que :

$$\delta_1 = \frac{\omega - \delta_2}{1 - \frac{q}{q-1} \cdot \delta_2} \tag{7.41}$$

alors le minimum de h est atteint en ce couple ; sinon, il est atteint en $(\delta^-_{\text{GV}}(L), \delta^+_{\text{GV}}(L))$.

Pour finir, on remarque que s'il existe un couple $(\delta_1, \delta_2) \in \left[\delta^-_{\text{GV}}(L), 1 - \frac{1}{q}\right] \times \left[1 - \frac{1}{q}, \delta^+_{\text{GV}}(L)\right]$ qui vérifie l'équation (7.41), alors il en existe particulièrement un qui vérifie de surcroit $\delta_2 = \delta^+_{\text{GV}}(L)$.

Finalement, la figure 7.12 représente la complexité optimale de notre méthode pour trouver les lointaines-collisions dans une liste d'éléments d'un espace de Hamming q-aire.

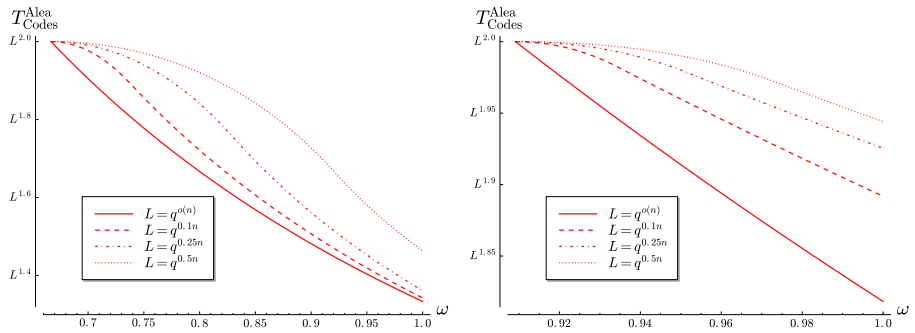


Figure 7.12 – Complexité de notre méthode pour la recherche de lointaine-collisions dans les espaces q-aire de Hamming. À gauche $q = 3$ et à droite $q = 11$.

Remarque 7.3.3. Lorsque L est sous-exponentiel en n, la complexité de notre méthode est $O\left(L^{\frac{1}{1 - \frac{qw}{2(q-1)}}}\right)$.

Chapitre 8

Des approches dérivées

Dans ce chapitre, nous proposons différentes variations de nos méthodes de recherche de presque/lointaines-collisions présentées dans les deux chapitres précédents. Toutefois, il faut noter que seule la méthode présentée dans la section 8.1 apporte une amélioration. Les autres approches sont des pistes que nous avons explorées et que nous avons jugées intéressantes d'exposer.

Comme l'illustre la figure 8.1, il arrive parfois que la méthode des projections (cf. section 5.4.2) soit plus performante que notre méthode des codes (cf. sous-section 7.3.1) pour rechercher des presque-collisions sur $\mathbb{F}_q^n$.

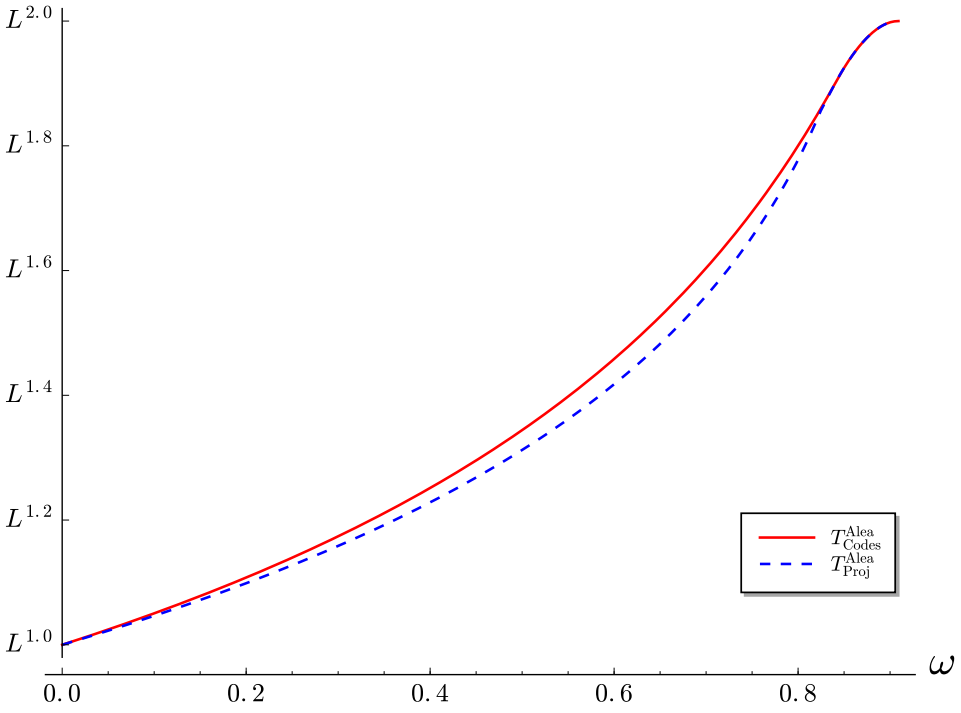


Figure 8.1 – Comparaison de la méthode des codes avec celle des projections pour une instance particulière où la méthode des projections est plus efficace ($q = 11$ et $L = q^{0.1n}$).

Nous avons alors cherché à construire un algorithme hybride qui interpole les deux méthodes. Cette idée a donnée naissance aux méthodes décrites dans la section 8.1 et la sous-section 8.2.1. Le premier algorithme hybride améliore notre méthode originale ainsi que la méthode LSH des projections. En revanche, le second algorithme hybride n'apporte pas d'amélioration puisque nous verrons qu'il revient essentiellement à choisir la meilleure des deux méthodes sus-cités.

Dans la section 8.2, nous proposons de résoudre le problème des presque-collisions de façon récursive : la méthode des codes est appliquée à un premier niveau pour ranger les éléments de la liste par paquets grossiers. Le second niveau de recherche consiste à traiter chaque paquet indépendamment comme un nouveau problème de recherche de presque-collisions. Notons que cette nouvelle méthode est en fait une généralisation de notre seconde méthode hybride décrite dans la sous-section 8.2.1. Malheureusement, elle n'aboutit à aucune amélioration.

La méthode récursive de recherche de presque-collisions nous a amené à étudier le problème des presque-collisions dans une boule de $\mathbb{F}_q^n$. Nous avons ainsi pu appliquer nos résultats du chapitre 6 à un nouvel espace.

Enfin, dans la dernière section, nous proposons une approche par changement de géométrie : nous plongeons les vecteurs de la liste $\mathscr{L}$ dans un nouvel espace. Nous espérons alors que dans l'espace d'arrivée, la probabilité de collision des éléments proches soit augmentée. Notons que nous avons également exploité cette idée dans la section 8.1. En effet, notre premier algorithme hybride consiste en fait à projeter les éléments de $\mathscr{L}$ dans un espace de plus petite dimension en espérant que dans ce nouvel espace, les éléments proches entrent en collision plus facilement. Dans la section 8.3, nous envoyons les éléments de $\mathscr{L}$ dans un espace de plus grande dimension en prenant soin de conserver les distances relatives entre les éléments. Pour ce faire, nous *répliquons* les vecteurs de $\mathscr{L}$; c'est-à-dire que nous remplaçons chaque $\mathbf{x} \in \mathscr{L}$ par un nombre fixé de concaténations de $\mathbf{x}$ avec lui-même. En procédant ainsi, nous conservons bien les distances relatives entres les éléments de $\mathscr{L}$ et nous conservons également la taille de $\mathscr{L}$; en revanche, nous augmentons la dimension de l'espace ambiant. Nous verrons l'impact que cela a sur la probabilité de collision des éléments proches et pourquoi nous avons espéré qu'en procédant ainsi, nous pourrions améliorer la recherche de presque-collisions. Cependant, nous verrons aussi que notre changement de géométrie ne disperse pas les mots de $\mathscr{L}$ de manière uniforme dans le nouvel espace ambiant ; ce qui nous fait perdre d'un coté, le gain que nous avions obtenu de l'autre.

8.1 Une première approche hybride

Dans cette section, nous proposons de résoudre le problème des presque-collisions avec un algorithme qui interpole notre méthode des codes et la méthode LSH des projections. Celui-ci consiste à appliquer notre méthode des codes sur une liste $\mathscr{L}'$ qui est simplement une projection de la liste originale $\mathscr{L}$; c'est-à-dire que chaque élément de $\mathscr{L}'$ est le projeté sur un ensemble de positions I d'un élément de $\mathscr{L}$. En procédant ainsi, nous espérons diminuer la distance relative entre deux éléments proches et ainsi augmenter leur probabilité de collision.

Nous décrivons plus formellement notre méthode hybride avec le pseudo-code 8.1.1 .

Algorithme 8.1.1 : Algorithme hybride

Entrées : une liste $\mathscr{L} \subseteq \mathbb{F}_q^n$;
une distance $w \in \left[\!\left[0, n - \frac{n}{q}\right]\!\right]$;
Paramètres : un nombre d'itération s ;
un entier $k \in [\![1, n]\!]$;
une distance $w' \in [\![0, w]\!]$;
Sortie : $\mathscr{L}^* = \left\{(\mathbf{x}, \mathbf{y}) \in \mathscr{L}^2 \ : \ \Delta(\mathbf{x}, \mathbf{y}) \leqslant w\right\}$.

1 $\mathscr{L}^* \leftarrow \varnothing$;
2 **répéter** s **fois**
3 choisir I uniformément dans $\{A \subseteq [\![1, n]\!] : \#A = k\}$;
4 $\mathscr{L}_1^* \leftarrow \left\{(\mathbf{x}, \mathbf{y}) \in \mathscr{L}^2 : \Delta(\mathbf{x}_I, \mathbf{y}_I) \leqslant w'\right\}$; `/* cf. note 8.1.1 */`
5 $\mathscr{L}_2^* \leftarrow \{(\mathbf{x}, \mathbf{y}) \in \mathscr{L}_1^* : \Delta(\mathbf{x}, \mathbf{y}) \leqslant w\}$;
6 $\mathscr{L}^* \leftarrow \mathscr{L}^* \cup \mathscr{L}_2^*$;
7 **finRépéter**
8 **retourner** $\mathscr{L}^*$;

Note 8.1.1. Pour toute itération de l'algorithme 8.1.1, la liste $\mathscr{L}_1^*$ est produite en effectuant une recherche de presque-collisions sur la liste $\mathscr{L}' := \left\{\mathbf{x}_I \in \mathbb{F}_q^k \ : \ \mathbf{x} \in \mathscr{L}\right\}$. Nous effectuons alors cette recherche de presque-collisions à l'aide de notre méthode des codes.

Le théorème suivant donne la complexité de l'algorithme hybride 8.1.1 :

Théorème 8.1.1. *Soient* $L := \#\mathscr{L} = q^{\lambda n}$ *et* $w = \lfloor \omega n \rfloor$ *les paramètres d'un problème de presque-collisions pour des constantes* $\lambda \in [0, 1]$ *et* $\omega \in \left[0, 1 - \frac{1}{q}\right]$. *Pour tout* $k \in [\![1, \lambda n]\!]$ *et tout* $w' \in [\![0, w]\!]$, *l'algorithme hybride 8.1.1 peut résoudre le problème des presque-collisions en un temps de l'ordre de :*

$$T_{\text{Hybride}}^{\text{Aléa}}(k, w') = O\left(\frac{\binom{n}{w} \cdot T_{\text{Codes}}^{\text{Aléa}}\left(q, k, q^{\lambda n}, w'\right)}{\binom{k}{w'}\binom{n-k}{w-w'}}\right) \tag{8.1}$$

où $T_{\text{Codes}}^{\text{Aléa}}\left(q, k, q^{\lambda n}, w'\right)$ *est le coût de la méthode des codes pour la recherche de* w'*-presque-collisions dans une liste* $\mathscr{L}' \subseteq \mathbb{F}_q^k$ *de taille* $q^{\lambda n}$. *La valeur de* $T_{\text{Codes}}^{\text{Aléa}}\left(q, k, q^{\lambda n}, w'\right)$ *est donnée par le corollaire 7.3.4.*

Démonstration du théorème 8.1.1.

La probabilité de succès d'une itération de l'algorithme hybride 8.1.1 est :

$$\begin{aligned}
\mathbb{P}_{\text{succ}} &:= \mathbb{P}\left(|\mathbf{x}_I| \leqslant w' \ \middle| \ |\mathbf{x}| \leqslant w\right) \\
&= \Theta\left(\mathbb{P}\left(|\mathbf{x}_I| = w' \ \middle| \ |\mathbf{x}| = w\right)\right) \\
&= \Theta\left(\frac{\binom{k}{w'}\binom{n-k}{w-w'}}{\binom{n}{w}}\right)
\end{aligned}$$

On choisit alors $s := \frac{1}{\mathbb{P}_{\text{succ}}}$ pour que l'espérance du nombre de solutions retournées soit de l'ordre du nombre de solutions recherchées. De plus, lors d'une itération, le coût de la recherche de presque-collisions domine toujours le coût du filtrage (étape **5** dans l'algorithme 8.1.1) car cette étape coûte le nombre de solutions du problème de presque-collisions. Ainsi, le coût d'une itération est de l'ordre de :

$$O\left(T_{\text{Codes}}^{\text{Aléa}}\left(q, k, q^{\lambda n}, w'\right)\right)$$

□

L'algorithme hybride 8.1.1 est nécessairement au moins aussi performant que la méthode des projections et celle des codes puisque chacune de ces deux méthodes est une instance de cet algorithme. En effet, lorsque l'on choisit $k = n$ et $w' = w$ on retrouve la méthode des codes et lorsque l'on choisit $k = \lfloor \lambda n \rfloor$ et $w' = 0$ on obtient essentiellement la méthode des projections. Nous espérons alors qu'il existe un jeu de paramètre (k, w') qui nous donne un algorithme plus performant que la méthode des codes et la méthode des projections. La figure 8.2 nous montre que cela est effectivement possible.

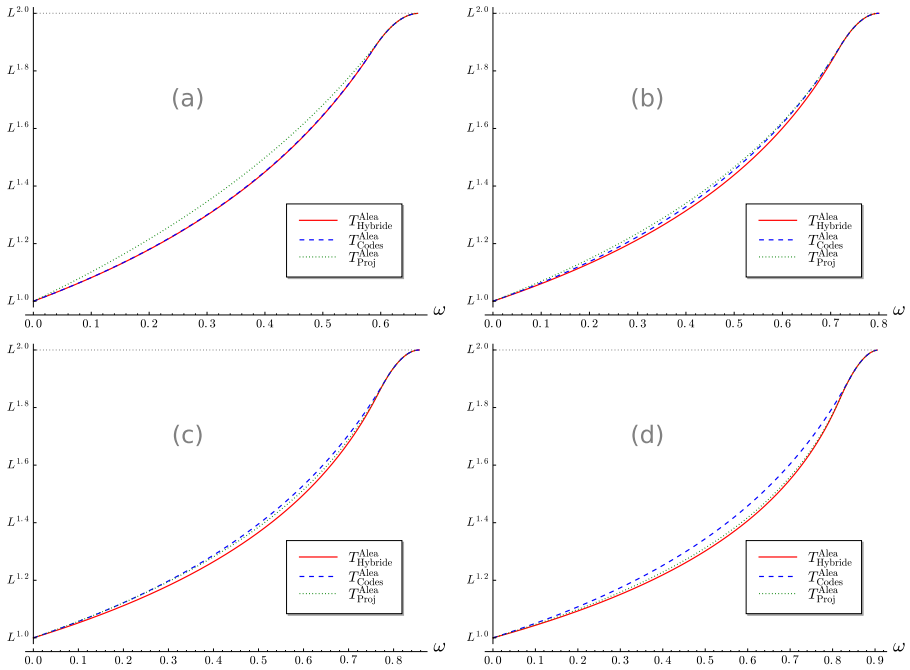


Figure 8.2 – Comparaison de la méthode hybride avec la méthode des projections et celle des codes ($L = q^{0.1n}$). (a) $q = 3$, (b) $q = 5$, (c) $q = 7$ et (d) $q = 11$.

L'algorithme hybride pour la recherche de lointaine-collisions. L'algorithme hybride 8.1.1 s'adapte à la recherche de lointaines-collisions. Il suffit pour cela de choisir le paramètre w' dans les grandes distances $[\![w, n]\!]$ et de changer le sens des inégalités dans l'algorithme

8.1.1. On obtient alors l'algorithme hybride suivant :

Algorithme 8.1.2 : Algorithme hybride pour la recherche de lointaine-collisions

Entrées : une liste $\mathscr{L} \subseteq \mathbb{F}_q^n$;
une distance $w \in \left[\!\left[n - \frac{n}{q}, n\right]\!\right]$;
Paramètres : un nombre d'itérations s ;
un entier $k \in [\![1, n]\!]$;
une distance $w' \in [\![w, n]\!]$;
Sortie : $\mathscr{L}^* = \left\{(\mathbf{x}, \mathbf{y}) \in \mathscr{L}^2 \ : \ \Delta(\mathbf{x}, \mathbf{y}) \geqslant w\right\}$.

1 $\mathscr{L}^* \leftarrow \varnothing$;
2 **répéter** s **fois**
3 choisir I uniformément dans $\{A \subseteq [\![1, n]\!] : \#A = k\}$;
4 $\mathscr{L}_1^* \leftarrow \left\{(\mathbf{x}, \mathbf{y}) \in \mathscr{L}^2 : \Delta(\mathbf{x}_I, \mathbf{y}_I) \geqslant w'\right\}$; `/* cf. note 8.1.2 */`
5 $\mathscr{L}_2^* \leftarrow \{(\mathbf{x}, \mathbf{y}) \in \mathscr{L}_1^* : \Delta(\mathbf{x}, \mathbf{y}) \geqslant w\}$;
6 $\mathscr{L}^* \leftarrow \mathscr{L}^* \cup \mathscr{L}_2^*$;
7 **finRépéter**
8 **retourner** $\mathscr{L}^*$;

Note 8.1.2. Cette fois-ci, nous devons effectuer une recherche de lointaine-collisions à l'étape 4 de l'algorithme 8.1.2. Pour cela, nous faisons appel à notre méthode des codes utilisant à la fois des décodeurs mais aussi des anti-décodeurs.

La démonstration du théorème suivant est analogue à celle du théorème 8.1.1.

Théorème 8.1.2. *Soient $L := \#\mathscr{L} = q^{\lambda n}$ et $w = \lfloor \omega n \rfloor$ les paramètres d'un problème de lointaines-collisions pour des constantes $\lambda \in [0, 1]$ et $\omega \in \left[1 - \frac{1}{q}, 1\right]$. Pour tout $k \in [\![1, \lambda n]\!]$ et tout $w' \in [\![w, n]\!]$, l'algorithme hybride 8.1.2 peut résoudre le problème des lointaines-collisions en un temps de l'ordre de :*

$$T_{\text{Hybride}}^{\text{Aléa}}(k, w') = O\left(\frac{\binom{n}{w} \cdot T_{\text{Codes}}^{\text{Aléa}}(q, k, q^{\lambda n}, w')}{\binom{k}{w'}\binom{n-k}{w-w'}}\right) \tag{8.2}$$

où $T_{\text{Codes}}^{\text{Aléa}}(q, k, q^{\lambda n}, w')$ est le coût de la méthode des codes (avec décodeurs et anti-décodeurs) pour la recherche de w'-lointaines-collisions dans une liste $\mathscr{L}' \subseteq \mathbb{F}_q^k$ de taille $q^{\lambda n}$. La valeur de $T_{\text{Codes}}^{\text{Aléa}}(q, k, q^{\lambda n}, w')$ est donnée par le corollaire 7.3.7.

La méthode des projections ne concerne que la recherche de presque-collisions. Nous ne comparons donc notre algorithme hybride 8.1.2 qu'avec la méthode des codes. Nous savons que nous ne pouvons qu'être plus performant puisque la méthode des codes est une instance de notre algorithme hybride ; nous espérons cependant faire mieux. Malheureusement, l'algorithme hybride 8.1.2 est optimal lorsque $k = n$ et $w' = w$; ce qui correspond exactement à l'instance de la méthode des codes.

8.2 Une approche récursive pour la recherche de presque-collisions

8.2.1 Une seconde approche hybride

Nous pouvons imaginer une autre façon de combiner la méthode des codes et celle des projections. Rappelons toutefois que la méthode que nous décrivons dans cette section

n'améliore pas la recherche de presque-collisions.

Notre second algorithme hybride se décompose en deux temps : nous remplissons tout d'abord une table de hachage chainée grâce à la méthode des projections puis nous appliquons la méthode des codes sur chacune des listes contenues dans chacune des cellules de la table de hachage. En fait, nous rangeons dans un premier temps les éléments de la liste $\mathscr{L}$ dans des paquets grossiers avec la projection puis nous utilisons la méthode des codes pour chercher des presque-collisions dans chacun de ces paquets.

L'algorithme 8.2.1 donne un pseudo-code de notre second algorithme hybride pour trouver les presque-collisions dans une liste de $\mathbb{F}_q^n$.

Algorithme 8.2.1 : Algorithme hybride version 2

Entrées : une liste $\mathscr{L} \subseteq \mathbb{F}_q^n$;
une distance $w \in \left[\!\left[0, n - \frac{n}{q}\right]\!\right]$;
Paramètres : un nombre d'itérations s ;
un entier $k \in [\![1, n]\!]$;
Sortie : $\mathscr{L}^* = \left\{(\mathbf{x}, \mathbf{y}) \in \mathscr{L}^2 \ : \ \Delta(\mathbf{x}, \mathbf{y}) \leqslant w\right\}$.

$\mathscr{L}^* \leftarrow \varnothing$;
répéter s **fois**
  initialiser une table de hachage chainée $\mathscr{T}$ de taille q^k ;
  choisir I uniformément dans $\{A \subseteq [\![1, n]\!] : \#A = k\}$;
  pour tout $\mathbf{x} \in \mathscr{L}$ **faire**
    ajouter $\mathbf{x}$ dans la liste $\mathscr{T}[\mathbf{x}_I]$; `/* par abus de notation, on confond ici` $\mathbf{x}_I$ `et l'entier dont il est l'écriture en base` q `*/`
  finPour
  pour tout $i \in \left[\!\left[1, q^k\right]\!\right]$ **faire**
    $\mathscr{L}_i^* \leftarrow \left\{(\mathbf{x}, \mathbf{y}) \in \mathscr{T}[i]^2 : \Delta(\mathbf{x}, \mathbf{y}) \leqslant w\right\}$; `/* cf. note 8.2.1 */`
    $\mathscr{L}^* \leftarrow \mathscr{L}^* \cup \mathscr{L}_i^*$;
  finPour
finRépéter
retourner $\mathscr{L}^*$;

Note 8.2.1. Pour tout $i \in \left[\!\left[1, q^k\right]\!\right]$, la liste $\mathscr{T}[i]$ contient des vecteurs de $\mathbb{F}_q^n$ dont k positions sont fixées ; les $n-k$ autres positions ont quant à elles des valeurs aléatoires. Ainsi, produire la liste $\mathscr{L}_i^*$ revient à rechercher tous les couples de vecteurs à distance au plus w dans la liste $\mathscr{T}[i]\big|_J := \left\{\mathbf{x}_J \subseteq \mathbb{F}_q^{n-k} \ : \ \mathbf{x} \in \mathscr{T}[i]\right\}$ où $J := [\![1, n]\!] \setminus I$. Nous effectuons cette recherche de presque-collisions à l'aide de notre méthode des codes.

L'algorithme hybride 8.2.1 s'analyse simplement en réutilisant les résultats déjà obtenus pour la méthode des projections et celle des codes :

Théorème 8.2.1. *Soient $L := \#\mathscr{L} = q^{\lambda n}$ et $w = \lfloor \omega n \rfloor$ les paramètres d'un problème de presque-collisions pour des constantes $\lambda \in [0, 1]$ et $\omega \in \left[0, 1 - \frac{1}{q}\right]$. Pour tout $k \in [\![1, \lambda n]\!]$, l'algorithme hybride 8.2.1 peut résoudre ce problème en un temps de l'ordre de :*

$$T_{\text{Hybride2}}^{\text{Aléa}}(k) = O\left(\frac{q^k \cdot \binom{n}{k} \cdot T_{\text{Codes}}^{\text{Aléa}}\left(q, n-k, q^{\lambda n - k}, w\right)}{\binom{n-w}{k}}\right) \tag{8.3}$$

où $T_{\text{Codes}}^{\text{Aléa}}\left(q, n-k, q^{\lambda n-k}, w\right)$ est le coût de la méthode des codes pour la recherche de w-presque-collisions dans une liste $\mathscr{L}' \subseteq \mathbb{F}_q^{n-k}$ de taille $q^{\lambda n-k}$. La valeur de $T_{\text{Codes}}^{\text{Aléa}}\left(q, n-k, q^{\lambda n-k}, w\right)$ est donnée par le corollaire 7.3.4.

Démonstration du théorème 8.2.1.

La probabilité de succès d'une itération de l'algorithme hybride 8.2.1 est :

$$\mathbb{P}_{\text{succ}} := \frac{\binom{n-w}{k}}{\binom{n}{k}}$$

On choisit alors $s := \frac{1}{\mathbb{P}_{\text{succ}}}$ pour que l'espérance du nombre de solutions retournées soit de l'ordre du nombre de solutions recherchées. De plus, le coût d'une itération est :

$$q^{\lambda n} + q^k T^{\text{Aléa}}_{\text{Codes}}\left(q, n-k, q^{\lambda n - k}, w\right)$$

Or $T^{\text{Aléa}}_{\text{Codes}}\left(q, n-k, q^{\lambda n-k}, w\right) \in \left[q^{\lambda n-k}, q^{2(\lambda n-k)}\right]$ donc le coût d'une itération est de l'ordre de :

$$O\left(q^k T^{\text{Aléa}}_{\text{Codes}}\left(q, n-k, q^{\lambda n-k}, w\right)\right)$$

□

La méthode des projections et celle des codes sont des instances de l'algorithme hybride 8.2.1. En effet, en choisissant $k = \lfloor \lambda n \rfloor$ on retrouve la méthode des projections tandis que lorsque $k = 0$ on retrouve essentiellement la méthode des codes. Nous espérons alors qu'il existe $k \in [\![0, \lfloor \lambda n \rfloor]\!]$ qui soit un meilleur choix que les deux valeurs précédentes. La figure 8.3 montre trois exemples d'évolution de la complexité de l'algorithme 8.2.1 en fonction de k. Sur ces cas particuliers, on remarque que la valeur de k optimale est soit 1, soit λn. Une analyse différentielle d'une version asymptotique de l'équation (8.3) (donnée par la formule de Stirling) montre que ce résultat est général. Ainsi, l'algorithme hybride 8.2.1 n'apporte aucune amélioration puisque l'optimiser revient essentiellement à prendre la meilleure méthode entre celle des projections et celle des codes.

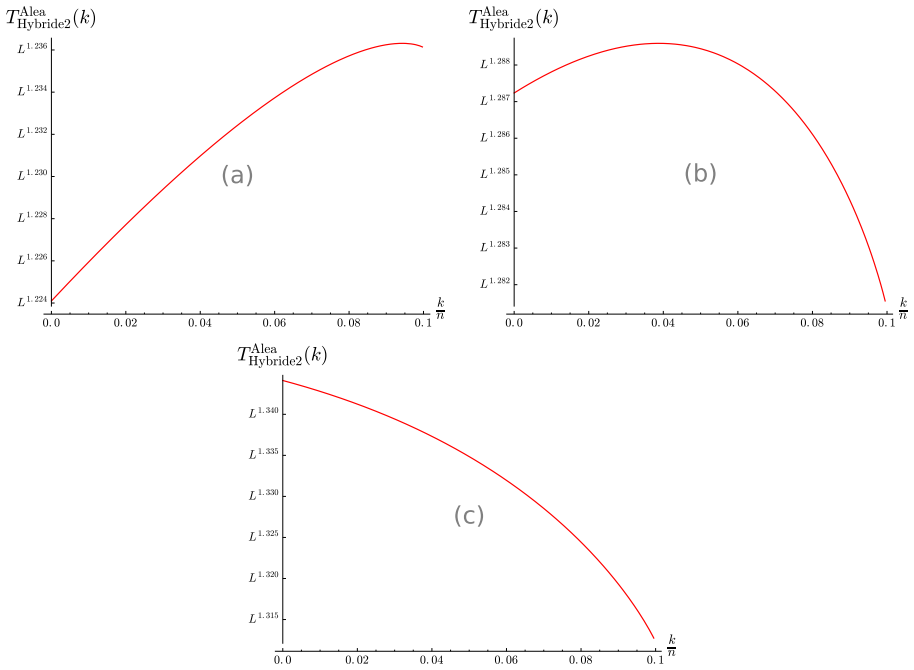


Figure 8.3 – Complexité de l'algorithme hybride 8.2.1 en fonction de $\frac{k}{n}$. (a) $q = 5, L = q^{0.1n}, \omega = 0.3$, (b) $q = 7, L = q^{0.1n}, \omega = 0.4$ et (c) $q = 11, L = q^{0.1n}, \omega = 0.5$.

8.2.2 Une généralisation de l'approche récursive sur deux niveaux

L'approche hybride de la sous-section 8.2.1 nous a inspiré l'algorithme que nous décrivons dans cette sous-section. En effet, celui-ci est une généralisation de l'algorithme 8.2.1.

Replaçons-nous dans le cas générique du problème des presque-collisions où l'espace ambiant est un espace métrique $(\mathscr{E}, \Delta)$ probabilisé avec la probabilité uniforme. Nous supposons que pour tout $\mathbf{x}, \mathbf{x}' \in \mathscr{E}$ et pour toute distance d, on a $\mathbb{P}(\mathbf{u} \in \mathcal{B}(\mathbf{x}, d)) = \mathbb{P}(\mathbf{u} \in \mathcal{B}(\mathbf{x}', d))$ où $\mathbf{u}$ est tiré uniformément dans $\mathscr{E}$. Dans l'algorithme 6.1.1, lorsqu'une adresse contient plus de 2 éléments, nous pouvons appliquer une autre recherche de presque-collisions sur la liste contenue à cette adresse plutôt que d'effectuer une recherche exhaustive des couples proches. En fait, en procédant ainsi, nous pouvons construire un algorithme récursif pour résoudre le problème des presque-collisions. Cependant, pour simplifier l'analyse, nous allons nous contenter d'étudier le cas où la profondeur de la récursion est 2. Nous décrivons l'algorithme ainsi obtenu à l'aide du pseudo-code 8.2.2.

Algorithme 8.2.2 : Recherche de presque-collisions avec hachage en liste récursif.

Entrées : une liste $\mathscr{L} \subseteq \mathscr{E}$;
une distance admissible w ;
Paramètres : un nombre d'itérations s ;
un entier T ;
une famille de fonctions $\mathscr{F} \subseteq \{\mathscr{E} \to \mathscr{P}(\llbracket 1, T \rrbracket)\}$;
Sortie : $\mathscr{L}^* = \{(\mathbf{x}, \mathbf{y}) \in \mathscr{L}^2 \ : \ \Delta(\mathbf{x}, \mathbf{y}) \leqslant w\}$.

1 $\mathscr{L}^* \leftarrow \varnothing$;
2 **répéter** s **fois**
3 choisir h uniformément dans $\mathscr{F}$;
4 initialiser une table de hachage chainée $\mathscr{T}$ de taille T ;
5 **pour tout** $\mathbf{x} \in \mathscr{L}$ **faire**
6 **pour tout** $i \in h(\mathbf{x})$ **faire**
7 ajouter $\mathbf{x}$ dans la liste $\mathscr{T}[i]$;
8 **finPour**
9 **finPour**
10 **pour tout** $i \in \llbracket 1, T \rrbracket$ **faire**
11 $\mathscr{L}_i^* \leftarrow \{(\mathbf{x}, \mathbf{y}) \in \mathscr{T}[i]^2 : \Delta(\mathbf{x}, \mathbf{y}) \leqslant w\}$; `/* cf. note 8.2.2 */`
12 $\mathscr{L}^* \leftarrow \mathscr{L}^* \cup \mathscr{L}_i^*$;
13 **finPour**
14 **finRépéter**
15 **retourner** $\mathscr{L}^*$;

Note 8.2.2. Pour chaque itération de l'algorithme et pour tout $i \in \llbracket 1, T \rrbracket$, les éléments de $\mathscr{T}[i]$ sont uniformément répartis dans l'ensemble $\mathscr{E}_{h,i} := \{\mathbf{x} \in \mathscr{E} \ : \ i \in h(\mathbf{x})\}$. Ainsi, l'étape **11** de l'algorithme 8.2.2 est une recherche de presque-collisions dans une liste d'éléments tirés uniformément dans $\mathscr{E}_{h,i}$.

L'algorithme original 6.1.1 est une instance de cet algorithme où l'étape **11** est résolue avec une recherche exhaustive. En outre, nous avons proposé une autre instance de cet algorithme dans la sous-section 8.2.1. En effet, sur l'espace de Hamming $\mathbb{F}_q^n$, le premier algorithme hybride 8.2.1 est une instance de l'algorithme 8.2.2. Pour le remarquer, il suffit de choisir la famille $\mathscr{F}$ comme la famille $\mathscr{F}_{\mathrm{Proj}}(k)$ des projections sur k positions et de résoudre l'étape 11 avec la méthode des codes.

Par la suite, à chaque itération de l'algorithme 8.2.2, nous allons choisir la fonction de hachage floue $h := h_{\mathcal{C}}^d$ où $\mathcal{C}$ est un code aléatoire de taille T sur $\mathscr{E}$ et pour tout $\mathbf{x} \in \mathscr{E}$, $h_{\mathcal{C}}^d(\mathbf{x})$ est l'ensemble des mots du code $\mathcal{C}$ à distance au plus d de $\mathbf{x}$. Ainsi, à chaque itération et

pour tout $\mathbf{c} \in \mathcal{C}$, il nous faudra résoudre le problème des presque-collisions sur la boule $\mathcal{B}(\mathbf{c}, d)$. Sans perte de généralité, nous pouvons translater le problème sur la boule $\mathcal{B}(\mathbf{c}, d)$.

Nous utilisons les mêmes notations 6.2.1 du chapitre 6 pour énoncer le théorème suivant :

Théorème 8.2.2. *Nous supposons que pour tout code aléatoire $\mathcal{C}$ de taille T un décodeur en liste idéal calcule la fonction $h_{\mathcal{C}}^{d}$ en un temps de l'ordre de $O(\max(1, T \cdot \mathcal{P}(\leqslant d)))$. Ainsi, l'algorithme 8.2.2 peut résoudre le problème des presque-collisions en un temps de l'ordre de :*

$$T_{\text{Récursif}}^{\text{Aléa}}(d) = O\left(\frac{L \cdot \mathcal{P}(\leqslant d) + T_{\text{Codes}}^{\text{Aléa}}(\mathcal{B}(\mathbf{0}, d), L \cdot \mathcal{P}(\leqslant d), w)}{\mathcal{P}(\leqslant d \mid w)}\right) \tag{8.4}$$

où $T_{\text{Codes}}^{\text{Aléa}}(\mathcal{B}(\mathbf{0}, d), L \cdot \mathcal{P}(\leqslant d), w)$ est le coût de la méthode des codes pour résoudre le problème des w-presque-collisions dans une liste de $L \cdot \mathcal{P}(\leqslant d)$ éléments tirés uniformément dans $\mathcal{B}(\mathbf{0}, d)$.

8.2.3 Le problème des presque-collisions sur une boule de $\mathbb{F}_2^n$

À ce stade, il nous reste à analyser notre méthode des codes dans le modèle particulier où $\mathscr{L}$ est une liste d'éléments tirés uniformément dans *une boule de* $\mathscr{E}$. Ceci est l'objet de cette sous-section ; cependant, nous nous concentrons ici exclusivement sur les espaces binaires de Hamming.

Dans cette sous-section, Δ et $|\cdot|$ dénotent la distance et le poids de Hamming et $\mathcal{B}(\mathbf{c}, r)$ et $\mathcal{S}(\mathbf{c}, r)$ sont respectivement la boule et la sphère de centre $\mathbf{c}$ et de rayon r sur $\mathbb{F}_2^n$.

Si des vecteurs sont uniformément distribués dans $\mathcal{B}(\mathbf{0}, d)$ alors ils sont en fait typiquement localisés sur la sphère $\mathcal{S}(\mathbf{0}, d) := \{\mathbf{x} \in \mathbb{F}_2^n \ : \ |\mathbf{x}| = d\}$. Dans cette sous-section, nous allons donc nous concentrer sur le problème des presque-collisions sur la sphère $\mathcal{S}(\mathbf{0}, d)$ où $d \in \left[0, \frac{n}{2}\right]$.

Soit $\mathscr{L}'$ la liste de vecteurs de $\mathcal{S}(\mathbf{0}, d)$ dans laquelle nous recherchons les couples d'éléments proches. Notons alors L' sa taille. Pour construire nos fonctions de hachage nous utilisons des codes qui ne vivent pas dans le même espace que les vecteurs de $\mathscr{L}'$. En effet, à chaque itération de l'algorithme 6.2.1, nous choisissons un code $\mathcal{C}'$ en tirant T' mots tirés uniformément sur la sphère $\mathcal{S}(\mathbf{0}, D)$ où $T' \in \mathbb{N}$ et $D \in [\![0, n]\!]$ sont des paramètres à déterminer. Nous appliquons alors la fonction de hachage floue suivante :

$$\begin{array}{rccc} h_{\mathcal{C}}^{d'} : & \mathcal{S}(\mathbf{0}, d) & \longrightarrow & \mathscr{P}(\mathcal{C}) \\ & \mathbf{x} & \longmapsto & \{\mathbf{c} \in \mathcal{C} \ : \ \Delta(\mathbf{x}, \mathbf{c}) \leqslant d'\} \end{array} \tag{8.5}$$

où $d' \in [\![0, n]\!]$ est un troisième paramètre à optimiser. Comme dans la section 6.2, nous supposons que des décodeurs en liste idéaux calculent les fonctions $h_{\mathcal{C}}^{d'}$ en un temps de l'ordre de la taille de la liste retournée.

Notation 8.2.3. *Nous utilisons des notations analogues à la section 6.2 :*

$$\mathcal{P}_{D,d}(\leqslant d') := \mathbb{P}\left(\mathbf{u} \in \mathcal{B}(\mathbf{x}, d')\right) \tag{8.6}$$

$$\mathcal{P}_{D,d}(\leqslant d' \mid w) := \mathbb{P}\left(\mathbf{u} \in \mathcal{B}(\mathbf{x}, d') \cap \mathcal{B}(\mathbf{y}, d')\right) \tag{8.7}$$

où $\mathbf{u}$ est choisi uniformément dans $\mathcal{S}(\mathbf{0}, D)$ et $\mathbf{x}, \mathbf{y} \in \mathcal{S}(\mathbf{0}, d)$ sont tels que $\Delta(\mathbf{x}, \mathbf{y}) = w$.

Lemme 8.2.4. *Soient $D := \lfloor \gamma n \rfloor$, $d := \lceil \delta n \rceil$ et $d' := \lceil \delta' n \rceil$ pour des constantes $\gamma, \delta \in [0, 1]$ et $\delta' \in [|\gamma - \delta|, 1]$. On a :*

$$\mathcal{P}_{D,d}(\leqslant d') = \begin{cases} \tilde{\Theta}\left(2^{n\left(\delta h_2\left(\frac{\delta' - \gamma + \delta}{2\delta}\right) + (1-\delta) h_2\left(\frac{\delta' + \gamma - \delta}{2(1-\delta)}\right) - h_2(\gamma)\right)}\right) & \text{si } \delta' < \delta + \gamma - 2\delta\gamma \\ \tilde{\Theta}(1) & \text{sinon} \end{cases} \tag{8.8}$$

Démonstration du lemme 8.2.4.

Nous remarquons tout d'abord :

$$\mathcal{P}_{D,d}\left(\leqslant d'\right) \quad = \quad \frac{\#\mathcal{B}\left(\mathbf{x},d'\right)\cap\mathcal{S}\left(\mathbf{0},D\right)}{\#\mathcal{S}\left(\mathbf{0},D\right)}$$

où $|\mathbf{x}|=d$.
Soit $\mathbf{c}\in\mathcal{B}\left(\mathbf{x},d'\right)\cap\mathcal{S}\left(\mathbf{0},D\right)$ et soit $a:=\Delta\left(\mathbf{c},\mathbf{x}\right)\in[0,d']$. La figure 8.4 représente la situation.

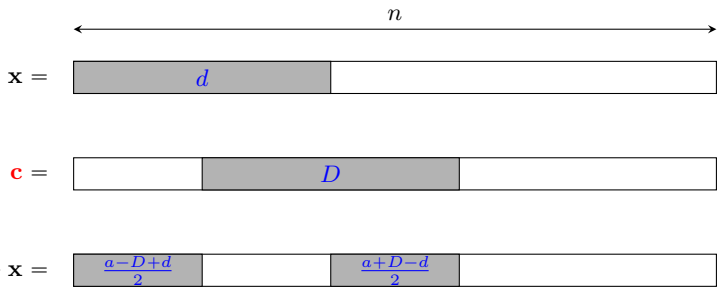


Figure 8.4 – Configuration du vecteur $\mathbf{c}\in\mathcal{B}\left(\mathbf{x},d'\right)\cap\mathcal{S}\left(\mathbf{0},D\right)$ à permutation près des positions.

Nous avons alors :

$$\begin{aligned}\#\mathcal{B}\left(\mathbf{x},d'\right)\cap\mathcal{S}\left(\mathbf{0},D\right) &= \sum_{\substack{a=0\\ a\equiv D+d\ [2]}}^{d'}\binom{d}{\frac{a-D+d}{2}}\binom{n-d}{\frac{a+D-d}{2}}\\ &= \tilde{\Theta}\left(\sup_{\alpha\in[0,\delta']\cap A}2^{n\left(\delta h_2\left(\frac{\alpha-\gamma+\delta}{2\delta}\right)+(1-\delta)h_2\left(\frac{\alpha+\gamma-\delta}{2(1-\delta)}\right)\right)}\right)\end{aligned}$$

où $A:=[\gamma-\delta,\gamma+\delta]\cap[(1-\gamma)-(1-\delta),(1-\gamma)+(1-\delta)]$.
Soit f la fonction suivante :

$$\begin{array}{rccc} f: & A & \longrightarrow & [0,1]\\ & \alpha & \longmapsto & \delta h_2\left(\frac{\alpha-\gamma+\delta}{2\delta}\right)+(1-\delta)h_2\left(\frac{\alpha+\gamma-\delta}{2(1-\delta)}\right)\end{array}$$

Nous avons :

$$f'(\alpha)=\frac{1}{2}\Big(\log_2(\delta-\alpha+\gamma)-\log_2(\delta+\alpha-\gamma)+\log_2(2-\delta-\alpha-\gamma)-\log_2(-\delta+\alpha+\gamma)\Big)$$

Et donc $f'(\alpha)>0$ lorsque $\alpha>\delta+\gamma-2\delta\gamma$ et $f'(\alpha)\leqslant 0$ sinon. Donc la borne supérieure de $f(\alpha)$ est atteinte dans $[0,\delta']\cap A$ pour $\alpha=\min\left(\delta',\delta+\gamma-2\delta\gamma\right)$.

Nous terminons la preuve en remarquant que $\#\mathcal{S}\left(\mathbf{0},D\right)=\binom{n}{D}=\tilde{\Theta}\left(2^{nh_2(\gamma)}\right)$. □

Lemme 8.2.5. *Soient $w:=\lfloor\omega n\rfloor$, $d:=\lceil\delta n\rceil$, $D:=\lfloor\gamma n\rfloor$ et $d':=\lceil\delta' n\rceil$ pour des constantes $\omega\in\left[0,\frac{1}{2}\right]$, $\delta\in\left[\frac{\omega}{2},\frac{1}{2}\right]$, $\gamma\in[0,1]$ et $\delta'\in\left[\max\left(\frac{\omega}{2},|\gamma-\delta|\right),1\right]$. On a :*

$$\mathcal{P}_{D,d}\left(\leqslant d'\mid w\right)=\begin{cases}\tilde{\Theta}\left(1\right) & \text{si } \delta'\geqslant\delta+\gamma-2\delta\gamma\\ \tilde{\Theta}\left(2^{n\left(\left(\delta-\frac{\omega}{2}\right)h_2\left(\frac{\alpha_0}{\delta-\frac{\omega}{2}}\right)+\left(1-\delta-\frac{\omega}{2}\right)h_2\left(\frac{\delta'-\delta+\alpha_0}{1-\delta-\frac{\omega}{2}}\right)+\omega h_2\left(\frac{\gamma-\delta'+\delta-2\alpha_0}{\omega}\right)-h_2(\gamma)\right)}\right) & \text{sinon}\end{cases} \tag{8.9}$$

où $\alpha_0\in\left[0,\delta-\frac{\omega}{2}\right]\cap\left[\delta-\delta',1-\delta'-\frac{\omega}{2}\right]\cap\left[\frac{\gamma-\delta'+\delta-\omega}{2},\frac{\gamma-\delta'+\delta}{2}\right]$ maximise l'expression.

Démonstration du lemme 8.2.5.

Nous remarquons tout d'abord :

$$\mathcal{P}_{D,d}\left(\leqslant d' \mid w\right) \quad = \quad \frac{\#\mathcal{B}\left(\mathbf{x}, d'\right) \cap \mathcal{B}\left(\mathbf{y}, d'\right) \cap \mathcal{S}\left(\mathbf{0}, D\right)}{\#\mathcal{S}\left(\mathbf{0}, D\right)}$$

où $|\mathbf{x}| = |\mathbf{y}| = d$ et $\Delta\left(\mathbf{x}, \mathbf{y}\right) = w$. La figure 8.5 représente alors la situation.

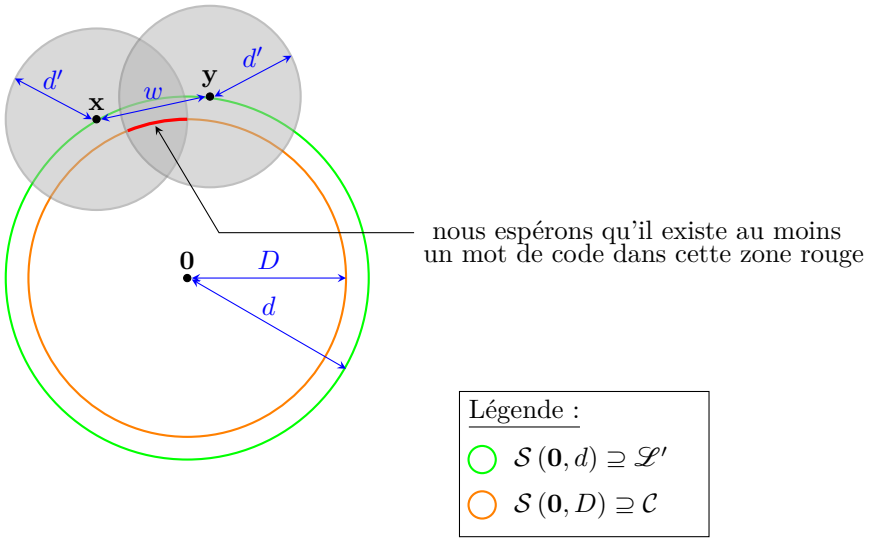


Figure 8.5 – Représentation schématique de l'intersection de boules qui nous intéresse.

Soit $\mathbf{c} \in \mathcal{B}\left(\mathbf{x}, d'\right) \cap \mathcal{B}\left(\mathbf{y}, d'\right) \cap \mathcal{S}\left(\mathbf{0}, D\right)$ et soient $s := \Delta\left(\mathbf{c}, \mathbf{x}\right) \in [\![0, d']\!]$ et $t := \Delta\left(\mathbf{c}, \mathbf{x}\right) \in [\![0, d']\!]$. Nous notons aussi a la taille de l'intersection des supports de $\mathbf{c}$, $\mathbf{x}$ et $\mathbf{y}$. Nous représentons la situation à l'aide de la figure 8.6.

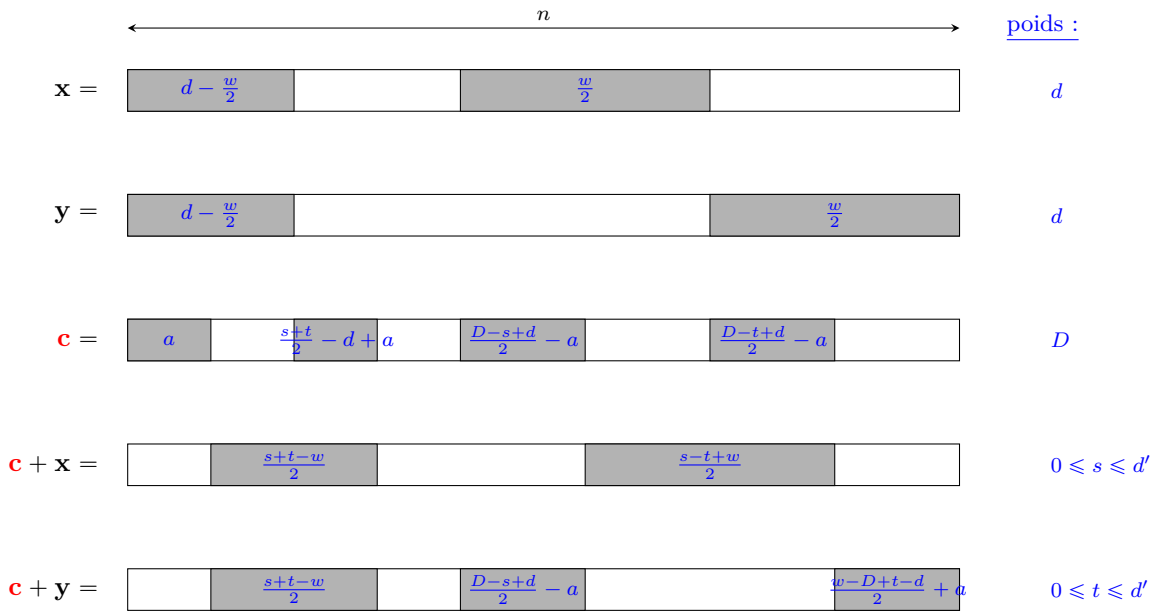


Figure 8.6 – Configuration du vecteur $\mathbf{c} \in \mathcal{B}(\mathbf{x}, d') \cap \mathcal{B}(\mathbf{y}, d') \cap \mathcal{S}(\mathbf{0}, D)$ à permutation près des positions.

Nous avons ainsi :

$$\#\mathcal{B}(\mathbf{x}, d') \cap \mathcal{B}(\mathbf{y}, d') \cap \mathcal{S}(\mathbf{0}, D)$$

$$= \sum_{\substack{s=0 \\ s\equiv D\ [2]}}^{d'} \sum_{\substack{t=0 \\ t\equiv D\ [2]}}^{d'} \sum_{a\in A} \binom{d-\frac{w}{2}}{a} \binom{n-d-\frac{w}{2}}{\frac{s+t}{2}-d+a} \binom{\frac{w}{2}}{\frac{D-s+d}{2}-a} \binom{\frac{w}{2}}{\frac{D-t+d}{2}-a}$$

où $A := [\![0, d-\frac{w}{2}]\!] \cap [\![d-\frac{s+t}{2}, n-\frac{w+s+t}{2}]\!] \cap [\![\frac{D-s+d-w}{2}, \frac{D-s+d}{2}]\!] \cap [\![\frac{D-t+d-w}{2}, \frac{D-t+d}{2}]\!]$.

Aucune des deux boules $\mathcal{B}(\mathbf{x}, d')$ et $\mathcal{B}(\mathbf{y}, d')$ ne domine l'autre ; autrement dit, nous avons $\Delta(\mathbf{c}, \mathbf{x}) = \Delta(\mathbf{c}, \mathbf{y}) = s = t$. La figure 8.7 est alors une mise à jour de la figure précédente 8.6.

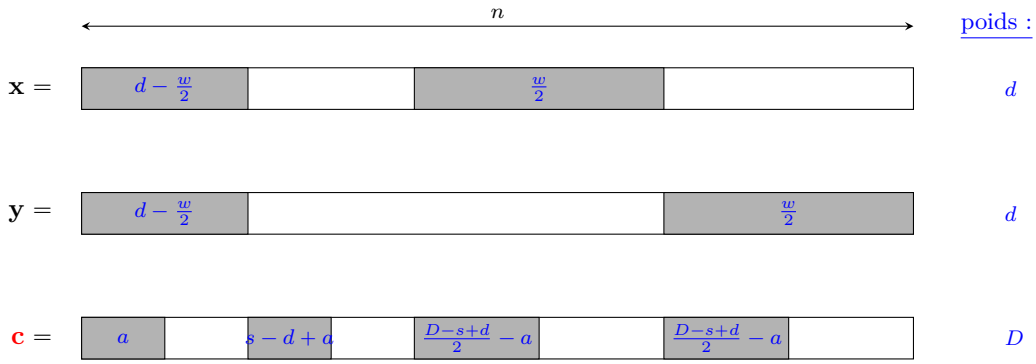


Figure 8.7 – Configuration du vecteur $\mathbf{c} \in \mathcal{S}(\mathbf{x}, d') \cap \mathcal{S}(\mathbf{y}, d') \cap \mathcal{S}(\mathbf{0}, D)$ à permutation près des positions.

Ce qui nous donne :

$$\begin{aligned}
& \#\mathcal{B}\left(\mathbf{x}, d'\right) \cap \mathcal{B}\left(\mathbf{y}, d'\right) \cap \mathcal{S}\left(\mathbf{0}, D\right) \\
= \quad & \Theta\left(\sup_{s \in [\![0, d']\!]} \left(\#\mathcal{S}\left(\mathbf{x}, s\right) \cap \mathcal{S}\left(\mathbf{y}, s\right) \cap \mathcal{S}\left(\mathbf{0}, D\right) \right) \right) \\
= \quad & \Theta\left(\sup_{\substack{s \in [\![0, d']\!] \\ s \equiv D+d \ [2]}} \left(\sum_{a \in A} \binom{d - \frac{w}{2}}{a} \binom{n - d - \frac{w}{2}}{s - d + a} \binom{\frac{w}{2}}{\frac{D-s+d}{2} - a}^2 \right) \right) \\
= \quad & \tilde{\Theta}\left(2^{n\left(\sup_{(\beta,\alpha) \in U \times V} f(\beta, \alpha)\right)}\right)
\end{aligned}$$

où :

$$f(\beta, \alpha) := \left(\delta - \frac{\omega}{2}\right) h_2\left(\frac{\alpha}{\delta - \frac{\omega}{2}}\right) + \left(1 - \delta - \frac{\omega}{2}\right) h_2\left(\frac{\beta - \delta + \alpha}{1 - \delta - \frac{\omega}{2}}\right) + \omega h_2\left(\frac{\gamma - \beta + \delta - 2\alpha}{w}\right).$$

et :

$$U := \left[0, \delta'\right] \cap \left[\tfrac{\omega}{2}, 1 - \tfrac{\omega}{2}\right] \cap \left[\delta - \gamma, \delta + \gamma\right] \cap \left[\gamma - \delta, 2 - \delta - \gamma\right]$$

$$V := \left[0, \delta - \tfrac{\omega}{2}\right] \cap \left[\delta - \beta, 1 - \beta - \tfrac{\omega}{2}\right] \cap \left[\tfrac{\gamma - \beta + \delta - \omega}{2}, \tfrac{\gamma - \beta + \delta}{2}\right]$$

(avec les hypothèses du lemme, l'ensemble $U \times V$ n'est pas vide).
Une analyse des dérivées partielles de la fonction f nous permet de montrer que le maximum de f est atteint pour :

$$(\beta, \alpha) = \begin{cases} \left(\delta + \gamma - 2\delta\gamma, \gamma\left(\delta - \frac{\omega}{2}\right)\right) & \text{si } \delta' \geqslant \delta + \gamma - 2\delta\gamma \\ \left(\delta', \alpha_0\right) & \text{sinon} \end{cases}$$

où α_0 maximise la fonction $f(\delta', \cdot)$.
Nous terminons la preuve en remarquant que $\#\mathcal{S}\left(\mathbf{0}, D\right) = \binom{n}{D} = \tilde{\Theta}\left(2^{nh_2(\gamma)}\right)$ □

À propos de la valeur de α_0 dans le lemme 8.2.5. Soit f la fonction définie comme dans la preuve du lemme 8.2.5. Une analyse différentielle de la fonction $f(d', \cdot)$ nous permet de montrer que α_0 est l'unique réel dans $\left[0, \delta - \frac{\omega}{2}\right] \cap \left[\delta - \delta', 1 - \delta' - \frac{\omega}{2}\right] \cap \left[\frac{\gamma - \delta' + \delta - \omega}{2}, \frac{\gamma - \delta' + \delta}{2}\right]$ solution de l'équation (d'inconnue α) :

$$\left(\delta - \frac{\omega}{2} - \alpha\right)\left(1 - \frac{\omega}{2} - \delta' - \alpha\right)\left(\gamma - \delta' + \delta - 2\alpha\right)^2 = \alpha\left(\delta' - \delta + \alpha\right)\left(\omega - \gamma + \delta' - \delta + 2\alpha\right)^2 \tag{8.10}$$

Nous pouvons alors utiliser la méthode de Cardan pour déterminer les solutions de l'équation (8.10) et ainsi décrire les variations de $f(d', \cdot)$; ce qui nous permet finalement de déterminer α_0.

Corollaire 8.2.6 (du théorème 6.2.3)**.** *Soient $w := \lfloor \omega n \rfloor$ et $d := \lceil \delta n \rceil$ pour des constantes $\omega \in \left[0, \frac{1}{2}\right]$ et $\delta \in \left[\frac{\omega}{2}, \frac{1}{2}\right]$.*

Nous supposons que pour tout $D \in [\![0, n]\!]$, pour tout code aléatoire $\mathcal{C}' \subseteq \mathcal{S}\left(\mathbf{0}, D\right)$ de taille T' et pour toute distance $d' \in \left[\!\left[\max\left(\frac{w}{2}, |D - d|\right), d + D - \frac{2dD}{n}\right]\!\right]$, il existe un décodeur en liste idéal que s'exécute en un temps de l'ordre de $O(\max\left(1, T' \cdot \mathcal{P}_{D,d}\left(\leqslant d'\right)\right))$. L'algorithme 6.2.1 peut résoudre le problème des w-presque-collisions dans une liste de L' éléments tirés uniformément sur $\mathcal{S}\left(\mathbf{0}, d\right)$. L'impact mémoire est alors de l'ordre de :

$$S_{\text{Codes}}^{\text{Aléa}} = O\left(L'\right) \tag{8.11}$$

et la complexité en temps est de l'ordre de :

$$T_{\text{Codes}}^{\text{Aléa}}\left(D, d'\right) = O\left(\frac{L' \cdot \mathcal{P}_{D,d}\left(\leqslant d'\right) + \left(L' \cdot \mathcal{P}_{D,d}\left(\leqslant d'\right)\right)^2}{\mathcal{P}_{D,d}\left(\leqslant d' \mid w\right)}\right) \tag{8.12}$$

Nous ne pouvons pas appliquer directement le théorème 6.2.6 car le lemme 6.2.4 n'est pas forcément vérifié pour un mauvais choix de D. Posons f_γ et g_γ les deux fonctions définies par :

$$f_\gamma(\delta') := \frac{L' \cdot \mathcal{P}_{D,d}\left(\leqslant \lceil \delta' n \rceil\right)}{\mathcal{P}_{D,d}\left(\leqslant \lceil \delta' n \rceil \mid w\right)} \tag{8.13}$$

$$g_\gamma(\delta') := \frac{\left(L' \cdot \mathcal{P}_{D,d}\left(\leqslant \lceil \delta' n \rceil\right)\right)^2}{\mathcal{P}_{D,d}\left(\leqslant \lceil \delta' n \rceil \mid w\right)} \tag{8.14}$$

où $D := \lfloor \gamma n \rfloor$. En étudiant les variations de f_γ et g_γ sur $A := \left[\max\left(\frac{\omega}{2}, \left|\frac{\gamma}{n} - \delta\right|\right), \delta + \gamma - 2\delta\gamma\right]$, on peut optimiser le paramètre d' de l'équation (8.12). La complexité alors obtenue est :

$$T_{\text{Codes}}^{\text{Aléa}}(\gamma) := \inf_{\delta' \in A}\left(T_{\text{Codes}}^{\text{Aléa}}\left(\lfloor \gamma n \rfloor, \lceil \delta' n \rceil\right)\right) = \inf_{\delta' \in A} \max\left(f_\gamma(\delta'), g_\gamma(\delta')\right) = \inf_{\substack{\delta' \in A \\ \delta' \geqslant \delta_{\text{GV}}^-(L')}} g_\gamma(\delta') \tag{8.15}$$

où $\delta_{\text{GV}}^-(L') \in \left[|\gamma - \delta|, \gamma + \delta - 2\delta\gamma\right]$ est tel que $L' \cdot \mathcal{P}_{D,d}\left(\leqslant \lceil \delta' n \rceil\right) = 1$.

La figure 8.8 montre différents exemples de variations des fonctions f_γ et g_γ pour différents choix de d et γ.

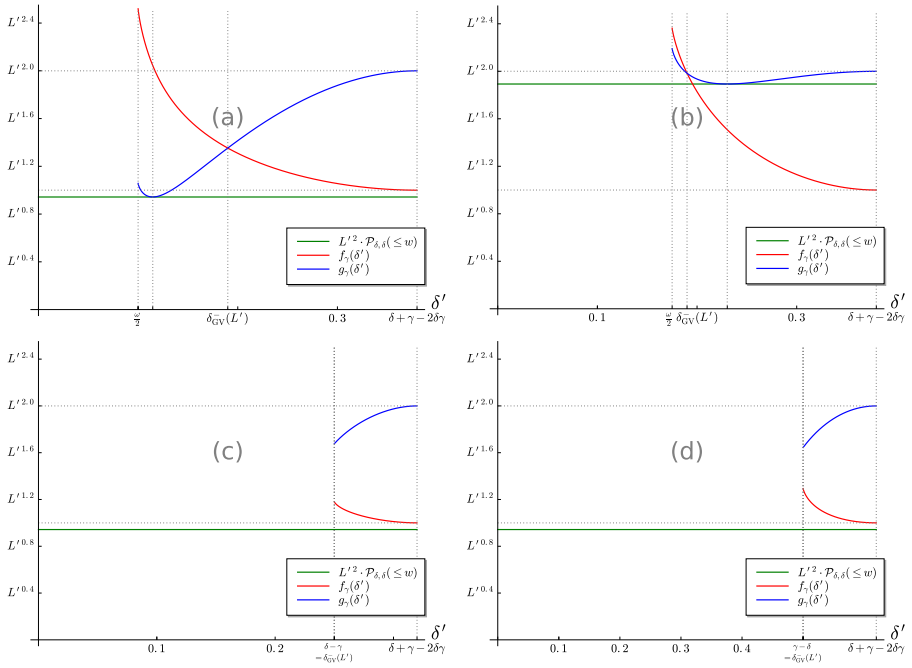


Figure 8.8 – Représentations graphiques de f_γ et g_γ ($L' = L\binom{n}{d}2^{-n}$ avec $L = 2^{0.3n}$). (a) $\delta = 0.3$, $\gamma = 0.2$, $\omega = 0.2$, (b) $\delta = 0.3$, $\gamma = 0.2$, $\omega = 0.35$, (c) $\delta = 0.3$, $\gamma = 0.05$, $\omega = 0.2$ et (d) $\delta = 0.3$, $\gamma = 0.8$, $\omega = 0.2$.

La figure 8.9 représente la complexité de la recherche de presque-collisions sur $\mathcal{B}(\mathbf{0}, d)$ en fonction du rayon D de la sphère sur laquelle vivent nos codes. Nous remarquons alors

que prendre $D = d$ n'est pas le meilleur choix. En effet, dans l'exemple de la figure, le choix optimal est $D \simeq \lfloor 0.1782n \rfloor$ tandis que $d = \lceil 0.3n \rceil$.

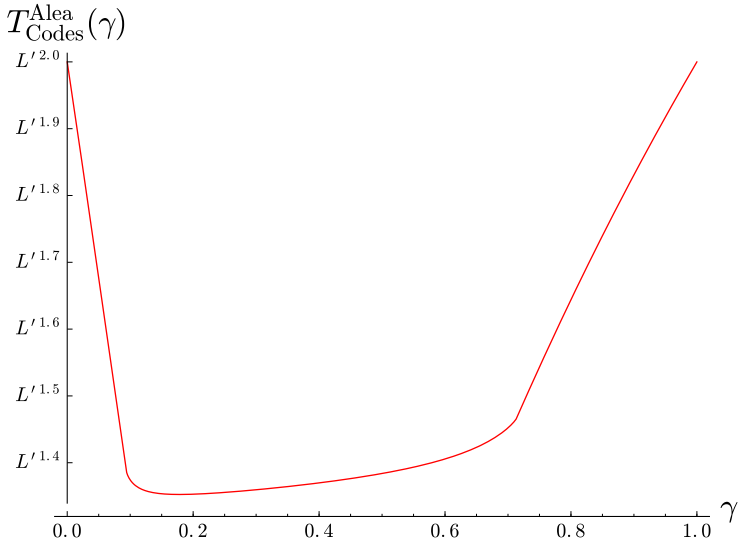


Figure 8.9 – Complexité de notre recherche de presque-collisions dans $\mathcal{B}(\mathbf{0}, d)$ en fonction de $D := \lfloor \gamma n \rfloor$ ($\delta = 0.3$, $\omega = 0.2$, $L = 2^{0.3n}$ et $L' = L \cdot \binom{n}{d} \cdot 2^{-n}$).

Finalement, on choisit le paramètre $D := \lfloor \gamma n \rfloor$ qui minimise la complexité de notre méthode :

$$T^{\text{Aléa}}_{\text{Codes}} = \inf_{\gamma \in [0,1]} T^{\text{Aléa}}_{\text{Codes}}(\lfloor \gamma n \rfloor) \tag{8.16}$$

Par la suite, nous effectuerons cette minimisation de manière numérique. La figure 8.10 représente la complexité de notre méthode pour la recherche de w-presque-collisions dans une liste composée de L' éléments tirés uniformément dans la boule $\mathcal{B}(\mathbf{0}, d) \subseteq \mathbb{F}_2^n$.

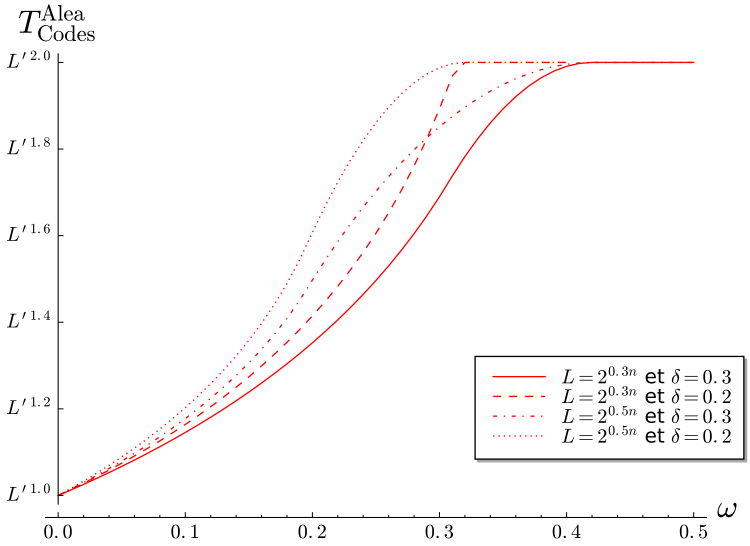


Figure 8.10 – Complexité de notre recherche de presque-collisions dans $\mathcal{B}(\mathbf{0}, d)$ en fonction de $w := \lfloor \omega n \rfloor$ pour différentes tailles de liste $L' = L \cdot \binom{n}{d} \cdot 2^{-n}$ et différents rayon $d := \lceil \delta n \rceil$.

8.2.4 Analyse de l'algorithme récursif sur $\mathbb{F}_2^n$

Nous avons à présent les outils pour analyser l'algorithme 8.2.2 dans le cas binaire. On applique le théorème 8.2.2 :

$$T^{\text{Aléa}}_{\text{Récursif}}(d) = O\left(\frac{L\cdot\mathcal{P}(\leqslant d) + T^{\text{Aléa}}_{\text{Codes}}(\mathcal{B}(\mathbf{0},d), L\cdot\mathcal{P}(\leqslant d), w)}{\mathcal{P}(\leqslant d \mid w)}\right) \tag{8.17}$$

où $T^{\text{Aléa}}_{\text{Codes}}(\mathcal{B}(\mathbf{0},d), L\cdot\mathcal{P}(\leqslant d), w)$ est le coût de la méthode des codes pour résoudre le problème des w-presque-collisions dans une liste de $L\cdot\mathcal{P}(\leqslant d)$ éléments tirés uniformément dans $\mathcal{B}(\mathbf{0},d) \subseteq \mathbb{F}_2^n$. Cette complexité est donnée par l'équation (8.16).

On constate alors que pour L et w fixés, la complexité $T^{\text{Aléa}}_{\text{Récursif}}(d)$ est constante pour n'importe quel $d \in [\![d^-_{\text{GV}}(L), \frac{n}{2}]\!]$. On peut alors vérifier que cette complexité est celle de la méthode des codes ; pour cela, nous rappelons que la méthode des codes est l'instance de l'algorithme récursif 8.2.2 correspondant au choix $d = \max(d^-_{\text{GV}}(L), d_w)$.

Une explication de l'inefficacité de la méthode récursive peut être donnée. Pour chaque itération de l'algorithme 8.2.2, nous résolvons l'étape **11** avec une procédure itérative. Nous qualifions d'interne les itérations de cette procédure. Notons s' le nombre d'itérations internes nécessaires pour effectuer l'étape **11** d'une itération globale. Ainsi, pour tout $(\ell,\ell',i) \in [\![1,s]\!]\times[\![1,s']\!]\times[\![1,T]\!]$, nous notons $\mathcal{C}_{\ell,\ell',i}$ le code sur une sphère binaire associé à l'itération globale $\ell \in [\![1,s]\!]$, à l'adresse i de la table de hachage et à l'itération interne $\ell' \in [\![1,s']\!]$. L'algorithme 8.2.2 est alors équivalent à l'algorithme 6.2.1 où les codes utilisés sont $\left\{\bigcup_{i=1}^{T}\mathcal{C}_{\ell,\ell',i}\right\}_{\substack{\ell\in[\![1,s]\!]\\ \ell'\in[\![1,s']\!]}}$. Il semblerait alors que la structure de ces codes n'apportent rien de particulier par rapport à des codes aléatoires.

8.3 Une approche par changement de géométrie

Revenons sur l'algorithme hybride 8.1.1. Cet algorithme consiste à appliquer notre méthode des codes sur une liste $\mathscr{L}'$ pré-calculée à partir de la liste originale $\mathscr{L}$. Ce pré-calcul vise à diminuer la dimension de l'espace ambiant ; sachant que cette dimension est fortement liée à la difficulté du problème des presque-collisions. Plutôt que de diminuer la dimension du problème, nous avons imaginé l'augmenter en "répliquant" les vecteurs de $\mathscr{L}$. Nous allons tout d'abord expliquer l'intuition à l'origine de cette idée et ce que signifie formellement "répliquer" un vecteur.

Dans cette section, nous supposons que l'espace ambiant est la sphère unitaire euclidienne $\mathcal{S}_1^n$ munie de la métrique des angles définie dans la section 7.1. Cependant, notre discours se généralise trivialement à d'autres espaces métriques.

Supposons un code aléatoire $\mathcal{C}$ de taille L (où $L := \#\mathscr{L}$) sur $\mathcal{S}_1^n$ muni d'un décodeur idéal qui décode à distance d. Nous souhaitons alors maximiser la probabilité de collision de deux mots proches ; c'est-à-dire maximiser la probabilité :

$$\mathcal{P}(\leqslant d \mid w) := \mathbb{P}(\mathbf{u} \in \mathcal{B}(\mathbf{x},d) \cap \mathcal{B}(\mathbf{y},d)) \tag{8.18}$$

où $\mathbf{u}$ est tiré uniformément sur $\mathcal{S}_1^n$ et $\mathbf{x},\mathbf{y} \in \mathcal{S}_1^n$ sont tels que $\Delta(\mathbf{x},\mathbf{y}) = w$. D'après le lemme 7.1.3, nous avons :

$$\mathcal{P}(\leqslant d \mid w) = \tilde{\Theta}\left(\left(1 - \frac{2\cos^2(d)}{1+\cos(w)}\right)^{\frac{n}{2}}\right) \tag{8.19}$$

Cette probabilité est asymptotiquement optimale dès lors que $d = \frac{\pi}{2}$. Cependant, pour une taille de liste L exponentielle en n, ce rayon de décodage est plus grand que la distance de Gilbert-Varshamov que nous rappelons avec l'équation suivante :

$$d^-_{\text{GV}}(L) = \arcsin\left(L^{\frac{-1}{n}}\right) \tag{8.20}$$

Or un décodage au delà de la distance de Gilbert-Varshamov retourne une liste de taille exponentielle ; démultipliant ainsi le nombre de couples à vérifier à chaque itération de notre algorithme. Malheureusement, ce phénomène n'est pas compensé par le gain obtenu sur la probabilité de collision.

Ainsi, nous souhaitons décoder à une distance aussi proche que possible de $\frac{\pi}{2}$ pour maximiser la probabilité de collision de deux mots proches mais dans le même temps, nous ne voulons pas décoder au delà de $d^-_{\mathrm{GV}}(L)$ pour que les cellules de notre table de hachage soient occupées typiquement par $\Omega(1)$ éléments. Nous pouvons toutefois augmenter artificiellement la distance de Gilbert-Varshamov de L sans changer la distance relative entre les points de $\mathscr{L}$. Nous obtenons ce résultat en changeant la dimension de l'espace ambiant. Pour cela, nous remplaçons chaque vecteur de $\mathscr{L}$ par un vecteur correspondant à M répliques du vecteur considéré. Les répliques vivent alors dans la sphère euclidienne $\mathcal{S}_M^{nM} := \{\mathbf{x} \in \mathbb{R}^n M : ||\mathbf{x}|| = M\}$. Nous normalisons la situation en divisant toutes les positions des répliques par $\sqrt{M}$.

Notation 8.3.1. *Pour tout $\mathbf{x} \in \mathcal{S}_1^n$, nous notons $\mathbf{x}^{|M}$ le vecteur de $\mathcal{S}_1^{nM}$ définit par :*

$$\mathbf{x}^{|M} := \frac{1}{\sqrt{M}} \cdot (\underbrace{\mathbf{x}|\ldots|\mathbf{x}}_{M \text{ fois}})$$

où $|$ est l'opération de concaténation.

Nous notons aussi $\mathscr{L}^{|M}$, la liste constituée des répliques de $\mathscr{L}$:

$$\mathscr{L}^{|M} := \left\{ \mathbf{x}^{|M} \in \mathcal{S}_1^{nM} : \mathbf{x} \in \mathscr{L} \right\}$$

Notre changement de géométrie consiste alors à considérer $\mathscr{L}^{|M}$ à la place de $\mathscr{L}$. Dans ce cas, les vecteurs que nous devons hacher sont de longueur nM ; cette valeur sera aussi la longueur des codes à décoder. Cependant, la taille de la liste $\mathscr{L}^{|M}$ est toujours L. Donc la distance de Gilbert-Varshamov est maintenant :

$$d^-_{\mathrm{GV}}(L) = \arcsin\left(L^{\frac{-1}{nM}}\right)$$

Elle tend alors vers $\frac{\pi}{2}$ lorsque M tend vers l'infini. De plus, la distance entre les répliques $\mathbf{x}^{|M}$ et $\mathbf{y}^{|M}$ est la même que la distance entre $\mathbf{x}$ et $\mathbf{y}$ puisque :

$$\begin{aligned}
\Delta\left(\mathbf{x}^{|M}, \mathbf{y}^{|M}\right) &= 1 - \left\langle \mathbf{x}^{|M}, \mathbf{y}^{|M} \right\rangle \\
&= 1 - M \cdot \left\langle \tfrac{1}{\sqrt{M}}\mathbf{x}, \tfrac{1}{\sqrt{M}}\mathbf{y} \right\rangle \\
&= 1 - \tfrac{M}{\sqrt{M}^2} \cdot \langle \mathbf{x}, \mathbf{y} \rangle \\
&= \Delta(\mathbf{x}, \mathbf{y})
\end{aligned}$$

Toutefois, il ne faut pas oublier de prendre en compte un phénomène important : lors de notre changement de géométrie, les éléments de la nouvelle liste $\mathscr{L}^{|M}$ ne sont pas uniformément distribués dans $\mathcal{S}_1^{nM}$. On peut le voir facilement en remarquant notamment que tous les mots dont la norme n'est pas un multiple de M ne peuvent pas apparaitre dans $\mathscr{L}^{|M}$. Cette *mauvaise* distribution entraine un remplissage déséquilibré des cellules de la table de hachage ; c'est-à-dire qu'un nombre exponentiel de cellules sont vides tandis que d'autres sont remplies avec un nombre exponentiel d'éléments. Et bien qu'en moyenne, le taux de remplissage soit de $\Theta(1)$, le nombre de couples à tester est par contre exponentiellement plus grand. Plus exactement, pour toute distance de décodage $d \in \left[0, \frac{\pi}{2}\right]$, le nombre de couples à tester par cellule lors d'une itération de notre recherche de presque-collisions est en moyenne :

$$L^2 \int_0^{\pi} \left(\mathcal{P}_n(\leqslant x) \cdot \mathcal{P}_n(\leqslant d \mid x)\right) dx = O\left(L^2 \sup_{0 \leqslant x \leqslant \pi} \left(\mathcal{P}_n(\leqslant x) \cdot \mathcal{P}_{nM}(\leqslant d \mid x)\right)\right) \quad (8.21)$$

où $\mathcal{P}_n(\leqslant \cdot)$ et $\mathcal{P}_n(\leqslant \cdot \mid \cdot)$ sont donnés dans les lemmes 7.1.2 et 7.1.3 mais où cette fois-ci, nous précisons la dimension de l'espace considéré.

La complexité de notre méthode utilisant des répliques est donc finalement :

$$T^{\text{Aléa}}_{\text{Repliques}}(d) = O\left(\frac{L \cdot \mathcal{P}_{nM}(\leqslant d) + L^2 \cdot \sup\limits_{0<x<\pi} (\mathcal{P}_n(\leqslant x) \cdot \mathcal{P}_{nM}(\leqslant d \mid x))}{\mathcal{P}_{nM}(\leqslant d \mid w)}\right) \tag{8.22}$$

On pose alors f_M et g_M les deux fonctions suivantes :

$$f_M(d) := \frac{L \cdot \mathcal{P}_{nM}(\leqslant d)}{\mathcal{P}_{nM}(\leqslant d \mid w)} \tag{8.23}$$

$$g_M(d) := \frac{L^2 \cdot \sup\limits_{0<x<\pi} (\mathcal{P}_n(\leqslant x) \cdot \mathcal{P}_{nM}(\leqslant d \mid x))}{\mathcal{P}_{nM}(\leqslant d \mid w)} \tag{8.24}$$

La complexité optimale de notre méthode est donc $\inf_{d \in [0, \frac{\pi}{2}]} (\max(f_D(d), g_D(d)))$. Sur la figure 8.11 nous avons représenté les fonctions f_M et g_M pour différents choix de M. Nous avons considéré une instance du problème des presque-collisions où $L = \left(\frac{4}{3}\right)^{\frac{n}{2}+o(1)}$ et $w = \frac{\pi}{3}$. Nous avons vu au début de la section 7.1 que cette instance est directement reliée au problème SVP qui est un problème important dans le domaine de la cryptographie basée sur les réseaux euclidiens. La figure 8.12 représente pour cette même instance la complexité de notre méthode des répliques en fonction de M. Nous constatons alors que celle-ci augmente légèrement à mesure que M grandit. Nous en concluons que la méthodes des répliques est optimale lorsque $M = 1$.

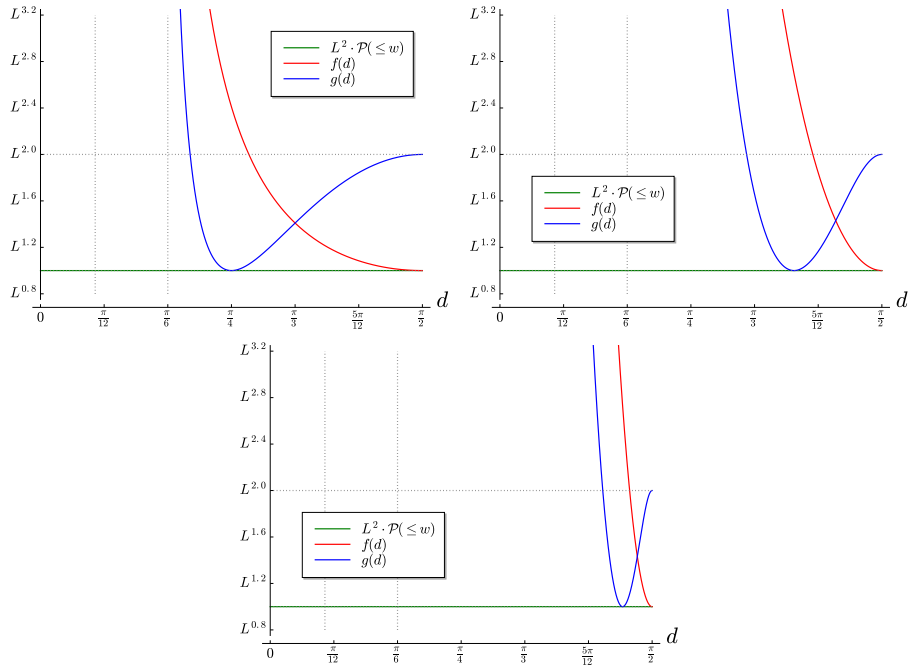


Figure 8.11 – Représentations graphiques des fonctions f_M et g_M pour $L = \left(\frac{4}{3}\right)^{\frac{n}{2}+o(1)}$ et $w = \frac{\pi}{3}$. À gauche $M = 1$, à droite $M = 10$ et en bas $M = 100$.

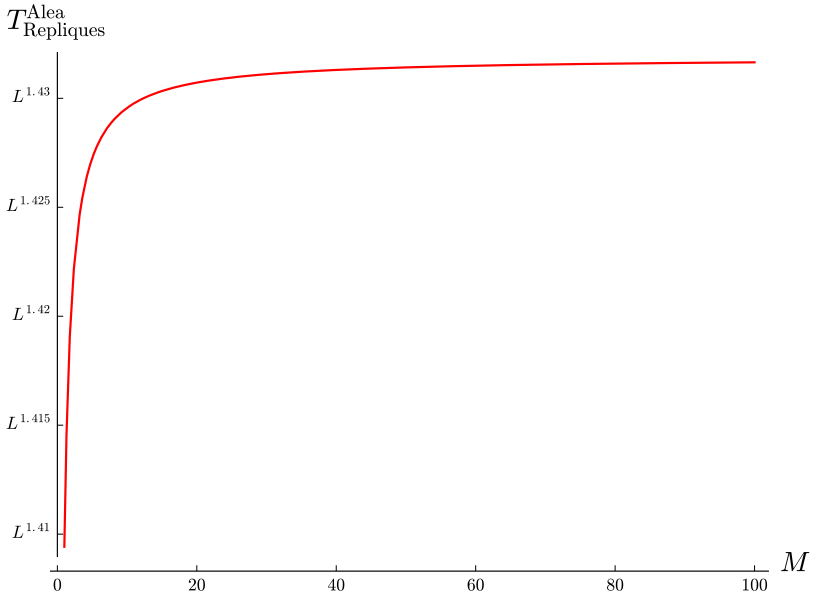


Figure 8.12 – Complexité de la méthode du changement de géométrie en fonction de M pour $L = \left(\frac{4}{3}\right)^{\frac{\pi}{2}+o(1)}$ et $w = \frac{\pi}{3}$.

Chapitre 9

Construction pratique de fonctions de hachage floues

Rappelons que notre méthode suppose des décodeurs en liste idéaux de codes aléatoires ; c'est-à-dire que pour tout code $\mathcal{C} \subseteq \mathbb{F}_q^n$ et tout rayon de décodage $d \in [\![0, n]\!]$, nous devons être en mesure de retourner l'ensemble des mots de $\mathcal{C}$ à distance au plus d du mot à décoder en un temps de l'ordre de la taille des listes de décodage.

Plutôt que de considérer des codes aléatoires, nous allons utiliser des familles de codes particuliers. Nous choisissons des familles pour lesquelles nous connaissons un algorithme de décodage en liste qui s'exécute en un temps de l'ordre de la taille des listes retournées (avec éventuellement un facteur sous-exponentiel en n, idéalement polynomial). Pour une taille de liste de décodage fixée, nous attendons également de ces décodeurs en liste qu'ils nous permettent d'obtenir une probabilité de collision aussi proche que possible de celle que nous obtenons avec des décodeurs en liste idéaux de codes aléatoires ; autrement dit, nous espérons que les décodages de deux mots proches dans le code de substitution retournent deux listes qui s'intersectent avec une aussi bonne probabilité qu'avec des décodeurs idéaux de codes aléatoires.

Il existe notamment au moins deux familles de codes qui pourraient être de bons candidats :

— le produit cartésien $\mathcal{C} := \mathcal{C}_1 \times \ldots \mathcal{C}_s$ de s codes aléatoires q-aires de dimension $O\big(\log_q(n)\big)$. Il existe un algorithme de décodage simple qui s'exécute en un temps polynomial et qui produit un mot de code à distance $d^-_{\mathrm{GV}}(n^s)\,(1 + O(n^{-\varepsilon}))$ pour $\varepsilon > 0$ lorsque le rendement de chaque code $\mathcal{C}_i$ est proche de $\frac{s \log_q(n)}{n}$. De plus, il existe un décodage en liste de ces codes qui produit une liste de ℓ mots de codes proches avec une complexité de l'ordre de $O(\ell \cdot \mathrm{poly}\,(n))$ où $\mathrm{poly}\,(n)$ est un polynôme en n. En outre, il existe une version sur $\mathcal{S}_1^n$ de ces codes qui ont déjà été étudiés dans [BDGL16].

— les codes polaires binaires $[n, k]$ proposés dans [Arı09] et décodés avec l'algorithme décrit dans [Kor09, chap. 3]. Ce décodeur produit un mot de code à distance $d^-_{\mathrm{GV}}(2^k) + O\big(2^{-n^{\varepsilon}}\big)$ en un temps de l'ordre de $O(n \log_2(n))$ pour $0 < \varepsilon < \frac{1}{2}$. De plus, les codes polaires possèdent aussi un algorithme de décodage en liste qui produit une liste de ℓ mots proches en un temps de l'ordre de $O(\ell n \log_2(n))$ et une complexité mémoire de $O(\ell n)$ [TV15]. Le décodage en liste des codes polaires permet d'en améliorer fortement les performances. Dans la section 1.4 du chapitre 1, nous avons parlé assez longuement des codes polaires et de leurs décodages simple et en liste.

Dans le cadre de la recherche de presque-collisions, les deux familles de codes sus-citées sont relativement proches des codes aléatoires que nous avons utilisés dans les chapitres

précédents. De plus, ces deux familles possèdent chacune un algorithme de décodage en liste efficace.

Dans la suite, nous appelons *codes juxtaposés* les produits cartésiens de codes. Dans la section 9.1 de ce chapitre, nous analysons l'utilisation de décodages à maximum de vraisemblance de codes juxtaposés permutés dans nos méthodes de recherche de presque-collisions. Dans cette même section, nous donnons un algorithme de décodage en liste des codes juxtaposés permutés. Nous utilisons alors cet algorithme pour la recherche de presque-collisions et proposons d'analyser expérimentalement la méthode obtenue. Enfin, dans la section 9.2 nous remplaçons les codes juxtaposés permutés par des cosets de codes polaires que nous décodons en liste avec l'algorithme de Tal et Vardy. Nous analysons alors les résultats obtenus dans le cadre de la recherche de presque-collisions.

9.1 La recherche de presque-collisions avec des codes juxtaposés

Le comportement du décodage en liste d'un code polaire ne s'analyse pas facilement : en particulier, nous avons des difficultés à déterminer la probabilité que les listes de décodage de deux éléments à distance w s'intersectent. Au contraire, les codes juxtaposés s'analysent très bien dans le cadre de la recherche de presque-collisions. L'idée d'utiliser des codes juxtaposés a déjà été proposée dans [BDGL16] pour la sphère euclidienne. Dans [GMO10] et [Dub10], on retrouve cette même idée pour le cas binaire mais les analyses proposées par les auteurs ne correspondent pas au modèle de proximité qui nous intéresse ; à savoir le décodage générique à poids constant. Nous avons vu dans la section 5.6 que dans le modèle de proximité de [GMO10], l'utilisation de codes juxtaposés pour la recherche de presque-collisions n'entraine qu'un surcoût polynomial par rapport à des décodeurs idéaux de codes aléatoires. Nous allons voir dans cette section que dans notre modèle de proximité, ce facteur est plus important.

9.1.1 Construction du code

Nous n'allons pas utiliser exactement la construction des codes juxtaposés décrite au début de ce chapitre. Tout d'abord, nous allons aussi considérer la sous-famille linéaire de ces codes ; c'est-à-dire que nous supposerons que pour tout $i \in [\![1, s]\!]$, le $\mathcal{C}_i$ est linéaire. De plus, nous allons plutôt utiliser des codes juxtaposés permutés. La permutation permet de mélanger les positions sur lesquelles deux mots proches diffèrent sans changer la distance entre ces mots. Nous espérons ainsi répartir équitablement la distance de deux mots proches dans chacun des codes $\mathcal{C}_i$. Cela nous permettra par la suite de simplifier notre analyse.

Plus formellement, nous définissons les codes juxtaposés permutés de longueur n, de dimension k et d'ordre s de la façon suivante :

Définition 9.1.1. *Soient n, k et s trois entiers tels que $0 \leqslant s \leqslant k \leqslant n$. Un code juxtaposé $[n, k]$ d'ordre s est le code $\mathcal{C}$ tel que :*

$$\mathcal{C} := \pi(\mathcal{C}_1 \times \cdots \times \mathcal{C}_s) := \{\pi(\mathbf{c}) \ : \ \mathbf{c} \in \mathcal{C}_1 \times \cdots \times \mathcal{C}_s\} \tag{9.1}$$

où $\pi \in S_n$ est une permutation des n positions et pour tout $i \in [\![1, s]\!]$, $\mathcal{C}_i$ est un code de longueur $n_i \in \left\{\left\lfloor \frac{n}{s} \right\rfloor, \left\lfloor \frac{n}{s} \right\rfloor + 1\right\}$ et de dimension $k_i \in \left\{\left\lfloor \frac{k}{s} \right\rfloor, \left\lfloor \frac{k}{s} \right\rfloor + 1\right\}$.

Comme $\mathcal{C}$ est de longueur n et de dimension k, nous avons nécessairement :

$$k_1 + \cdots + k_s = k \quad \text{et} \quad n_1 + \cdots + n_s = n \tag{9.2}$$

À chaque itération de notre algorithme de recherche de presque-collisions, nous choisissons un code $\mathcal{C}$ uniformément parmi les codes juxtaposés permutés d'ordre s où s est

un paramètre à optimiser. Un décodage par maximum de vraisemblance d'un mot $\mathbf{y} \in \mathbb{F}_q^n$ consiste essentiellement à rechercher exhaustivement pour chaque $i \in [\![1, s]\!]$, le mot du code $\mathcal{C}_i$ le plus proche de $\mathbf{y}_i$ où $(\mathbf{y}_1, \ldots, \mathbf{y}_s) := \pi^{-1}(\mathbf{y}) \in \mathbb{F}_q^{n_1} \times \cdots \times \mathbb{F}_q^{n_s}$. Ce décodage a une complexité en temps de l'ordre de $O\left(sq^{\frac{k}{s}}\right)$.

9.1.2 Un algorithme de décodage en liste

Soit $\mathcal{C}$ un code juxtaposé permuté $[n, k]_q$ d'ordre s. Ce code peut être décodé en listes de taille ℓ en un temps de l'ordre de $O\left(nq^{\frac{k}{s}} + s\ell\right)$. L'algorithme de décodage que nous proposons est simplement une adaptation sur les espaces de Hamming de l'algorithme décrit dans [BDGL16, §5] et qui traite le cas de la sphère unitaire euclidienne.

Supposons que nous voulons décoder $\mathbf{y} \in \mathbb{F}_q^n$. Pour chaque $i \in [\![1, s]\!]$ on pose $\mathbf{y}_i$ tels que $\pi^{-1}(\mathbf{y}) = (\mathbf{y}_1, \cdots, \mathbf{y}_s) \in \mathbb{F}_q^{n_1} \times \cdots \times \mathbb{F}_q^{n_s}$. Notre algorithme de décodage consiste alors essentiellement à décoder $(\mathbf{y}_1, \cdots, \mathbf{y}_s)$ dans le code $\mathcal{C}_1 \times \cdots \times \mathcal{C}_s$. Pour cela, nous commençons par définir pour chaque $i \in [\![1, s]\!]$ la fonction de coût φ_i suivante :

$$\begin{array}{rccc} \varphi_i : & \mathcal{C}_i & \longrightarrow & [\![0, n_i]\!] \\ & \mathbf{c}_i & \longmapsto & \Delta(\mathbf{c}_i, \mathbf{y}_i) \end{array} \tag{9.3}$$

Pour tout $i \in [\![1, s]\!]$, on note $m_i := \#\mathcal{C}_i = q^{k_i}$. Nous ordonnons les mots $\mathbf{c}_{i,1}, \mathbf{c}_{i,2}, \cdots, \mathbf{c}_{i,m_i}$ du code $\mathcal{C}_i$ selon leurs distances à $\mathbf{y}_i$ – c'est-à-dire selon leur coût φ_i : $\varphi_i(\mathbf{c}_{i,1}) \leqslant \varphi_i(\mathbf{c}_{i,2}) \leqslant \cdots \leqslant \varphi_i(\mathbf{c}_{i,m_i})$ – et les plaçons dans une liste ordonnée $\mathscr{L}_i$.

L'algorithme de décodage fait appel à l'arbre $\mathscr{T}(\mathscr{L}_1, \ldots, \mathscr{L}_s)$ de profondeur s défini par les s listes ordonnées $\mathscr{L}_1, \ldots, \mathscr{L}_s$. La racine de $\mathscr{T}$ contient le mot vide et les nœuds de profondeur $i \in [\![1, s]\!]$ sont tous les i-uplets $(\mathbf{c}_1, \ldots, \mathbf{c}_i) \in \mathscr{L}_1 \times \cdots \times \mathscr{L}_i$. De plus, pour tout $i \in [\![1, s]\!]$, les fils d'un nœud $\mathbf{x}$ de profondeur $i - 1$ sont tous les $(\mathbf{x}, \mathbf{c}_i)$ où $\mathbf{c}_i \in \mathscr{L}_i$. Ces fils sont rangés selon l'ordre de $\mathscr{L}_i$.

Nous définissons une fonction de coût pour les nœuds de l'arbre $\mathscr{T}$: pour tout $i \in [\![0, s]\!]$ la fonction de coût ψ_i des nœuds de profondeur i est :

$$\begin{array}{rccc} \psi_i : & \mathscr{L}_1 \times \cdots \times \mathscr{L}_i & \longrightarrow & [\![0, n]\!] \\ & (\mathbf{c}_1, \cdots, \mathbf{c}_i) & \longmapsto & \varphi_1(\mathbf{c}_1) + \cdots + \varphi_i(\mathbf{c}_i) + \varphi_{i+1}(\mathbf{f}_{i+1}) + \cdots + \varphi_s(\mathbf{f}_s) \end{array} \tag{9.4}$$

où pour tout $j \in [\![i + 1, s]\!]$, $\mathbf{f}_j$ est le premier mot de la liste $\mathscr{L}_j$. Notons que le coût $\psi_s(\mathbf{c}_1, \cdots, \mathbf{c}_s)$ d'une feuille de $\mathscr{T}$ est la distance du mot de code $(\mathbf{c}_1, \cdots, \mathbf{c}_s) \in \mathcal{C}_1 \times \cdots \times \mathcal{C}_s$ au mot $\pi^{-1}(\mathbf{y})$ que nous voulons décoder.

Finalement, l'algorithme de décodage en liste de $\mathcal{C}$ est décrit par le pseudo-code 9.1.1.

Algorithme 9.1.1 : Décodage en liste d'un code juxtaposé permuté d'ordre s.

Paramètre : $\mathcal{C} := \pi(\mathcal{C}_1 \times \cdots \times \mathcal{C}_s)$: un code juxtaposé permuté d'ordre s.
Entrées : un mot $\mathbf{y} \in \mathbb{F}_q^n$;
une distance de décodage $d \in [\![0, n]\!]$.
Sortie : $\mathscr{L} = \{\mathbf{c} \in \mathcal{C} \ : \ \Delta(\mathbf{c}, \mathbf{y}) \leqslant d\}$.

$\mathscr{L}_{\text{tmp}} \leftarrow \varnothing$;
$(\mathbf{y}_1, \cdots, \mathbf{y}_s) \leftarrow \pi^{-1}(\mathbf{y})$; /* pour tout $i \in [\![1, M]\!], \mathbf{y}_i \in \mathbb{F}_q^{n_i}$ */
pour tout $i \in [\![1, s]\!]$ **faire**
  $\mathscr{L}_i \leftarrow$ trier les mots du code $\mathcal{C}_i$ selon leurs distances à $\mathbf{y}_i$; /* c'est-à-dire selon la fonction de coût φ_i définie par l'équation (9.3) */
finPour
effectuer un parcours en profondeur de $\mathcal{T}(\mathscr{L}_1, \ldots, \mathscr{L}_s)$ en explorant les fils de gauche à droite et en évitant les nœuds $(\mathbf{c}_1, \cdots, \mathbf{c}_i)$ ainsi que tous leurs frères de droite dès lors que $\psi_i(\mathbf{c}_1, \cdots, \mathbf{c}_i) > d$; /* cf. l'équation (9.4) pour la définition de ψ_i */
ajouter toutes les feuilles visitée à $\mathscr{L}_{\text{tmp}}$;
retourner $\mathscr{L} := \pi(\mathscr{L}_{\text{tmp}}) = \{\pi(\mathbf{y}) \ : \ \mathbf{y} \in \mathscr{L}_{\text{tmp}}\}$;

La complexité de l'algorithme 9.1.1 est donnée par la proposition suivante :

Proposition 9.1.2. *Soit $\mathcal{C}$ un code juxtaposé permuté $[n, k]_q$ d'ordre s. Étant donné un mot $\mathbf{y} \in \mathbb{F}_q^n$, la complexité pour calculer la liste $\mathscr{L}$ de* **tous** *les mots du code $\mathcal{C}$ à distance au plus d de $\mathbf{y}$ est de l'ordre de :*

$$T_{\text{juxt}} := O\left(nq^{\frac{k}{s}} + s\ell\right) \tag{9.5}$$

où $\ell := \#\mathscr{L}$.

Démonstration de la proposition 9.1.2.

Des algorithmes de tri permettent d'effectuer l'opération consistant à trier les mots de chaque code $\mathcal{C}_i$ selon la fonction de coût φ_i en un temps de l'ordre de $O\left(\sum_{i=1}^{s} q^{k_i} \log\left(q^{k_i}\right)\right) = O\left(kq^{\frac{k}{s}}\right)$.

L'algorithme 9.1.1 parcourt l'arbre $\mathcal{T}$ dont chaque chemin qui relie la racine à une feuille représente un mot du code juxtaposé. Toutefois, l'arbre n'est pas parcouru intégralement, des sous-arbres entiers sont ignorés. Les chemins ignorés sont ceux qui correspondent à des mots de codes à distance $> d$ de $\mathbf{y}$.

De plus, le nombre de chemins visités est de l'ordre de $O(\#\mathscr{L}) := O(\ell)$ et donc, comme $\mathcal{T}$ est de profondeur s, le coût du parcours en profondeur est de l'ordre de $O(s\ell)$.

Finalement, nous obtenons la complexité de l'algorithme 9.1.1 en additionnant les deux complexités précédemment calculées. □

9.1.3 Application à la recherche de presque-collisions

L'algorithme de décodage 9.1.1 des codes juxtaposés est un très bon candidat pour construire une fonction de hachage floue pour la recherche de presque-collisions. Outre son faible coût d'exécution, il retourne la liste de *tous* les mots de codes à distance au plus d d'un mot quelconque de l'espace ambiant. En outre, d'après le lemme 9.1.3, cette liste a la même taille moyenne qu'avec un décodage idéal d'un code aléatoire.

Lemme 9.1.3. *Pour tout $\mathbf{y} \in \mathbb{F}_q^n$, nous avons :*

$$\mathbb{E}\left(\#\left(\mathcal{B}(\mathbf{y}, d) \cap \mathcal{C}\right)\right) = \mathbb{E}\left(\#\left(\mathcal{B}(\mathbf{y}, d) \cap \mathcal{R}\right)\right) = \frac{T \cdot \sum_{t=0}^{d} \binom{n}{t}}{q^n} \tag{9.6}$$

où $\mathcal{R}$ est code aléatoire composé de T mots tirés uniformément dans $\mathbb{F}_q^n$.

Démonstration du lemme 9.1.3.

Pour tout $t \in [\![0, d]\!]$, l'espérance du nombre de mots du code $\mathcal{C}$ à distance exactement t est :

$$\sum_{\mathbf{t} \in A(t)} \prod_{i=1}^{s} \frac{T_i}{q^{n_i}} \binom{n_i}{t_i} = \left(\prod_{i=1}^{s} \frac{T_i}{q^{n_i}} \right) \cdot \left(\sum_{\mathbf{t} \in A(t)} \prod_{i=1}^{s} \binom{n_i}{t_i} \right)$$

où $A(t) := \{(t_1, \cdots, t_s) \in [\![0, n_1]\!] \times \cdots \times [\![0, n_s]\!] \ : \ t_1 + \cdots + t_s = t\}$.
Or nous avons d'une part :

$$\sum_{\mathbf{t} \in A(t)} \prod_{i=1}^{s} \binom{n_i}{t_i} = \binom{n}{t}$$

et d'autre part, par construction :

$$\prod_{i=1}^{s} \frac{T_i}{q^{n_i}} = \frac{T}{q^n}$$

□

Finalement, l'algorithme de décodage 9.1.1 des codes juxtaposés semble retourner une liste ayant des propriétés similaires à celle retournée par le décodage idéal d'un code aléatoire. En effet, notons $\mathscr{L}_{\mathcal{R}}(\mathbf{y}) := \mathcal{B}(\mathbf{y}, d) \cap \mathcal{R}$ où $\mathcal{R} \subseteq \mathbb{F}_q^n$ est un code aléatoire de taille T. De façon analogue, nous notons $\mathscr{L}_{\mathcal{C}}(\mathbf{y})$ la liste retournée par l'algorithme 9.1.1. D'après ce que nous avons vu précédemment, nous avons :

$$\mathbb{E}\left(\#\mathscr{L}_{\mathcal{R}}(\mathbf{y})\right) = \mathbb{E}\left(\#\mathscr{L}_{\mathcal{C}}(\mathbf{y})\right) \tag{9.7}$$

et nous avons même pour tout $t \in [\![0, d]\!]$:

$$\mathbb{E}\left(\#\left(\mathscr{L}_{\mathcal{R}}(\mathbf{y}) \cap \mathcal{S}(\mathbf{y}, t)\right)\right) = \mathbb{E}\left(\#\mathscr{L}_{\mathcal{C}}(\mathbf{y}) \cap \mathcal{S}(\mathbf{y}, t)\right) \tag{9.8}$$

où $\mathcal{S}(\mathbf{y}, t) := \left\{\mathbf{x} \in \mathbb{F}_q^n \ : \ \Delta(\mathbf{x}, \mathbf{y}) = t\right\}$. Pour rappel, les sources d'aléa dans les équations (9.7) et (9.8) sont le choix du code aléatoire et du code juxtaposé. Ce résultat montre que les itérations de notre algorithme de recherche de presque-collisions ont le même coût que l'on utilise des codes juxtaposés ou des codes aléatoires. Cependant, nous allons voir que la probabilité de succès de ces itérations n'est pas la même dans les deux situations.

Les codes juxtaposés ne permettent pas de trouver des presque-collisions aussi efficacement que les codes aléatoires. Pour le comprendre, il faut commencer par remarquer que les éléments de $\mathscr{L}_{\mathcal{C}}(\mathbf{y})$ ne sont pas indépendants les uns des autres contrairement aux éléments de $\mathscr{L}_{\mathcal{R}}(\mathbf{y})$. Par exemple, soient $\mathbf{c}$ et $\mathbf{c}'$ deux mots tirés uniformément dans la liste de décodage. La probabilité que les n_1 premiers bits de $\pi(\mathbf{c})$ et $\pi(\mathbf{c}')$ soient égaux est plus grande lorsque l'on considère la liste retournée par l'algorithme 9.1.1 plutôt que celle du décodage idéal d'un code aléatoire. Ainsi, la probabilité de collision n'est pas la même avec un code aléatoire et un code juxtaposé.

Lemme 9.1.4. *Soit $d := \lfloor \delta n \rfloor$ et $w := \lfloor \omega n \rfloor$ pour des constantes $\delta \in \left[0, \frac{1}{2}\right]$ et $\omega \in \left[0, \frac{1}{2}\right]$. Soient $\mathbf{x}, \mathbf{y} \in \mathbb{F}_q^n$ tels que $\Delta(\mathbf{x}, \mathbf{y}) = w$. La probabilité de collision P^* de $\mathbf{x}$ et $\mathbf{y}$ est :*

$$\begin{aligned} P^* &:= \mathbb{P}\left(h_{\mathcal{C}}^d(\mathbf{x}) \cap h_{\mathcal{C}}^d(\mathbf{y}) \neq \varnothing\right) && (9.9) \\ &= \tilde{\Omega}\left(\left(\tfrac{n}{s}\right)^{-2s} \cdot q^k \cdot \mathcal{P}_n\left(\leqslant d \mid w\right)\right) && (9.10) \end{aligned}$$

où $\mathcal{C}$ est tiré uniformément parmi les codes juxtaposés permutés $[n, k]_q$ d'ordre s et où $h_{\mathcal{C}}^d(\mathbf{x})$ est l'ensemble des mots de $\mathcal{C}$ à distance d de $\mathbf{x}$ calculé grâce à l'algorithme 9.1.1.

Démonstration du lemme 9.1.4.

Pour tout vecteur $\mathbf{v} \in \mathbb{F}_q^n$, on note $(\mathbf{v}_1, \ldots, \mathbf{v}_s) := \pi^{-1}(\mathbf{v}) \in \mathbb{F}_q^{n_1} \times \cdots \times \mathbb{F}_q^{n_s}$.
Soit un couple $(\mathbf{x}, \mathbf{y}) \in \mathbb{F}_q^n$ tel que $\Delta(\mathbf{x}, \mathbf{y}) = w$. Pour minorer la probabilité de l'évènement $\left\{h_{\mathcal{C}}^d(\mathbf{x}) \cap h_{\mathcal{C}}^d(\mathbf{y}) \neq \varnothing\right\}$, nous regardons la probabilité d'un sous-évènement inclus dans celui-ci. En effet, d'une part, on ne regarde que les cas où le support de $\mathbf{x} - \mathbf{y}$ est équitablement distribué dans les codes $\mathcal{C}_i$; c'est-à-dire que, pour tout $i \in [\![1, s]\!]$, nous supposons $\Delta(\mathbf{x}_i, \mathbf{y}_i) \in \{w', w'+1\}$ où $w' := \left\lfloor \frac{w}{s} \right\rfloor$. D'autre part, nous ne considérons qu'une sous-liste de la liste de décodage retournée par l'algorithme 9.1.1 :

$$g_{\mathcal{C}}^d(\mathbf{x}) := \left\{ \mathbf{c} \in h_{\mathcal{C}}^d(\mathbf{x}) \ : \ \forall i \in [\![1, s]\!] \, \Delta(\mathbf{c}_i, \mathbf{x}_i) \leqslant d' \right\}$$

où $d' := \left\lfloor \frac{\delta n}{s} \right\rfloor$.
Ainsi, le sous-évènement considéré est une intersection d'évènements indépendants ; ce qui nous permet de montrer la minoration suivante :

$$\begin{aligned} P^* &\geqslant \mathbb{P}\left(\forall i \in [\![1, s]\!], \Delta(\mathbf{x}_i, \mathbf{y}_i) \in \{w', w'+1\} \text{ et } g_{\mathcal{C}}^d(\mathbf{x}) \cap g_{\mathcal{C}}^d(\mathbf{y}) \neq \varnothing\right) \\ &\geqslant \mathbb{P}\left(\forall i \in [\![1, s]\!], \Delta(\mathbf{x}_i, \mathbf{y}_i) \in \{w', w'+1\}\right) \\ &\quad \cdot \textstyle\prod_{i=1}^s \mathbb{P}\left(\exists \mathbf{c}_i \in \mathcal{C}_i \ : \ \Delta(\mathbf{x}_i, \mathbf{c}_i) \leqslant d' \text{ et } \Delta(\mathbf{y}_i, \mathbf{c}_i) \leqslant d' \mid \Delta(\mathbf{x}_i, \mathbf{y}_i) \in \{w', w'+1\}\right) \end{aligned} \tag{9.11}$$

Nous avons alors :

$$P^* = \Omega\left(\frac{\binom{n'}{w'}^s}{\binom{n}{w}} \cdot \left(q^{k'} \cdot \mathcal{P}_{n'}\left(\leqslant d' \mid w'\right)\right)^s \right) \tag{9.12}$$

$$= \Omega\left(\frac{\binom{n'}{w'}^s}{\binom{n}{w}} \cdot q^k \cdot \mathcal{P}_{n'}\left(\leqslant d' \mid w'\right)^s \right) \tag{9.13}$$

où $\mathcal{P}_{n'}\left(\leqslant d' \mid w'\right)$ est défini dans la notation 6.2.1.

Sur $\mathbb{F}_q$, le lemme 7.3.3 donne une valeur asymptotique de cette quantité ; cependant, le facteur polynomial n'est pas précisé dans ce lemme. Or nous ne pouvons pas l'omettre dans l'équation (9.13). Une utilisation plus précise de la formule de Stirling dans la démonstration du lemme 7.3.3 nous permet de montrer que :

$$\mathcal{P}_{n'}\left(\leqslant d' \mid w'\right) = \Omega\left(\left(\sqrt{n'}\right)^{-3} \cdot \left(\frac{\left(1 - \frac{1}{q-1}\right)^{2(\delta-\alpha_0)-\omega} \left(\frac{1}{q-1}\right)^{(\delta-\alpha_0)}}{q^{1-(1-\omega)h_q\left(\frac{\alpha_0}{1-\omega}\right)-\omega h_q\left(\frac{\delta-\alpha_0}{\omega}\right)-(\delta-\alpha_0)h_q\left(2-\frac{\omega}{\delta-\alpha_0}\right)}} \right)^{n'} \right) \tag{9.14}$$

où $\alpha_0 \in \left[\max(0, \delta - \omega), \min\left(1-\omega, \delta - \frac{\omega}{2}\right)\right]$ maximise l'expression.
Ainsi, nous avons :

$$P^* = \tilde{\Omega}\left(\frac{\binom{n'}{w'}^s}{\binom{n}{w}} \cdot \left(\sqrt{\tfrac{n}{s}}\right)^{-3s} \cdot q^k \cdot \mathcal{P}_n\left(\leqslant d \mid w\right) \right)$$

□

Remarque 9.1.1. Sur $\mathcal{S}_1^n$, il faut regarder la probabilité que pour tout $i \in [\![1, s]\!]$, $\mathbf{x}_i$ et $\mathbf{y}_i$ vivent sur un voisinage de la sphère $\mathcal{S}_{1/\sqrt{s}}^{n/s}$ et $\Delta(\mathbf{x}, \mathbf{y})$ est dans un voisinage de $\frac{w}{s}$.

Finalement, nous pouvons montrer le théorème suivant :

Théorème 9.1.5. *Soit $T_{\text{Codes}}^{\text{Aléa}}$ la complexité optimale théorique de l'algorithme 6.2.1 (voir le corolaire 7.3.4 pour une approximation asymptotique de cette complexité). L'utilisation*

de codes juxtaposés permutés dans l'algorithme 6.2.1 permet de résoudre le problème des presque-collisions dans $\mathbb{F}_q^n$ en un temps de l'ordre de :

$$T_{\text{juxt}}^{\text{Aléa}} = q^{O\left(\sqrt{n\log(n)}\right)} \cdot T_{\text{Codes}}^{\text{Aléa}} \tag{9.15}$$

Démonstration du théorème 9.1.5.

D'après la proposition 9.1.2, pour chaque itération de l'algorithme 6.2.1, le coût moyen pour décoder chaque éléments de $\mathscr{L}$ est $\left(nq^{\frac{k}{s}} + s\ell\right) \cdot L \cdot \ell$ où $L := \#\mathscr{L}$ et ℓ est la taille moyenne des listes de décodage. De plus, pour chaque itération, le nombre moyen de couples qui entrent en collision et dont on doit vérifier la proximité est $\frac{(L\cdot\ell)^2}{q^k}$. Ainsi, la complexité moyenne de l'algorithme 6.2.1 instancié avec des codes juxtaposés est :

$$\begin{aligned} T_{\text{juxt}}^{\text{Aléa}} &= \frac{\left(nq^{\frac{k}{s}} + s\ell\right) \cdot L \cdot \ell + \frac{(L\cdot\ell)^2}{q^k}}{P*} \\ &= q^{\frac{k}{s}} \cdot \frac{L \cdot \ell + \frac{(L\cdot\ell)^2}{q^k}}{P*} \end{aligned}$$

Pour un rayon de décodage d donné, nous avons vu que la taille moyenne ℓ des listes de décodage est la même avec les codes juxtaposés permutés et les codes aléatoire. Finalement, en utilisant le lemme 9.1.4, nous pouvons montrer que :

$$\begin{aligned} T_{\text{juxt}}^{\text{Aléa}} &= \tilde{O}\left(q^{\frac{k}{s}} \cdot \left(\tfrac{n}{s}\right)^{2s} \cdot T_{\text{Codes}}^{\text{Aléa}}\right) \\ &= \tilde{O}\left(q^{\gamma(s)} \cdot T_{\text{Codes}}^{\text{Aléa}}\right) \end{aligned}$$

où $\gamma(s) := \frac{k}{s} + 2s\log_q\left(\frac{n}{s}\right)$. Une étude de la dérivée de γ montre que le minimum de γ est atteint lorsque :

$$s = \sqrt{\frac{-k\log(q)}{2 - 2\log\left(\frac{n}{s}\right)}}$$

La méthode du point fixe nous permet alors de montrer que la valeur optimale de s est de l'ordre de $O\left(\sqrt{\frac{n}{\log(n)}}\right)$. Pour une telle valeur de s, l'exposant $\gamma(s)$ est de l'ordre de $O\left(\sqrt{n\log(n)}\right)$. □

Dans le théorème 9.1.5, le facteur $q^{O\left(\sqrt{n\log(n)}\right)}$ est sous-exponentiel en n. Les codes juxtaposés permutés permettent donc d'atteindre asymptotiquement la complexité théorique donnée par le corollaire 7.3.4 pour la recherche de presque-collisions. Cependant, ce facteur reste super-polynomial ; ce qui rend les codes juxtaposés permutés bien moins intéressants en pratique.

Notons que le facteur super-polynomial que nous obtenons avec les codes juxtaposés est plus grand que celui que nous obtenons avec la méthode de recherche de presque-collisions de May et Ozerov [MO15] (cf. théorème 5.5.3 du chapitre 5). La méthode May-Ozerov est plus efficace car elle factorise les opérations de décodage.

9.2 La recherche de presque-collisions avec des codes polaires

Dans cette section, nous analysons expérimentalement des décodages en liste de codes polaires pour savoir si ceux-ci sont de meilleures fonctions de hachage floues que les décodages en liste de codes juxtaposés permutés.

Nous avons déjà présenté les codes polaires et leurs décodages dans la section 1.4 du chapitre 1. Dans la suite, nous utilisons essentiellement le décodage en liste des codes

polaires proposé par Tal et Vardy dans [TV15].

Soit un code polaire $\mathcal{C}[n,k]$. Contrairement aux codes juxtaposés, le décodage de Tal et Vardy du code $\mathcal{C}$ n'est pas paramétré par un rayon de décodage d mais par la taille ℓ de la liste retournée. Les mots de codes retournés par le décodage d'un mot $\mathbf{y} \in \mathbb{F}_q^n$ ne sont donc pas spécialement localisés dans une boule centrée en $\mathbf{y}$. Ces mots ne sont pas non plus les ℓ mots de code *les plus proches* de $\mathbf{y}$. En fait, nous pouvons seulement dire que les mots de la liste de décodage se situent typiquement au voisinage d'une sphère centrée en $\mathbf{y}$ et de rayon de l'ordre de $d^-_{\mathrm{GV}}(2^k)(1+o(1))$.

Dans l'algorithme 6.2.1 de recherche de presque-collisions, la longueur n et la dimension k du code que nous devons utiliser sont fixées pour toutes les itérations. Or pour un couple $[n,k]$ donné, il n'existe qu'un seul code polaire linéaire de performance optimale ; du moins si l'on suit la construction décrite dans la section 1.4. Nous allons donc intégrer de l'aléa dans nos itérations en choisissant des cosets permutés du code polaire $\mathcal{C}$; c'est-à-dire qu'à chaque itération, nous choisissons uniformément une permutation $\pi \in S_n$ et un vecteur $\mathbf{v} \in \mathbb{F}_q^n$ et nous considérons le code $\pi(\mathbf{v}+\mathcal{C}) := \{\pi(\mathbf{v}+\mathbf{c}) \ : \ \mathbf{c} \in \mathcal{C}\}$.

Pour analyser l'impact du décodage en liste des cosets permutés d'un code polaire dans l'algorithme 6.2.1, nous mesurons expérimentalement la probabilité suivantes :

$$P^*(k,\ell) := \mathbb{P}\left(h_\ell(\mathbf{x}^*) \cap h_\ell(\mathbf{y}^*) \neq \varnothing\right) \tag{9.16}$$

où h_ℓ est un décodage en liste retournant ℓ mots d'un code choisi uniformément parmi les cosets permutés d'un code polaire $[n,k]$ et $\mathbf{x}^*$ et $\mathbf{y}^*$ sont tels que $\Delta(\mathbf{x}^*,\mathbf{y}^*) = w$.

Rappelons que n et w sont des paramètres du problème de presque-collisions que nous cherchons à résoudre ; n est la longueur des mots de la liste et w est la distance maximale des mots proches que nous recherchons dans cette liste. Ainsi, k et ℓ seront les deux paramètres à optimiser.

En adaptant le lemme 6.1.4, nous pouvons utiliser l'estimation de $P^*(k,\ell)$ pour donner la complexité moyenne $T^{\text{Aléa}}_{\text{polaire}}$ de l'algorithme 6.2.1 instancié avec des codes polaires :

$$T^{\text{Aléa}}_{\text{polaire}}(k,\ell) = O\left(\frac{L\ell + \frac{L^2\ell^2}{2^k}}{P^*(k,\ell)}\right) \tag{9.17}$$

où L est la taille de la liste dans laquelle nous recherchons les presque-collisions.

Nous considérons un problème de presque-collisions où des couples à distance $w := \lfloor 0.1n \rfloor$ sont recherchés dans une liste de $L := 2^{n/64}$ éléments de $\mathbb{F}_2^n$. La figure 9.1 représente la complexité $T^{\text{Aléa}}_{\text{polaire}}(k,\ell)$ de notre méthode en fonction de la taille ℓ de la liste de décodage du code polaire pour $n = 1024$ et $k = 32$. La figure 9.2 représente cette même complexité en fonction de la dimension k du code polaire utilisé pour $n = 1024$ et $\ell = 16$.

Finalement, pour différentes longueurs n, nous avons optimisé les paramètres k et ℓ pour obtenir la complexité optimale de notre méthode :

$$T^{\text{Aléa}}_{\text{polaire}} := \inf_{k,\ell}\left(T^{\text{Aléa}}_{\text{polaire}}(k,\ell)\right) \tag{9.18}$$

Nous avons alors comparé $T^{\text{Aléa}}_{\text{polaire}}$ avec la complexité $T^{\text{Aléa}}_{\text{Codes}}$ (cf. théorème 6.2.6). La figure 9.3 représente le facteur multiplicatif entre ces deux complexités. Notons que ce facteur est bien meilleur que celui que nous avons obtenu dans la section précédente avec les codes juxtaposés ou même que le facteur super-polynomial de la méthode de May et Ozerov (cf. section 5.5 chapitre 5). Nous conjecturons même le résultat suivant :

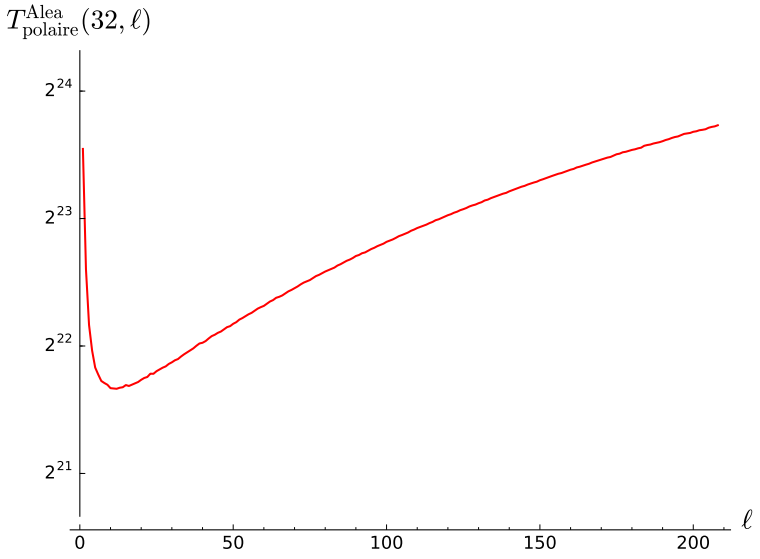


Figure 9.1 – Complexité de l'algorithme 6.2.1 instancié avec des cosets permutés d'un code polaire $[1024, 32]$ que l'on décode en listes de taille ℓ ($n = 1024$, $L = 2^{16}$ et $w = 102$).

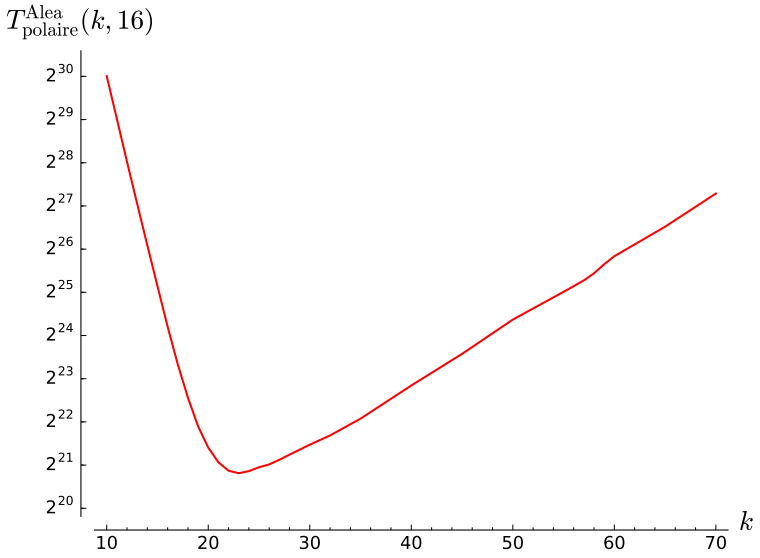


Figure 9.2 – Complexité de l'algorithme 6.2.1 instancié avec des cosets permutés d'un code polaire $[1024, k]$ que l'on décode en listes de taille $\ell = 16$ ($n = 1024$, $L = 2^{16}$ et $w = 102$).

Conjecture 9.2.1. *L'utilisation de cosets permutés de codes polaires dans notre algorithme 6.2.1 n'engendre qu'un facteur polynomial par rapport à la complexité théorique obtenue avec des décodeurs idéaux de codes aléatoires :*

$$T^{\text{Aléa}}_{\text{polaire}} = \text{poly}\,(n) \cdot T^{\text{Aléa}}_{\text{Codes}} \tag{9.19}$$

où $\text{poly}\,(n)$ *est un polynôme en* n.

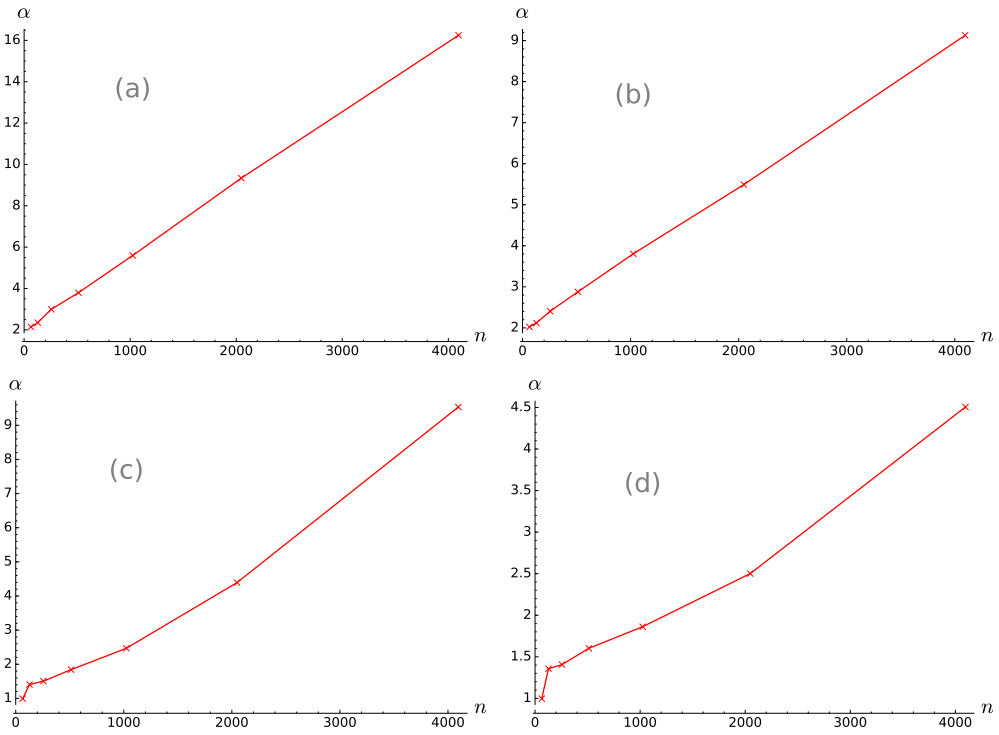


Figure 9.3 – Facteur multiplicatif α tel que $T^{\text{Aléa}}_{\text{polaire}} = O\big(\alpha \cdot T^{\text{Aléa}}_{\text{Codes}}\big)$ en fonction de n. Les instances du problème de presque-collisions que nous considérons sont : (a) $L = 2^{n/64}$ et $w = \lfloor 0.1n \rfloor$, (b) $L = 2^{n/128}$ et $w = \lfloor 0.1n \rfloor$, (c) $L = 2^{n/64}$ et $w = \lfloor 0.01n \rfloor$ et (d) $L = 2^{n/128}$ et $w = \lfloor 0.01n \rfloor$.

Pour finir, il faut noter que dans ce chapitre, nous nous sommes concentrés sur l'algorithme 6.2.1. Cependant, nous pouvons également utiliser le décodage en liste des codes polaires dans l'algorithme hybride 8.1.1 du chapitre 8.

Conclusion et perspectives

Notre travail s'est principalement axé autour du problème de recherche de presque-collisions. Nous proposons une nouvelle solution pour résoudre ce problème qui surpasse celles proposées dans la littérature. De plus, nous donnons une description générique de notre méthode, ce qui permet de l'appliquer dans divers modèles et contextes. Par exemple, nous traitons aussi bien la recherche de presque-collisions sur la sphère euclidienne que dans les espaces de Hamming binaires et non-binaires. Notre utilisation du décodage en liste de Tal et Vardy des codes polaires [TV15] pour construire des fonctions de hachage floues nous a permis de réduire le surcoût super-polynômial dont souffrait la plupart des méthodes proposées jusqu'ici telles que la méthode de May et Ozerov [MO15].

Avec la menace que représentent les ordinateurs quantiques pour la protection de l'information, une nouvelle cryptologie a dû s'instaurer. Celle-ci se veut être une alternative au paradigme de sécurité fondé sur la théorie des nombres. En 2017, le NIST (*National Institute of Science and Technology*) a lancé un appel pour définir les futurs standards cryptographiques qui devront résister aux ordinateurs quantiques. De nombreuses propositions du monde entier s'affrontent aujourd'hui dans cette compétition. Ainsi, plusieurs grandes familles de crypto-systèmes sont de potentiels candidats pour protéger nos futurs systèmes de communication contre la menace des ordinateurs quantiques. Parmi elles, nous pouvons citer les deux les plus populaires, à savoir la cryptographie fondée sur les réseaux euclidiens et celle fondée sur les codes correcteurs d'erreurs. Ces deux types de cryptographie reposent sur des problèmes mathématiques jugés calculatoirement difficiles, même pour un potentiel attaquant qui posséderait un ordinateur quantique. Par exemple, ces problèmes peuvent être :

- le décodage générique de codes linéaires (binaires, q-aires ou même en métrique rang) [Pra62, Ste88, Dum91, 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] ;
- la recherche de presque-collisions dans des fonctions de hachage cryptographiques [LMRS12, Leu13].

Cette liste énumère des problèmes qui ont tous un point commun : ils font appel à un problème de recherche de couples proches. Nous pouvons donc améliorer les solutions existantes en utilisant nos travaux sur la recherche de presque-collisions. Nous nous sommes particulièrement concentrés sur le décodage générique dans les espaces de Hamming binaires et non-binaires. Nous améliorons ainsi les solutions existantes.

Nous avons aussi appliqué notre recherche de presque-collisions au problème SVP. Toutefois, les paramètres du problème de presque-collisions qui interviennent dans la résolution de ce problème sont tels que notre méthode n'apporte aucune amélioration par rapport à celle de [BDGL16].

Un autre problème auquel nous nous sommes intéressés durant cette thèse est le problème de reconnaissance de codes LDPC. Nous avons proposé une nouvelle méthode qui interpole les méthodes existantes et qui les améliore [CT17, CT19b]. Plus exactement, notre méthode recherche des équations de parité de poids faible dans un code inconnu à partir d'une liste de mots de code bruités. Ainsi, notre méthode de reconnaissance de codes s'applique à n'importe quel code possédant des équations de parité creuses ou faisant intervenir de tels codes. Nous pouvons ainsi reconnaître des codes LDPC bien sûr, mais aussi des codes polaires, des turbo-codes, des codes convolutifs, des codes raptors, des codes fontaines...

Nous avons aussi remarqué que la recherche d'équations de parité creuses dans un code peut s'effectuer en résolvant un problème LPN. Ainsi, la méthode BKW [BKW03] peut être appliquée de la même façon que dans [EKM17] pour résoudre le problème de reconnaissance de codes.

Les perspectives de cette thèse

La reconnaissance de codes. Nous avons montré que nous pouvions utiliser les techniques de résolution du problème LPN pour reconstruire des codes LDPC ; notamment les méthodes de [EKM17]. Il est également possible de combiner ces techniques avec celles de Both et May pour le décodage générique [BM18] ; notamment la partie où il est recherché une somme creuse de colonnes dans une matrice. Une telle solution mérite d'être analysée plus en détail pour mesurer le gain qu'elle apporte.

D'autre part, les codes LDPC non-binaires sont de plus en plus utilisés en pratique. Nous aimerions donc généraliser nos méthodes de reconnaissance aux codes définis sur les corps finis de taille > 2 ; d'autant plus qu'une grande partie des algorithmes présentés dans ce manuscrit s'applique déjà aux espaces non-binaires.

Le décodage générique. L'optimisation de l'algorithme de Both et May pour le décodage générique sur $\mathbb{F}_q$ est très coûteuse lorsque la profondeur m de la récursion est trop élevée. Dans le cas binaire, nous devons choisir $m = 4$ pour obtenir une amélioration par rapport aux résultats présentés dans [BM18]. Dans les cas non-binaires, le choix $m = 3$ suffit pour atteindre les meilleures complexités de la littérature. Nous pouvons tout de même tenter d'atteindre des profondeurs m plus grandes et ainsi donner des complexités encore meilleures.

La recherche de presque-collisions. Le problème de recherche du plus proche voisin (*Nearest-Neighbors Search problem*) est un problème très étudié dans différents domaines d'informatique. Nous avons exploré des solutions de la littérature qui correspondaient le plus à notre modèle telles que l'approche LSH [IM98], l'utilisation de codes comme fonctions de hachage floues [GMO10, Dub10, BDGL16] ou encore l'approche de [MO15]. Cependant, il y a une solution que nous n'avons pas explorée, à savoir l'approche *data-dependent* [AINR14, AR15]. Cette approche utilise la structure des listes dans lesquelles nous recherchons nos couples proches. Nous pouvons alors nous demander si cette méthode ne peut pas être généralisée aux contextes qui nous intéressent pour améliorer le décodage générique ou encore la résolution du problème SVP.

D'autre part, nous avons proposé d'utiliser le décodage en liste de Tal et Vardy de codes polaires [TV15] pour construire des fonctions de hachage floues et ainsi améliorer la recherche de presque-collisions. Cependant, nos conclusions se basent sur des résultats expérimentaux et non sur une analyse de la structure des codes polaires. Une étude plus poussée pourrait amener à une mesure plus précise de la complexité de notre méthode ; notamment, nous pourrions démontrer la conjecture 9.2.1.

En outre, nous avons proposé une nouvelle façon de construire des codes polaires et plus généralement des codes $U|U+V$ récursifs. Les codes $U|U+V$ récursifs sont notamment définis par un ensemble de codes constituants. Nous avons vu que pour les codes polaires, ces codes sont les codes triviaux de longueur 1. Dans notre construction des codes $U|U+V$ récursifs, nous proposons de décoder exhaustivement les codes constituants de dimensions ou codimensions faibles. Au départ, nous avions choisi les codes constituants aléatoirement. Nous nous sommes alors aperçus que ce n'était pas un choix judicieux : par exemple, choisir des codes de Reed-Muller raccourcis est un meilleur choix. Nous pouvons alors nous demander si une autre famille de codes constituants permettrait d'améliorer davantage les performances de nos codes $U|U+V$ récursifs.

Dans le chapitre 7, nous avons appliqué notre méthode de recherche de presque-collisions à des espaces métriques bien particuliers : la sphère euclidienne ainsi que les espaces de Hamming binaires et non-binaires. Nous pouvons cependant élargir notre étude à d'autres espaces métriques. Par exemple, nous n'avons pas exploré le comportement de notre méthode dans les espaces de Grassmann. La recherche de presque-collisions dans les espaces de Grassmann pourrait être utilisée pour résoudre certains problèmes liés à la cryptographie fondée sur les codes en métrique rang.

Pour finir, revenons sur notre algorithme hybride 8.1.1 du chapitre 8. Celui-ci concerne les espaces binaires et non-binaires. Par rapport à notre méthode des codes originale, nous ajoutons une étape de pré-calcul sur les listes dans lesquelles nous cherchons les couples proches. Cette étape a pour but de réduire la dimension du problème de recherche de presque-collisions ; cette dimension étant l'une des raisons principales de la difficulté du problème. Sur $\mathbb{F}_q$, ce pré-calcul consiste simplement à projeter les éléments des listes sur un ensemble de positions fixées. Dans cette thèse, nous n'avons pas exploité cette idée pour d'autres espaces métriques. La technique consistant à changer de géométrie pour nous plonger dans un espace de dimension plus petite peut probablement être exploitée pour d'autres espaces métriques tels que la sphère euclidienne où peut-être même les espaces de Grassmann.

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 CheckK 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, Bardet Magali, 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.

[AI17] Alexandr Andoni and Piotr Indyk. *Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors)*, chapter

43. Nearest neighbors in high-dimensional spaces. CRC Press, Boca Ratin, FL, 3rd edition, 2017.

[AINR14] Alexandr Andoni, Piotr Indyk, Huy L. Nguyen, and Ilya Razenshteyn. Beyond locality-sensitive hashing. In *Proceedings of the Twenty-fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 1018–1028, Philadelphia, PA, USA, 2014. Society for Industrial and Applied Mathematics.

[AJPS17] Divesh Aggarwal, Antoine Joux, Anupam Praksh, and Miklos Santha. Mersenne-756839. First round submission to the NIST post-quantum cryptography call, November 2017.

[AR15] Alexandr Andoni and Ilya P. Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 793–801. ACM, 2015.

[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.

[Bar05] Johann Barbier. Reconstruction of turbo-code encoders. In *Proceedings of SPIE*, volume 5819, pages 463–473, 2005.

[Bar07] Johann Barbier. *Analyse de canaux de communication dans un contexte non coopératif*. PhD thesis, École Polytechnique, November 2007.

[BBB+17a] 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+17b] Magali Bardet, Élise Barelli, Olivier Blazy, Rodolfo Canto Torres, Alain Couvreur, Phillipe Gaborit, Ayoub Otmani, Nicolas Sendrier, and Jean-Pierre Tillich. BIG QUAKE. https://bigquake.inria.fr, 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, 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.

[BE98] Simon Berkovich and Eyas El-Qawasmeh. Reversing the error-correction scheme for a fault-tolerant indexing. In *Proceedings DCC '98 Data Compression Conference*, page 527, March 1998.

[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. *Proceedings of ICC '93 - IEEE International Conference on Communications*, 2 :1064–1070 vol.2, 1993.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer-Verlag, New York, 2006.

[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.

[BL16] Anja Becker and Thijs Laarhoven. Efficient (Ideal) Lattice Sieving Using Cross-Polytope LSH. In David Pointcheval, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *Progress in Cryptology – AFRICACRYPT 2016*, 2016.

[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. on line proceedings, see `http://wcc2017.suai.ru/Proceedings_WCC2017.zip`.

[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.

[BSH06] Johann Barbier, Guillaume Sicot, and Sébastien Houcke. Algebraic Approach of the Reconstruction of Linear and Convolutional Error Correcting Codes. In *World Academy of Science, Engineering and Technology*, volume 16, pages 66–71, November 2006.

[BT14] Marion Bellard and Jean-Pierre Tillich. Detecting and reconstructing an unknown convolutional code by counting collisions. In *Proc. IEEE Int. Symposium Inf. Theory - ISIT*, pages 2967–2971, Honolulu, Hawaii, USA, July 2014. IEEE.

[BY00] Ángela I. Barbero and Øyvind Ytrehus. Modifications of the rao-nam cryptosystem. In Johannes Buchmann, Tom Høholdt, Henning Stichtenoth, and Horacio Tapia-Recillas, editors, *Coding Theory, Cryptography and Related Areas : Proceedings of an International Conference on Coding Theory, Cryptography and Related Areas, held in Guanajuato, Mexico, in April 1998*, pages 1–12, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[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.

[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.

[CF09a] Mathieu Cluzeau and Matthieu Finiasz. Reconstruction of Punctured Convolutional Codes. In *Information Theory Workshop (ITW)*, Taormina, Italy, October 2009. IEEE.

[CF09b] Mathieu Cluzeau and Matthieu Finiasz. Recovering a Code's Length and Synchronization from a Noisy Intercepted Bitstream. In *Proc. IEEE Int. Symposium Inf. Theory - ISIT*, pages 2737–2741, Seoul, Korea, 2009. IEEE.

[CFT10] Mathieu Cluzeau, Matthieu Finiasz, and Jean-Pierre Tillich. Methods for the Reconstruction of Parallel Turbo Codes. In *Proc. IEEE Int. Symposium Inf. Theory - ISIT*, pages 2008–2012, Austin, Texas, USA, June 2010. IEEE.

[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.

[Cha09] Christophe Chabot. Reconstruction of families of codes. application to cyclic codes. In *Workshop on Coding and Cryptography (WCC)*, Ullensvang, Norway, 2009.

[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.

[CS09] Maxime Côte and Nicolas Sendrier. Reconstruction of convolutional codes from noisy observation. In *Proc. IEEE Int. Symposium Inf. Theory - ISIT*, pages 546–550, Seoul, Korea, 2009. IEEE.

[CS10] Maxime Côte and Nicolas Sendrier. Reconstruction of a turbo-code interleaver from noisy observation. In *Proc. IEEE Int. Symposium Inf. Theory - ISIT*, pages 2003–2007, Austin, Texas, USA, June 2010. IEEE.

[CS15] Tofar Chang and Yu Su. Dynamic Weighted Bit-Flipping Decoding Algorithms for LDPC Codes. *IEEE Transactions on Communications*, 63 :3950–3963, November 2015.

[CT08] Mathieu Cluzeau and Jean-Pierre Tillich. On the Code Reverse Engineering Problem. In *Proc. IEEE Int. Symposium Inf. Theory - ISIT*, pages 634–638, Toronto, Canada, 2008. IEEE.

[CT17] Kévin Carrier and Jean-Pierre Tillich. Identifying an unknown code by partial gaussian elimination. In *WCC Workshop on Coding and Cryptography*, September 2017.

[CT19a] 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.

[CT19b] Kévin Carrier and Jean-Pierre Tillich. Identifying an unknown code by partial gaussian elimination. *Designs, Codes and Cryptography*, 87 :685–713, 2019.

[DA19] Thomas Debris-Alazard. *Code-based Cryptography : New Approaches for Design and Proof ; Contribution to Cryptanalysis*. Theses, Sorbonne Universites, UPMC University of Paris 6, December 2019.

[dBDJdW18] Koen de Boer, Léo Ducas, Stacey Jeffery, and Ronald de Wolf. Attacks on the AJPS mersenne-based cryptosystem. In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography 2018*, volume 10786 of *LNCS*, pages 101–120, Fort Lauderdale, FL, USA, April 2018. Springer.

[Dem08] Michel Demazure. *Cours d'algèbre. Primalité. Divisibilité. Codes*. Cassini, 2008.

[Des07] Amit Jayant Deshpande. *Sampling-based algorithms for dimension reduction.* PhD thesis, Massachusetts Institute of Technology, 2007.

[DH76] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6) :644–654, 1976.

[DH07] Janis Dingel and Joachim Hagenauer. Parameter Estimation of a Convolutional Encoder from Noisy Observation. In *Proc. IEEE Int. Symposium Inf. Theory - ISIT*, pages 1776–1780, Nice, France, June 2007. IEEE.

[DHL+94] Danny Dolev, Yuval Harari, Nathan Linial, Noam Nisan, and Michal Parnas. Neighborhood preserving hashing and approximate queries. In Daniel Dominic Sleator, editor, *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. 23-25 January 1994, Arlington, Virginia, USA.*, pages 251–259. ACM/SIAM, 1994.

[DHS00] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification.* WileyInterscience, 2nd edition edition, 2000.

[DRU06] Changyan Di, Thomas J. Richardson, and Rüdiger L. Urbanke. Weight distribution of low-density parity-check codes. *IEEE Trans. Information Theory*, 52(11) :4839–4855, 2006.

[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.

[Dub10] Moshe Dubiner. Bucketing coding and information theory for the statistical high-dimensional nearest-neighbor problem. *IEEE Trans. Inform. Theory*, 56(8) :4166–4179, 2010.

[Dum89] Ilya 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.

[DW82] Luc Devroye and Terry J. Wagner. *Nearest neighbor methods in discrimination*, volume 2 of *Handbook of Statistics*, chapter 8, pages 193–197. North-Holland, Amsterdam, P.R. Krishnaiah and L.N. Kanal editors, edition, 1982.

[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.

[EDG14] Morteza Esmaeili, Mohammad Dakhilalian, and T. Aaron Gulliver. New secure channel coding scheme based on randomly punctured quasi-cyclic-low density parity check codes. *IET Communications*, 8(14) :2556–2562, 2014.

[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.

[FHS+17] Tomás Fabsic, Viliam Hromada, Paul Stankovski, Pavol Zajac, Qian Guo, and Thomas Johansson. A Reaction Attack on the QC-LDPC McEliece Cryptosystem. In *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017, Utrecht, The Netherlands, June 26-28, 2017, Proceedings*, volume 10346 of *LNCS*, pages 51–68. Springer, 2017.

[Fil97] Eric Filiol. Reconstruction of Convolutional Encoders over $GF(q)$. In *Cryptography and Coding : 6th IMA Int. Conf.*, pages 101–109, 1997.

[Fil00] Eric Filiol. Reconstruction of punctured convolutional encoders. In *Int. Symp. on Information Theory and Applications (ISITA)*, 2000.

[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. *IEEE Transactions on Information Theory*, 62, May 2014.

[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.

[For01] George David Forney. Codes on graphs : Normal realizations. *IEEE Trans. Inform. Theory*, 47(2) :520–548, 2001.

[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.

[FS14] Marco Ferrante and Monica Saltalamacchia. The coupon collectors's problem. *MATerials MATemàtics*, 2014 :35, May 2014.

[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.

[GG91] Allen Gersho and Robert M. Gray. *Vector Quantization and Data Compression.* Kluwer, Norwell, 1991.

[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.

[GMO10] Daniel M. Gordon, Victor S. Miller, and Peter Ostapenko. Optimal hash functions for approximate matches on the n-cube. *IEEE Trans. Inform. Theory*, 56(3) :984 – 991, March 2010.

[Goo57] Irving John Good. Saddle-point methods for the multinomial distribution. *Ann. Math. Statist.*, 28(4) :861–881, December 1957.

[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] Lov 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.

[GS91] Danièle Gardy and Patrick Solé. Saddle point techniques in asymptotic coding theory. In Gérard D. Cohen, Simon Litsyn, Antoine Lobstein, and Gilles Zémor, editors, *Algebraic Coding, First French-Soviet Workshop, Paris, France, July 22-24, 1991, Proceedings*, volume 573 of *LNCS*, pages 75–81. Springer, 1991.

[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 $\mathbb{F}_q$ 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, Masoumeh 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] Reza Hooshmand, Masoumeh Koochak Shooshtari, Taraneh Eghlidos, and Mohammad Reza 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.

[HZ07] Fenghua Huang, Zhitao Wang and Yiyu Zhou. A methode for blind recognition of convolution code based on Euclidean algorithm. In *International Conference on Wireless Communications, Networking and Mobile Computing*, pages 1414–1417, Shanghai, China, September 2007.

[IEE09] IEEE Std 802.11n. IEEE Standard for Information technology—Telecommunications and information exchange between systems Local and metropolitan area networks—Specific requirements. Part 11 : Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 5 : Enhancements for Higher Throughput. October 2009.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors : Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 604–613, New York, NY, USA, 1998. ACM.

[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, École 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 Reudiger L. Urbanke. Spatially coupled ensembles universally achieve capacity under belief propagation. *IEEE Transactions on Information Theory*, 59(12) :7761–7813, December 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.

[LdW15] Thijs Laarhoven and Benne de Weger. Faster sieving for shortest lattice vectors using spherical locality-sensitive hashing. volume 9230, pages 101–118, August 2015.

[Leo82] Jeffrey Leon. Computing automorphism groups of error-correcting codes. *IEEE Trans. Inform. Theory*, 28(3) :496–511, 1982.

[Leu13] Gaëtan Leurent. Time-memory trade-offs for near-collisions. In *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, volume 8424 of *LNCS*, pages 205–218. Springer, 2013.

[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.

[LMRS12] Mario Lamberger, Florian Mendel, Vincent Rijmen, and Koen Simoens. Memoryless near-collisions via coding theory. *Des. Codes Cryptogr.*, 62(1) :1–18, January 2012.

[LSLZ04] Peizhong Lu, Li Shen, Xiangyang Luo, and Yan Zou. Blind Recognition of Punctured Convolutional Codes. In *Proc. IEEE Int. Symposium Inf. Theory - ISIT*, page 457, Chicago, USA, July 2004. IEEE.

[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.

[LYSL10] Wang Lei, Hu Yihua, Hao Shiqi, and Qi Lin. The Method of Estimating the Length of Linear Cyclic Code Based on the Distribution of Code Weight. In *2nd International Conference on Information Science and Engineering (ICISE)*, pages 2459–2462. IEEE, 2010.

[Mac02] David J. C. MacKay. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, New York, NY, USA, 2002.

[Mar09] Mélanie Marazin. *Reconnaissance en aveugle de codeur à base de code convolutif : Contribution à la mise en oeuvre d'un récepteur intelligent.* PhD thesis, Université de Bretagne Occidentale, December 2009.

[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.

[MEP17] Benham Mafakheri, Taraneh Eghlidos, and Hossein Pilaram. An efficient secure channel coding scheme based on polar codes. *The ISC International Journal of Information Security*, 9(2) :13–20, 2017.

[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.

[MGB12] Mélanie Marazin, Roland Gautier, and Gilles Burel. Algebraic method for blind recovery of punctured convolutional encoders from an erroneous bitstream. *IET Signal Processing*, 6(2) :122–131, April 2012.

[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.

[MP69] Marvin Minsky and Seymour A. Papert. *Perceptrons : An Introduction to Computational Geometry.* The MIT Press, 1969.

[MRAS00] Chris Monico, Joachim Rosenthal, and Amin A. Shokrollahi. 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.

[NAF11] Ali Naseri, Omid Azmoon, and Samad Fazeli. Blind Recognition Algorithm of Turbo Codes for Communication Intelligence Systems. *International Journal of Computer Science Issues*, 8(1) :68–72, November 2011.

[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.

[NV08] Phong Nguyen and Thomas Vidick. Sieve algorithms for the shortest vector problem are practical. *Journal of Mathematical Cryptology*, 2 :181–207, July 2008.

[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 : 11^{th} 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.

[Ric95] B. Rice. Determining the parameters of a rate $\frac{1}{n}$ convolutional encoder over $GF(q)$. In *Third International Conference on Finite Fields and Applications*, Glasgow, 1995.

[RJ18] Dao Ren and Sha Jin. Improved Gradient Descent Bit Flipping Decoder for LDPC Codes on BSC Channel. *IEICE Electronics Express*, 15, April 2018.

[RN86] Thammavarapu R. N. Rao and Kil-Hyun Nam. Private-key algebraic-coded cryptosystems. In *Advances in Cryptology - CRYPTO'86*, volume 263 of *LNCS*, pages 35–48. Springer, 1986.

[Ros06] Gail L. Rosen. Examining coding structure and redundancy in DNA. *IEEE Engineering in Medecine and Biology*, 25(1) :62–68, January 2006.

[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.

[SAEA09] Ali Akbar Sobhi Afshar, Taraneh Eghlidos, and Mohammad Reza Aref. Efficient secure channel coding based on quasi-cyclic low-density parity-check codes. *IET Communications*, 3(2) :279–292, 2009.

[SDI05] Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk. *Nearest-neighbor methods in learning and vision : Theory and practice*. MIT Press, 2005.

[Sha48] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3) :379–423, 1948.

[SHB09] Guillaume Sicot, Sébastien Houcke, and Johann Barbier. Blind Detection of Interleaver Parameters. *Signal Processing*, 89(4) :450–462, 2009.

[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 Ron Simon. On the power of quantum computation. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, SFCS '94, pages 116–123, USA, 1994. IEEE Computer Society.

[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.

[ST87] René Struik and Johan van Tilburg. The Rao-Nam scheme is insecure against a chosen-plaintext attack. In *Advances in Cryptology - CRYPTO'87*, volume 293 of *LNCS*, pages 445–457. Springer, 1987.

[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 M. Tanner. A Recursive Approach to Low Complexity Codes. *IEEE Transactions on information theory*, 27(5), September 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.

[TTS14] Jean-Pierre Tillich, Audrey Tixier, and Nicolas Sendrier. Recovering the interleaver of an unknown turbo-code. In *Proc. IEEE Int. Symposium Inf. Theory - ISIT*, pages 2784–2788, Honolulu, Hawaii, USA, July 2014. IEEE.

[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.

[Val77] Leslie G. Valiant. Graph-theoretic arguments in low-level complexity. In *Mathematical Foundations of Computer Science 1977, 6th Symposium, Tatranska Lomnica, Czechoslovakia, September 5-9, 1977, Proceedings*, volume 53 of *LNCS*, pages 162–176. Springer, 1977.

[Val01] Antoine Valembois. Detection and recognition of a binary linear code. *Discrete Applied Mathematics*, 111 :199–218, July 2001.

[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.

[WYY10] Jiafeng Wang, Yang Yue, and Jun Yao. A Method of Blind Recognition of Cyclic Code Generator Polynomial. In *6th International Conference on Wireless Communications Networking and Mobile Computing (WiCOM)*. IEEE, 2010.

[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.

[YVK14] Arti D. Yardi, Saravanan Vijayakumaran, and Animesh Kumar. Blind reconstruction of binary cyclic codes. In *Proc of the European Wireless 2014*, 2014.

[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.

[ZHSY13] Jing Zhou, Zhiping Huang, Shaojing Su, and Shaowu Yang. Blind recognition of binary cyclic codes. *EURASIP Journal on Wireless Communications and Networking*, 2013.

[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.

Dans la même collection

Au sein de l'équipe SECRET :

- *Cryptographie fondée sur les codes : nouvelles approches pour constructions et preuves ; contribution en cryptanalyse.* Thomas Debris-Alazard, 2019
- *Topological Quantum Error-Correcting Codes beyond dimension 2.* Vivien Londe, 2019
- *Décodage des Codes Expanseurs Quantiques et Application au Calcul Quantique Tolérant aux Fautes.* Antoine Grospellier, 2019
- *Hidden Structures and Quantum Cryptanalysis.* Xavier Bonnetain, 2019
- *Mathématiques discrètes appliquées à la cryptographie symétrique.* Yann Rotella, 2018
- *Constructions pour la cryptographie à bas coût.* Sébastien Duval, 2018