Décodage générique de codes linéaires

Rodolfo Canto Torres

Décodage générique de codes linéaires

université de BORDEAUX

Mémoire de fin d'étude

Master *Sciences et Technologies*,
Mention *Informatique*,
Parcours *Cryptologie et Sécurité Informatique*.

Auteur
Rodolfo Canto Torres <rodolfo.canto-torres@inria.fr>

Superviseur
Nicolas Sendrier <nicolas.sendrier@inria.fr>

Tuteur
Emmanuel Fleury <fleury@labri.fr>

4 septembre 2015

Déclaration de paternité du document

Je certifie sur l'honneur que ce document que je soumet pour évaluation afin d'obtenir le diplôme de Master en *Sciences et Technologies*, Mention *Mathématiques* ou *Informatique*, Parcours *Cryptologie et Sécurité Informatique*, est entièrement issu de mon propre travail, que j'ai porté une attention raisonnable afin de m'assurer que son contenu est original, et qu'il n'enfreint pas, à ma connaissance, les lois relatives à la propriété intellectuelle, ni ne contient de matériel emprunté à d'autres, du moins pas sans qu'il ne soit clairement identifié et cité au sein de mon document.

Date et Signature

Remerciements

Je voudrais remercier énormément à tous les personnes qui m'ont aident de façon directe ou indirecte dans la réalisation de mon stage.

Je commence pour remercier à mon maître de stage Nicolas Sendrier pour ses conseils, opinions et critiques dans cet étude ; aussi pour sa patience, collaboration et bonne humeur offertes pendant la durée de ce stage. De la même façon, je remercie Anne Canteaut à la responsable de l'équipe-projet SECRET pour m'avoir orienté sur l'équipe-projet et avoir facilité le contact initial avec mon maître de stage. Et, finalement, à tous les membres de cet équipe pour tout la convivialité et bonne ambiance données.

Je continue pour remercier les responsables du Master Cryptologie et Sécurité informatique de Bordeaux pour leur travail, effort et temps. Je commence par remercier Gilles Zémor pour ses excellents cours, les discutions académiques avec lui et sa grande disposition aux documents administratifs. Un grande merci aussi à Emmanuel Fleury pour sa disponibilité et son aide irréfutable à tous mes doutes et questions pendant tout le master.

Résumé

Nous étudions ici le comportement asymptotique des meilleurs algorithmes de décodage générique des codes linéaires binaires lorsque le poids de l'erreur w est faible, c'est-à-dire $w = \mathbf{o}(n)$ où n est la longueur du code.

Cette étude sert à comprendre la sécurité du système cryptographique de type McEliece [4] basé sur les codes correcteurs d'erreurs. Les principales variantes utilisent soit des codes Goppa (ou plus généralement des codes alternants) auquel cas $w = \mathbf{O}\left(\frac{n}{\log_2(n)}\right)$, soit des codes MDPC [5] auquel cas $w = \mathbf{O}(\sqrt{n})$.

Les algorithmes étudiés ici sont l'algorithme de Prange [6], de Stern [7], de Dumer [2], de May, Meurer et Thomae [3], et de Becker, Joux, May et Meurer [1]. Nous montrons que pour un taux de transmission fixé $R = k/n$ (k la dimension du code), si le poids de l'erreur w est une fonction de n telle que $w = \mathbf{o}(n)$, alors nous avons

$$\lim_{n\to\infty} \frac{1}{w} \log_2 \mathrm{WF} = c$$

où c est constante identique pour tous les algorithmes mentionnés, égale à $-\log_2(1-R)$.

Le résultat est déjà significatif pour des valeurs « non asymptotiques » des paramètres utilisées dans certains systèmes. Par exemple, pour un code MDPC de longueur $n = 10000$, de dimension $k = 5000$ et corrigeant $w = 84$ erreurs, le nombre d'opérations élémentaires pour l'algorithme de Dumer est $2^{88.8}$ contre $2^{85.9}$ pour celui de Becker, Joux, May et Meurer. Cela correspond à un gain de seulement 3% dans l'exposant. Pour un même taux de transmission $k/n = 0.5$ et un nombre d'erreur voisin de la borne de Gilbert-Varshamov $w \approx 0.11n$, les complexités asymptotiques respectives des deux algorithmes mentionnés ci-dessus sont respectivement de $2^{0.115n}$ et $2^{0.099n}$ soit un gain de l'ordre de 15%.

Ce résultat signifie que si le nombre d'erreurs à corriger w est petit par rapport à la longueur n du code, ce qui est le cas pour tout les systèmes de chiffrement de type McEliece, alors les nombreux efforts et progrès algorithmiques des ces 50 dernières années sur le décodage générique n'ont qu'un impact limité sur la complexité algorithmique.

Dans le cas où le nombre des erreurs soit proportionnel à la longueur du mot, on aura que la complexité algorithmique est toujours exprime comme $\tilde{O}(2^{cn})$ où la constante c dépende de l'algorithme et ses paramètres. Pour étudier cette constante, on a développé un logiciel écrit en C qui obtient les paramètres optimaux des algorithmes mentionnés pour un taux de transmission

choisi pour lesquels la constante c est minimale. De cette forme, on a appris combien rapide puisse être un algorithme de décodage générique et combien sûr est un cryptosystème basé sur les codes.

Table des matières

Introduction

La cryptographie a été toujours une outil de sécurité pour la privacité et confidentiatialité d'information et ses cryptosystèmes sont développés à travers du temps. En 1965, Whitfield Diffie et Martin Hellman ont publié l'article "New directions in Cryptography" et établié un nouvelle generation de cryptosystèmes : le cryptosystème à clé publique. Ce cryptosystème permet à l'utilisateur de fournir une clé, dénominée *clé publique*, à un utilisateur quelconque pour que ce dernier puisse chiffrer les messages qu'il veut envoyer au prémier, de façon telle que le prémier utilisateur soit l'unique personne capable à déchiffrer le message à l'aide d'une autre clé, dénominé clé privée. Depuis ce date, plusiers cryptosystèmes à clé publique ont été proposé. En 1978, Robert McEliece a proposé un cryptosystème basée sur les codes correcteurs des erreurs qui suit une idée simple :

1. On obtaint un code correcteur linéaire C avec certains propiètes (de décodage) et on garde comme clé privée.
2. On construit un deuxième code correcteur C' à partir de C et on la donne comme clé publique.
3. Donc, pour chiffrer un message en clair x on l'ajoute un erreur aléatoire e et on codifie $x + e$ par $C\prime$, c'est-à-dire, on le multiplie par une matrice generatrice G' de C'.
4. Finallement, on peut déchiffrer le message chiffré, on utilise une méthode de correction des erreurs liée seulement à C qui permet obtenir x à partir de $G'(x + e)$ (quand e n'est pas trop "grand").

Comme tous les cryptosystèmes à clé publique, le cryptosystème de McEliece doit résoudre deux problèmes en sa implementation : l'obtention du clé privée à partir du clé publique et le déchiffrement directe du message chiffré. Cela se traduit dans notre cas en

1. On pourrait rétrouver le code correcteur C à partir de C'.
2. On pourrait corriger le erreur e dans $G'(x + e)$ sans connaître.

Le prémier problème est dénominé *indistinguabilité d'un code correcteur* et il a été étudié pendant beaucoup de temps, et malgré son importance on ne l'étudiera pas dans ce travail. Le deuxième problème est dénominé *problème de décodage* et cela sera le sujet de stage. Donc, la motivation de ce stage sera évaluer combien rapide peuvent être les méthodes de décodage quand on connait rien sur le code, c'est-à-dire, on considère le code comme un code aléatoire, et ce type de décodage est dénominé *décodage génerique*.

Pendant l'élaboration du stage, le point de vue a été toujours le point de vue du concepteur du crytposystème ; donc on supposera toujours que les algorithmes de décodage génerique doivent être manipules dans les conditions optimales, c'est-à-dire avec des bons paramètres (ou sinon qu'il est possible de trouver ces paramètres), et borner inférieurement la complexité algorithmique de ces méthodes.

On commence ce travail, avec une révision aux notions fondamentales réquises pour comprendre ce travail : la notion du code correcteur et la fonction d'entropie binaire. On traduira le problème de décodage sera traduit à un problème de décodage par syndrome. On obtiendra des formules asymptotiques pour le nombre combinatoire graçe à fonction entropie binaire. Dans le deuxième chapitre, on exposera algorithmes de décodage génerique plus remarquables de façon chronologique et on étudiera la complexité algorithmique de ces méthodes. Après, dans le troisième chapitre, on propose une borne asymptotique théorique pour la complexité de ces algorithmes, pour cela on construira une famille des fonctions bornes. Ces fonctions borne auront une propiète sur le valeur minimal très important qui permettra un étude assymptotique de ces fonctions. Cet étude assymptotique impliquera, sous certaines conditions, que la complexité algorithme assymptotique des algorithmes étudiés sont tous le même. Dans le dernier chapitre, on décrira un logiciel (développé dans ce stage) qui permet obtenir effectivement la complexité algorithmique optimal de ces algorithmes pour un taux de transmission et nombre des erreurs qui atteint la borne de Gilbert Varshamov.

CHAPITRE 1

Notions fondamentales

1.1 Codes correcteurs

On appellera *mot* a tout élément de $\mathbb{F}_2^n$. La définition suivante décrit la notion de distance sur les mots avec laquelle on aura une idée de proximité.

Définition 1.1.1 *Soit x, on define* le support $\mathrm{supp}(x)$ *comme l'ensemble de coordonnées non nulles et la* le poid de Hamming $\mathrm{wt}(x)$ *de x comme le cardinal de* $\mathrm{supp}(x)$*. Et, en particulier, on define* la distance de Hamming d_H *entre x et y par* $d_H(x, y) = \mathrm{wt}(x - y)$*, c'est-à-dire, le nombre des coordonnées differentes entre x et y.*

La structure principal de ce travail sera la de sous-espace linéraire et sera accompagne avec la notion de poids.

Définition 1.1.2 *On appellera* code linéaire *à tout sous-espace linéaire de $\mathbb{F}_2$. On define* la distance minimale $d_{\min}$ *de C comme la plus petite distance entre deux mots differents de C, équivalement, comme le plus petit poid d'Hamming d'un mot de C. Si* $\dim C = k$ *et* $d_{\min} = d$*, on dira que C est un $[n, k, d]$-code linéaire.*

Exemple 1.1.3 *L'ensemble*

$$C = \{x \in \mathbb{F}_2^7;\ x = (a, b, a, a, b, b, a + b),\ a, \in \mathbb{F}_2\}$$

est un $[7, 2, 4]$-code linéaire.

Le dénomination de code correcteurs provient du fait que sous certains hypothèses, on peut retrouver un mot c du codde à partir un mot erroné $c+e \in \mathbb{F}_2$.

Définition 1.1.4 *[Problème de Décodage] Soient C un code linéraire en $\mathbb{F}_2^n$, $x \in \mathbb{F}_2^n$ et $w \in \mathbb{Z}$. Trouver $c \in C$ tel que $d_H(w, c) \leq w$.*

Une façon de transformer cette problème est étudier la matrice de parité du code et le syndrome du mot à corriger.

Définition 1.1.5 *Soit C un $[n,k,d]$-code linéaire, on appelle* matrice de parité *à l'unique matrice $H \in \mathbb{F}_2^{n-k\times n}$ telle que $Hx = 0$, pour tout $x \in C$. Étant donné un mot x, on définit le* syndrome *de x par $\mathcal{S}_H(x) = Hx$.*

Exemple 1.1.6 *La matrice de parité pour le code C est*

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

Maintenant, on peut traduire le problème de décodage à une version qui utilise la version en fonction de syndrome. En effet, soit $c \in C$ un solution au problème de décodage, on fait $e = x-c$ et $s = Hx$. On obtient $He = Hx-0 = s$ et $\mathrm{wt}(e) \leq n$; donc e résout le problème de décodage par syndrome.

Définition 1.1.7 *[Computational Syndrome Decoding]*
Étant donné $H \in \mathbb{F}_2^{r\times n}$, $s \in \mathbb{F}^r$ et $w \in \mathbb{Z}$ positif, trouver $e \in \mathbb{F}_2^r$ tel que $He = s$ et $\mathrm{wt}(e) = w$. On dénote ce problème par $\mathrm{CSD}(H, w, s)$.

En conséquence il suffit résoudre un le problème de décodage par syndrome pour résoudre le problème de décodage. Réciproquement, si on a un syndrome s, on peut trouver, en temps polynomial, un mot x tel que $Hx = s$. Donc, si on peut résoudre le problème de décodage, on peut résoudre le problème e décodage par syndrome.

Une manière simple de regarder l'unicité des solutions au problème CSD est de regarder la distance de Gilbert-Varshamov.

Définition 1.1.8 *[Distance de Gilbert-Varshamov] La borne de Gilbert-Varshamov $d_0 = d_0(n, r)$ est le plus grand entier d_0 tel que*

$$\sum_{i=0}^{d_0} \binom{n}{i} \leq 2^r$$

Proposition 1.1.9 *Si $w \leq d_0(n, r)$, pour chaque $s \in \mathbb{F}_2^r$ on aura que $\mathrm{CSD}(H, w, s)$ admets au plus une solution.*

*D*émonstration
Il suffit compter le nombre des mots de poids inférieur à w et le nombre de mots en $\mathbb{F}_2$, et ils sont, respectivement, $\sum_{i=0}^{d_0} \binom{n}{i}$ et 2^r. □

1.2 Fonction d'entropie binaire

Dans ce travail on utiilisera toujours le logarithme en base 2, donc on dénotera cette fonction par $\log$ à place de $\log_2$.

Définition 1.2.1 *On définie la fonction* entropie binaire $h : [0, 1] \to \mathbb{R}$ *par $h(x) = x\log(x) + (1 - x)\log(1 - x)$.*

Cette fonction entropie sera très utile pour ce travail pour sa relation avec le nombre combinatoire. Le prochain théorème, surtout son corollaire, nous donnera une manière facile et précise de manier des nombres combinatoires.

Théorème 1.2.2 (Formule de Stirling) *On a le limite*

$$\lim_{n\to+\infty} \frac{n!}{\sqrt{2\pi n}(n/e)^n.} = 1$$

Corollaire 1.2.3 *On a le limite*

$$\lim_{n\to+\infty} \frac{\binom{n}{k}}{2^{nh(k/n)}\sqrt{n/(2\pi k(n-k))}} = 1$$

Grâce à ce corrolaire, on peut définir la bourne de Gilbert-Varshamov.

Définition 1.2.4 *Soit $k, n \in \mathbb{Z}$ tels que $k < n$, on définit la* borne *de Gilbert-Varshamov $w_0 \in]\,0, , n\,[$ par l'équation $h(w_0/n) = 1 - k/n$.*

Un lemme très utile pour le dernier chapitre sera le suivant.

Lemme 1.2.5 *Soit $a \in]0, 1[$, on a, à un facteur polynomial près, $h(ax) \geq ah(x)$, pour tout $x \leq [0, 1]$.*

*D*émonstration
Pour $x = 0$, cette inégalité est vraie, donc il reste vérifier que $ah'(ax) \geq ah'(x)$ pour tout $x > 0$. Pour cela, il suffit observer que $\log(1/ax - 1) \geq \log(1/x - 1)$ pour tout $x > 0$. □

Corollaire 1.2.6 *Soit $m, n \in \mathbb{N}$ et $a \in]0, 1[$, alors, à un facteur polynomial près, $\binom{m}{an} \geq \binom{m}{n}^a$.*

CHAPITRE 2

Algorithmes de décodage générique

On commence par expliquer les premiers algorithmes pour résoudre le *CSD* qui seront reutilisés comme des étapes intermédiaires pour les algorithmes plus récentes. Pour cela, on adoptera la notation suivante :

- Soit $n \in \mathbb{N}$, on note $[n]$ l'ensemble $\{1, 2, \ldots, n\}$ et $[n, m] = \{n, n+1, \ldots, m\}$.
- Pour tout ensemble fini I, on écrit $|I|$ pour indiquer son cardinal.
- Soient $M \in \mathbb{F}_2^{m \times n}$ et $I \subset [m]$, $J \subset [n]$ non vides. On définit $M_J^I \in \mathbb{F}_2^{|I| \times |J|}$ comme la sous-matrice M déterminé pour les lignes indexées par I et colonnes indexées par J. Dans le cas où $I = [m]$, on simplifiera cette notation par M^I et de la même manière dans le cas $J = [n]$.
- Pour tout $v \in \mathbb{F}_2^m$ et $I \subset [m]$, on définit v_I comme la projection de v sur les coordonnées indexées par I.
- On utilisera la notation *soft de Landau* $\tilde{O}$. On dit que $f = \tilde{O}(g)$ s'il existe un polynôme $p(n)$ et un entier N tels que $|f(n)| \leq p(n)|g(n)|$, pour tout $n \geq N$.

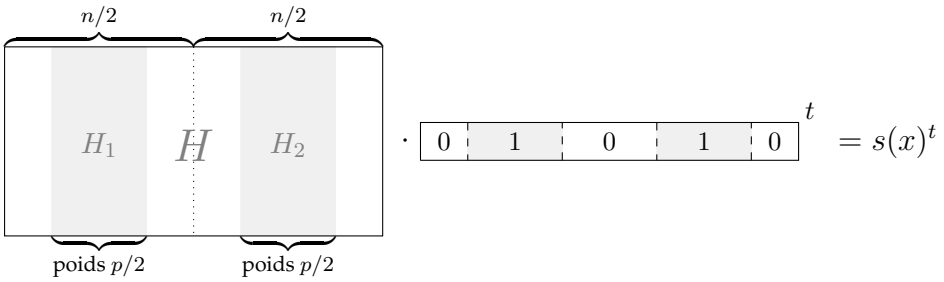
2.1 Décodage par collision

On peut regarder un $CSD(H, s, w)$ comme un problème de la somme de sous-ensemble : Il y existe un sous-ensemble de w colonnes de H dont la somme est égale à s. Donc, on peut utiliser le " paradoxe des anniversaires" pour faire mieux que la recherche exhaustive.

On divise $H = [H_1|H_2]$ avec $H_1, H_2 \in \mathbb{F}_2^{n-k \times \frac{n}{2}}$ et on construit deux listes

$$\mathcal{L}_1 = \{H_1 e_1 \ ; \ e_1 \in \mathbb{F}_2^{n/2} \ , \ \mathrm{wt}(e_1) = w/2\} \quad \text{et} \quad \mathcal{L}_2 = \{H_2 e_2 + s \ ; \ e_2 \in \mathbb{F}_2^{n/2} \ , \ \mathrm{wt}(e_2) = w/2\}.$$

Tout élément dans l'intersection de $\mathcal{L}_1$ et $\mathcal{L}_2$ sera une solution de $CDS(H, s, w)$.



Algorithme : Décodage par collision
Input : $H \in \mathbb{F}_2^{n-k \times n}$, $s \in \mathbb{F}_2^{n-k}$, $w \in \mathbb{Z}$ positif
Output : $\mathcal{L} \subset \{e \in \mathbb{F}_2^n \ ; \ He = s, \mathrm{wt}(e) = w\}$

Répéter
  Pour tout $e_1 \in \mathbb{F}_2^{n/2}$, $\mathrm{wt}(e_1) = w/2$ **faire**
    $x \leftarrow H_1 x_1$
    $T[x] \leftarrow T[x] \cup \{e_1\}$
  Pour tout $e_2 \in \mathbb{F}_2^{n/2}$, $\mathrm{wt}(e_2) = w/2$ **faire**
    $x \leftarrow H_2 x_2$
    Pour tout $e_1 \in T[x]$ **faire**
      $\mathcal{L} \leftarrow \mathcal{L} \cup \{(e_1, e_2)\}$
Retourner $\mathcal{L}$

On peut calculer la complexité de cet algorithme, de manière similaire aux prochains algorithmes présentés, de la façon suivante :

1. On calcule *le coût $\mathcal{K}$(possiblement) constant* de chaque itération.
2. On trouve *le nombre $\mathcal{N}$ des itérations*, lequel sera estimé comme l'inverse de *la probabilité de succès $\mathcal{P}$*.
3. Donc, on obtient *le facteur de travail* $\mathrm{WF} = \mathcal{N} \cdot \mathcal{K}$

Dans ce travail, on dénote la complexité algorithmique par WF ()*work factor*). Donc, on suit la procédure suivante :

1. On calcule le coût de cet algorithme :
   a) La production de la première liste indexée coûte $2\binom{n/2}{w/2}$.
   b) Pour chaque élément $e_2 \in \mathbb{F}_2^{n/2}$ de poids $w/2$ on fait une opération colonne plus une recherche de $\binom{n/2}{w/2}/2^{n-k}$ éléments en moyenne, cela nous donne $\binom{n/2}{w/2}(1 + \binom{n/2}{w/2}/2^{n-k})$.
   c) Donc, on a un coût
   $$3 * \binom{n/2}{w/2} + \binom{n/2}{w/2}^2 / 2^{n-k}.$$
2. Le succès sera quand les w erreurs seront divisés en $w/2$ erreurs dans les $n/2$ premières colonnes et les $n/2$ dernières colonnes de H, cela nous donne $\mathcal{P} = \binom{n/2}{w/2}^2 / \binom{n}{w}$.

3. Le cout de cet algorithme sera

$$\mathrm{WF}_{Collision} = 3 * \binom{n}{w} / \binom{n/2}{w/2} + \binom{n}{w} / 2^{n-k}.$$

Cela est coût brut de cet algorithme, mais si on obtenir une expression plus simple si on ne veut que les termes exponentiels. En effet, on réexamine la probabilité :

$$\mathcal{P} = \binom{n/2}{w/2}^2 / \binom{n}{w} \approx \left(\sqrt{\frac{n}{\pi w(n-w)}} 2^{\frac{n}{2} h(\frac{w}{n})}\right)^2 / \sqrt{\frac{n}{2\pi w(n-w)}} 2^{nh(\frac{w}{n})} = \sqrt{\frac{2n}{\pi w(n-w)}}.$$

Donc, on obtient $\mathcal{P} = \tilde{O}(1)$ et, en conséquence,

$$\tilde{O}(\mathrm{WF}_{Collision}) = \binom{n/2}{w/2} + \binom{n/2}{w/2} / 2^{n-k}.$$

Quand on réutilise cet algorithme comme une étape dans un algorithme plus complexe, on utilisera cet complexité.

2.2 Décodage par ensemble d'information (Prange)

Définition 2.2.1 *Soient C un $[n, k, d]$ code, $n \in \mathbb{N}$ et un $I \subset [n]$, on dira que I est un* ensemble d'information pour C *si pour tout $v \in \mathbb{F}_2^{|I|}$ existe un unique $c \in C$ tel que $c_I = v$.*

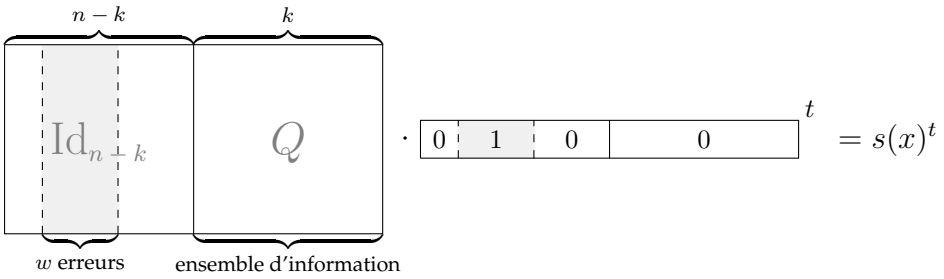
C'est claire qu'il y existe toujours au moins ensemble d'information I pour tout code choisi C et que $|I| = k$, mais la propieté qui motivera l'algorithme de cette sous-section est la proposition suivante.

Proposition 2.2.2 *Soit $v \in \mathbb{F}_2^n$, C un $[n, k, d]$-code. Alors,*

$$\{c \in C \ ; \ d(v, c) < d\} \subset \{c \in C \ ; \ c_I = v_I, I \text{ est un ensemble d'information pour } C\}.$$

*D*émonstration
Si $d(v, c) < d$ alors $I^* = \{i \in [n] \ ; c_i = v_i\}$ a cardinal au moins $n - (d - 1)$ donc $|C_{I^*}| = |C|$, en particulier $\dim C = \dim C_{I^*}$. On prends $I \subset I^*$ ensemble d'information pour C_{I^*} sur $(\mathbb{F}_2^n)_{I^*}$ et on montrera que I est un ensemble d'information pour C sur $\mathbb{F}_2^n$. En fait, pour tout $u \in \mathbb{F}_2^{|I|}$, il y existe un unique mot $c_{I^*} \in C_{I^*}$, et, en conséquence un unique $c \in C$, tel que $c_I = (c_{I^*})_I = u$. □

Cette propreté nous indique si on cherche un mot c du code C proche à un mot v, on peut le trouver tel que les coordonnées differentes soient dehors d'un ensemble d'information ; en particulier, l'erreur $e = v + c$ sera dans les $n - k$ coordonnées restantes. Une bonne façon d'utiliser cet avantage pour un $CDS(H, s, w)$ est de "placer" l'erreur dans la partie plus simple de H, la figure suivante donne une idée de notre objectif.



Pour accomplir ce but, on manipulera la matrice H par des opérations invertibles telles comme une permutation de colonnes et une élimination Gauss-Jordan dans ces files. Plus exactement, on prend aléatoirement une matrice de permutation $P_1 \in \mathbb{F}_2^{n\times n}$ et une matrice de permutation $P_2 \in \mathbb{F}_2^{n\times n}$ et $U \in \mathbb{F}^{n-k\times n-k}$ tel que

$$H' = U(HP) = [\mathrm{Id}_{n-k}|Q],$$

où $P = P_1P_2$ et Id_{n-k} est la matrice identité d'orden $n-k$. Donc, si e résout $CDS(H, s, w)$ alors $e' = P^{-1}e$ résout $CDS(H', Us, w)$. Si notre permutation P_1 a été favorable, on aura comme ensemble d'information les dernières k colonnes de H' et l'erreur e' dans les $n-k$ premières coordonnées, donc

$$e'_{[n-k]} = \mathrm{Id}_{n-k}e'_{[n-k]} + Q0 = H'e' = Us,$$

et, en conséquence, $e = P(Us|0)$. De cette façon, on obtient l'algorithmes dû à E. Prange **[?]**.

Algorithme : Décodage par ensemble d'information (Prange)
Input : $H \in \mathbb{F}_2^{n-k\times n}$, $s \in \mathbb{F}_2^{n-k}$, $w \in \mathbb{Z}$ positif
Output : $e \in \mathbb{F}_2^n$ tel que $He = s$ et $\mathrm{wt}(e) = w$
  Répéter
    Choisir aléatoirement une matrice $P_1 \in \mathbb{F}_2^{n\times n}$
    Calculer $P_2 \in \mathbb{F}_2^{n\times n}$ et $U \in \mathbb{F}_2^{n-k\times n-k}$ tel que $UHP_1P_2 = (\mathrm{Id}_{n-k}|Q)$
    si $\mathrm{wt}(Us) = w$ **alors**
      Retourner $P_1P_2(Us, 0)$

En conséquence, on aura :

1. On ignore le coût de la construction de P_1 (que peut être considéré comme une opérations file) et obtient $(n-k)^2(n-k-1)$ operations élémentaires pour obtenir la forme souhaitée.
2. Le succès sera quand les w erreurs seront dans les $n-k$ premières colonnes de H', cela nous donne $\mathcal{P} = \binom{n-k}{w}/\binom{n}{w}$.
3. Finalement, on obtient

$$\mathrm{WF}_{Prange} = (n-k)^2(n-k-1)\frac{\binom{n}{w}}{\binom{n-k}{w}}.$$

2.3 Algorithme de Lee et Brickell

On peut donner plus de liberté à la distribution des coordonnées non nuls de l'erreur dans l'algorithme de Prange en admettant que il y en a p dans l'ensemble d'information ; c'est-à-dire, $e' \in \mathbb{F}_2^n$ aura $\mathrm{wt}(e') = w$ et p coordonnées non nulles entre les k dernières et $w - p$ non nulles entre les $n - k$ premières. Pour cela, on essaie les $\binom{k}{p}$ erreurs partielles possibles $e^* \in \mathbb{F}_2^k$ de poids p et, similairement à Prange, on vérifie si $Us + Qe^*$ a poids $w - p$.

Algorithme : Décodage de Lee et Brickell
Input : $H \in \mathbb{F}_2^{n-k \times n}$, $s \in \mathbb{F}_2^{n-k}$, $w \in \mathbb{Z}$ positif
Output : $e \in \mathbb{F}_2^n$ tel que $He = s$ et $\mathrm{wt}(e) = w$
Répéter
Choisir aléatoirement une matrice $P_1 \in \mathbb{F}_2^{n \times n}$
Calculer $P_2 \in \mathbb{F}_2^{n \times n}$ et $U \in \mathbb{F}_2^{n-k \times n-k}$ tel que $UHP_1P_2 = (\mathrm{Id}_{n-k}|Q)$
Calculer $s' = US$
Faire la liste énumérée $\mathcal{L} = \{s' + Q'e^* \, ; e^* \in \mathbb{F}_2^{n-k}, \, \mathrm{wt}(e^*) = p\}$.
si $e^{**} \in \mathcal{L}$ avec $\mathrm{wt}(e^{**}) = w - p$
Retourner $P_1P_2(e^{**}, e^*)$

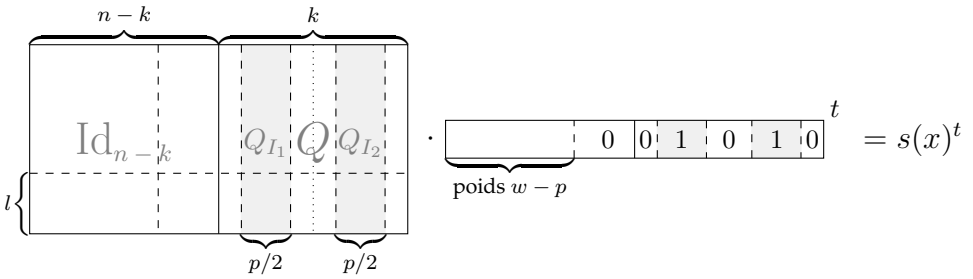
On calcule la complexité de cet algorithme :

1. Le coût est le même de celui de Prange plus l'énumération, ça nous donne $\mathcal{K} = (n-k)^2(n-k-1) + \binom{k}{p}$.
2. La probabilité de réussite sera améliorée à $\mathcal{P} = \dfrac{\binom{k}{p}\binom{n-k}{w-p}}{\binom{n}{w}}$.
3. On aura
$$\mathrm{WF}_{LB} = \frac{\binom{n}{w}}{\binom{n-k}{w-p}}\Big(\frac{(n-k)^2(n-k-1)}{\binom{k}{p}} + 1\Big);$$
si on prend en compte que $\binom{n-k}{w-p} < \binom{n-k}{w}$, on aura une borne inférieure pour cette complexité
$$\mathrm{WF}_{LB} \geq \frac{\binom{n}{w}}{\binom{n-k}{w-p}} \geq \frac{\mathrm{WF}_{Prange}}{(n-k)^2(n-k-1)} \qquad \text{alors} \qquad \mathrm{WF}_{LB} = \tilde{O}(\mathrm{WF}_{Prange})$$

Cette complexité motivera la borne proposée dans ce travail qui essaie de mesurer la complexité en fonction du nombre des erreurs.

2.4 Algorithme de Stern

On obtient $H' = [\mathrm{Id}_{n-k}|Q]$ et $s' = Us$ comme avant, mais on remplace la recherche exhaustive d'une possible solution dans somme de p colonnes de Q par la résolution (par collision) d'un sous-problème établi à partir de les l dernières lignes de Q : $CDS(Q^J, s'_I, p)$ où $I = [n-k+1, n]$, $J = [n-k-l+1, n-k]$ et l, p seront paramètres de l'algorithme à choisir.



Pour chaque solution $e^* \in \mathbb{F}_2^k$ de ce sous-problème, on aura $Qe^* = s'$ dans les l dernières coordonnées. Si le mot $(Qe^* + s')$ a poids $w-p$ (où uniquement les $n-k-l$ premières coordonnées seront les non nulles), alors $[Qe^* + s', e^*]$ sera une solution au problème $CSD(H', s', w)$.

Algorithme : Décodage de Stern
Input : $H \in \mathbb{F}_2^{n-k \times n}$, $s \in \mathbb{F}_2^{n-k}$, $s \in \mathbb{F}_2^{n-k}$, $w, l, p \in \mathbb{Z}$ positifs
Output : $e \in \mathbb{F}_2^n$ tel que $He = s$ et $\mathrm{wt}(e) = w$
  Répéter
    Choisir aléatoirement une matrice $P_1 \in \mathbb{F}_2^{n \times n}$
    Calculer $P_2 \in \mathbb{F}_2^{n \times n}$ et $U \in \mathbb{F}_2^{n-k \times n-k}$ tel que $H' = UHP_1P_2 = (\mathrm{Id}_{n-k}|Q)$
    Calculer $s' = Us$
    On fait $I = [n-k+1, n]$ et $J = [n-k-l+1, n-k]$
    Résoudre $CDS(Q^J, s'_I, p)$ par collision.
    Pour tout $e^* \in CDS(Q^J, s'_I, p)$ **faire**
      Si $\mathrm{wt}(Qe^* + s') = w - p$
        Retourner $P_1P_2(Qe^* + s', e^*)$

Maintenant, on analyse le facteur de travail de l'algorithme (à un facteur constant près) :

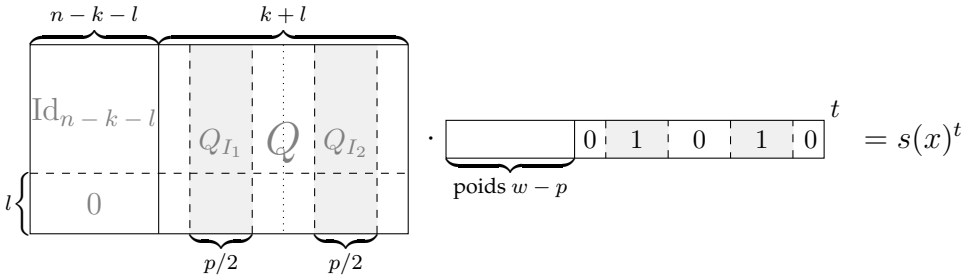
1. On aura un coût plus petit, il est l'élimination de Gauss et la résolution par collision du sous-problème

$$\mathcal{K} = (n-k)^2(n-k-1) + \binom{k/2}{p/2} + \binom{k/2}{p/2}^2 / 2^l.$$

2. La probabilité de réussite sera améliorée à $\mathcal{P} = \dfrac{\binom{k/2}{p/2}^2 \binom{n-k-l}{w-p}}{\binom{n}{w}}$.

3. On aura

$$\mathrm{WF}_{Stern} = \frac{\binom{n}{w}}{\binom{n-k-l}{w-p}} \Big(\frac{(n-k)^2(n-k-1)}{\binom{k/2}{p/2}^2} + \frac{1}{\binom{k/2}{p/2}} + \frac{1}{2^l} \Big).$$



2.5 Algorithme de Dumer

Cet algorithme est très similaire au précédent, mais il réduit le nombre de colonnes de l'élimination de Gauss dans la même quantité l de lignes du sous-problème, cela nous permet avoir plus des colonnes à tester que l'algorithme précédent.

Algorithme : Décodage de Dumer
Input : $H \in \mathbb{F}_2^{n-k \times n}$, $s \in \mathbb{F}_2^{n-k}$, $s \in \mathbb{F}_2^{n-k}$, $w, l, p \in \mathbb{Z}$ positifs
Output : $e \in \mathbb{F}_2^n$ tel que $He = s$ et $\mathrm{wt}(e) = w$
  Répéter
    Choisir aléatoirement une matrice $P_1 \in \mathbb{F}_2^{n \times n}$
    Calculer $P_2 \in \mathbb{F}_2^{n \times n}$ et $U \in \mathbb{F}_2^{n-k-l \times n-k-l}$ tel que
$H' = UHP_1P_2 = \binom{\mathrm{Id}_{n-k-l}}{0} | Q)$
    Calculer $s' = Us$,
    On fait $I = [n-k-l+1, n]$ et $J = [n-k-l+1, n-k]$
    Résoudre $CDS(Q^J, s'_J, p)$ par collision.
    Pour tout $e^* \in CDS(Q^J, s'_I, p)$ **faire**
      Si $\mathrm{wt}(Qe^* + s') = w - p$
        Retourner $P_1P_2(Qe^* + s', e^*)$

Maintenant, on analyse le facteur de travail de l'algorithme (à un facteur constant près) :

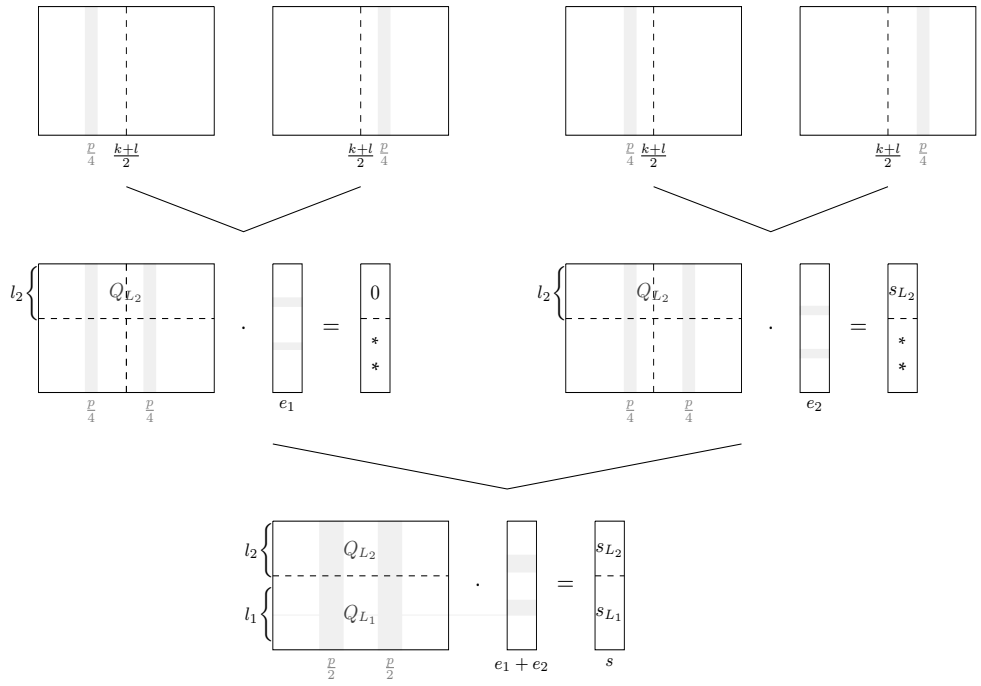
1. On aura un coût plus petit, il est l'élimination de Gauss et la résolution par collision du sous-problème

$$\mathcal{K} = (n-k-l)^2(n-k-l-1) + \binom{(k+l)/2}{p/2} + \binom{(k+l)/2}{p/2}^2 / 2^l.$$

2. La probabilité de réussite sera améliorée à $\mathcal{P} = \dfrac{\binom{(k+l)/2}{p/2}^2 \binom{n-k-l}{w-p}}{\binom{n}{w}}$.

3. On aura

$$\mathrm{WF}_{\mathrm{Dumer}} = \frac{\binom{n}{w}}{\binom{n-k-l}{w-p}} \Big(\frac{(n-k-l)^2(n-k-l-1)}{\binom{(k+l)/2}{p/2}^2} + \frac{1}{\binom{(k+l)/2}{p/2}} + \frac{1}{2^l} \Big).$$



2.6 Algorithme de May, Meurer et Thomae

On garde le même schéma que Dumer mais on améliore la résolution du problème au niveau du sous-problème établi : on améliore le méthode de collision en relaxant le choix des éléments des listes “à associer". Cette nouvelle méthode est nommée *AssocierColonnes* et on le décrit à continuation.

Soit $Q \in \mathbb{F}_2^{l \times k+l}$ et $s \in \mathbb{F}_2^l$, on cherche résoudre un problème $CDS(Q, s, p)$ en construisant des paires mots (avec support disjoints ou pas) $e_1, e_2 \in F_2^{k+l}$ telles que $\mathrm{wt}(e_1) = \mathrm{wt}(e_2) = \frac{p}{2}$ et

$$Qe_1 = Qe_2 + s;$$

le suivant diagramme décrit la construction de ces vecteurs e_1, e_2.

On obtiendra cet égalite par étapes, pour cela on divise les lignes de Q en deux parties disjoints L_1 et L_2, c'est-à-dire $L_1 \uplus L_2 = [l]$. On définie les listes

$$\begin{aligned}
\mathcal{L}_{1,1} &= \left\{(e_{11}, Q^{L_2} e_{11}) \,;\, \mathrm{supp}(e_{11}) \subset [1, \frac{k+l}{2}] \,,\, \mathrm{wt}(e_{11}) = \frac{p}{4}\right\} \\
\mathcal{L}_{1,2} &= \left\{(e_{12}, Q^{L_2} e_{12}) \,;\, \mathrm{supp}(e_{12}) \subset [\frac{k+l}{2} + 1, k+l] \,,\, \mathrm{wt}(e_{12}) = \frac{p}{4}\right\} \\
\mathcal{L}_{2,1} &= \left\{(e_{21}, Q^{L_2} e_{21}) \,;\, \mathrm{supp}(e_{21}) \subset [1, \frac{k+l}{2}] \,,\, \mathrm{wt}(e_{21}) = \frac{p}{4}\right\} \\
\mathcal{L}_{2,2} &= \left\{(e_{22}, Q^{L_2} e_{22} + s_{L_2}) \,;\, \mathrm{supp}(e_{22}) \subset [\frac{k+l}{2} + 1, k+l] \,,\, \mathrm{wt}(e_{22}) = \frac{p}{4}\right\}
\end{aligned}$$

Le prochain étape est associer les des éléments des $\mathcal{L}_{1,1}$ et $\mathcal{L}_{1,2}$ quand leurs deuxième composante est la même ; de cette forme on obtient une nouvelle liste

$$\mathcal{L}_1 = \{(e_1, Q_{L_1}e_1)\ ;\ e_1 = e_{11}+e_{12}\ ,\ (e_{1i}, Q_{L_2}e_{1i}) \in \mathcal{L}_{1,i}\ ,\ Q^{L_2}e_{11} = Q^{L_2}e_{12},\ i = 1, 2\}.$$

Donc, on obtient

$$\mathrm{wt}(e_1) = \frac{p}{2} \quad \text{et} \quad Q_{L_2}e_1 = 0,$$

pour tout $(e_1, Q^{L_1}e_1) \in \mathcal{L}_1$. De la même manière, on construit

$$\mathcal{L}_2 = \{(e_2, Q^{L_1}e_2+s_{L_1})\ ;\ e_2 = e_{21}+e_{22},\ (e_{2i}, Q^{L_2}e_{2i}) \in \mathcal{L}_{2,i},\ Q^{L_2}e_{21} = Q^{L_2}e_{22},\ i = 1, 2\}.$$

En conséquence,

$$\mathrm{wt}(e_2) = \frac{p}{2} \quad \text{et} \quad Q^{L_2}e_2 = s_{L_2},$$

pour tout $(e_2, Q^{L_1}e_2 + s_{L_1})$. Finalement, on obtient la liste des "possibles" solutions

$$\mathcal{L} = \{e_1+e_2\ ;\ (e_1, Q^{L_1}e_1) \in \mathcal{L}_1,\ (e_2, Q^{L_1}e_2+s_{L_1}) \in \mathcal{L}_2,\ Q^{L_1}e_1 = Q^{L_1}e_2+s_{L_1}\}.$$

Algorithme : Associer Colonnes
Input : $Q \in \mathbb{F}_2^{l\times k+l}$, $s \in \mathbb{F}_2^l$, $s \in \mathbb{F}_2^{n-k}$, $p \leq k + l$ positifs
Output : $\mathcal{L} \subset \{e \in \mathbb{F}_2^l;\ \mathrm{wt}(e) \leq p,\ Qe = s\}$
Paramètres : $L_1, L_2 \subset [l]$, $L_1 \uplus L_2 = L$ et $|L_i| = |l_i|$, pour $i = 1, 2$.

Répéter
Cosntruire $\mathcal{L}_{1,1}$, $\mathcal{L}_{1,2}$, $\mathcal{L}_{2,1}$ et $\mathcal{L}_{2,2}$
Ranger les listes $\mathcal{L}_{1,1}$ et $\mathcal{L}_{2,2}$ en rapport à la deuxièeme composantes de leurs éléments.
Pour tout $(e_{1,1}, Q_{L_2}e_{1,1}) \in \mathcal{L}_{1,1}$, faire
Pour tout $(e_{1,2}, Q_{L_2}e_{1,2}) \in \mathcal{L}_{1,2}$, faire
Si $Q_{L_2}e_{1,1} = Q_{L_2}e_{1,2}$ **alors**
Faire $e_1 = e_{1,1} + e_{1,2}$
Ajouter $(e_1, Q_{L_1}e_1)$ à $\mathcal{L}_1$
Pour tout $(e_{2,1}, Q_{L_2}e_{2,1}) \in \mathcal{L}_{2,1}$, faire
Pour tout $(e_{2,2}, Q_{L_2}e_{2,2} + s_{L_2}) \in \mathcal{L}_{2,2}$, faire
Si $Q_{L_2}e_{2,1} = Q_{L_2}e_{2,2} + s_{L_2}$ **alors**
Faire $e_2 = e_{2,1} + e_{2,2}$
Ajouter $(e_2, Q_{L_1}e_1 + s_{L_1})$ à $\mathcal{L}_2$
Ranger les listes $\mathcal{L}_2$ et en rapport à la deuxièeme composante de ses éléments.
Pour tout $(e_1, Q_{L_1}e_1) \in \mathcal{L}_1$, faire
Pour tout $(e_2, Q_{L_1}e_2 + s_{L_1}) \in \mathcal{L}_2$, faire
Si $Q_{L_1}e_1 = Q_{L_2}e_2 + s_{L_1}$ **alors**
Ajouter $e_1 + e_2$ à $\mathcal{L}$
Retourner $\mathcal{L}$

Pourtant, on observe que $\mathrm{wt}(e_1 + e_2) \leq p$; mais cela peut être remédié dans l'algorithme général en acceptant plus des erreurs dehors de l'ensemble d'information

---

Algorithme : Décodage de May, Meurer et Thomae
Input : $H \in \mathbb{F}_2^{n-k \times n}$, $s \in \mathbb{F}_2^{n-k}$, $s \in \mathbb{F}_2^{n-k}$, $w \in \mathbb{Z}$ positif
Output : $e \in \mathbb{F}_2^n$ tel que $He = s$ et $\mathrm{wt}(e) = w$
Paramètres p, l et l_1, l_2 tels que $l = l_1 + l_2$
Répéter
Choisir aléatoirement une matrice $P_1 \in \mathbb{F}_2^{n \times n}$
Calculer $P_2 \in \mathbb{F}_2^{n \times n}$ et $U \in \mathbb{F}_2^{n-k-l \times n-k-l}$ tel que
$H' = UHP_1P_2 = \binom{\mathrm{Id}_{n-k-l}}{0} |Q)$
Calculer $s' = Us$,
On fait $I = [n-k-l+1, n]$ et $J = [n-k-l+1, n-k]$.
Calculer $\mathcal{L} = \mathbf{AssocierColonnes}(Q^J, s'_I, p)$.
Pour tout $e \in \mathcal{L}$ **faire**
Si $\mathrm{wt}(Qe + s') = w - \mathrm{wt}(e)$
Retourner $P_1P_2(Qe + s', e)$

Maintenant, on analyse le facteur de travail de l'algorithme. On donnera une estimation au cas le plus fréquent en pratique où on trouve des listes associées $\mathcal{L}_1$ et $\mathcal{L}_2$ avec des mots de supports disjoints.

1. On aura toujours le coût de l'élimination de Gauss qui est polynomial et on l'ignore pour simplifier l'expression et le calcul, donc, il reste analyser le coût de la méthode "Associercolonnes". On commence par faire 2 fois le décodage par collision avec les listes $\mathcal{L}_{1,1}$ $\mathcal{L}_{1,2}$, $\mathcal{L}_{2,1}$ et $\mathcal{L}_{2,2}$, cela donne

$$\binom{(k+l)/2}{p/4} + \binom{(k+l)/2}{p/4}^2 / 2^{l_2} \quad \text{itérations}$$

en moyenne et des nouvelles listes $\mathcal{L}_1$ et $\mathcal{L}_2$ de taille en moyenne $\binom{(k+l)/2}{p/4}^2 / 2^{l_2}$. Donc, la collision de ces deux nouvelles listes nous donne

$$\binom{(k+l)/2}{p/4}^2 / 2^{l_2} + \left(\binom{(k+l)/2}{p/4}^2 / 2^{l_2}\right)^2 / 2^{l_1} \quad \text{itérations}$$

seulement pour les listes associées de mots de supports disjoints. En conséquence, le nombre des itérations, en ignorant des termes polynomiaux, est

$$\mathcal{K} = \binom{(k+l)/2}{p/4} + \binom{(k+l)/2}{p/4}^2 / 2^{l_2} + \left(\binom{(k+l)/2}{p/4}^2 / 2^{l_2}\right)^2 / 2^{l_1} \quad \text{itérations.}$$

2. La probabilité de réussite de la méthode *AssocierColonnes* est au moins 0.5 comme il est décrit en [3], donc on peut ignorer cette probabilité dans

nos calculs. La probabilité de réussite sera améliorée à nouveau. Pour le cas où les listes $\mathcal{L}_1$ et $\mathcal{L}_2$ ont des mots du supports disjoints, on aura

$$\binom{k+l}{p/2}\binom{k+l-p/2}{p/2}$$

représentations possibles d'un mot de longueur $k+l$ et poids p. On note que chacun de ces mots sont représentes $\binom{p}{p/2}$ fois dans cette manière. Donc la probabilité pour ce cas sera

$$\mathcal{P} = \frac{\left(\binom{k+l}{p/2}\binom{k+l-p/2}{p/2}\binom{p}{p/2}^{-1}\right)\binom{n-k-l}{w-p}}{\binom{n}{w}} = \frac{\binom{k+l}{p}\binom{n-k-l}{w-p}}{\binom{n}{w}}.$$

3. On note que $\binom{(k+l)/2}{p/4}^2 = \tilde{O}\left(\binom{k+l}{p/2}\right)$, donc on aura

$$\mathrm{WF}_{MMT} = \frac{\binom{n}{w}}{\binom{n-k-l}{w-p}\binom{k+l}{p}}\left(\sqrt{\binom{k+l}{p/2}} + \frac{\binom{k+l}{p/2}}{2^{l_2}} + \frac{\binom{k+l}{p/2}^2}{2^{l+l_2}}\right).$$

2.7 Algorithme de Becker, Joux, May et Meurer

On commence de la même manière que l'algorithme précédent et on aura comme objectif résoudre le même sous-problème établi. On utilisera une stratégie similaire pour le résoudre mais avec La principale différence est prendre en compte le fait que $1+1=0$ dans $\mathbb{F}_2$ et, en conséquence, on peut accepter plus de représentations à associer entre les 4 étapes qui dispose le nouvelle méthode *AssocierColonnes*.

Soit $Q \in \mathbb{F}_2^{l\times k+l}$ et $s \in \mathbb{F}_2^l$, on cherche à nouveau obtenir une solution $e \in \mathbb{F}^{k+l}$ au problème $CDS(Q,s,p)$. On construira des paires des mots $e_1, e_2 \in F_2^{k+l}$ avec des supports non disjoints tels que $Q(e_1+e_2)=s$.

On commence par décider que type de mot seront e_1 et e_2 de quelle façon ils s'approcheront de e. On choisit les paramètres $\epsilon_1 > 0$, $r_2 \in [\,0, k+l\,]$, et on définie le poids $p_1 = \frac{p}{2} + \epsilon_1$ pour cet étape ; en plus on choisira aléatoirement la cible $t_1^{(1)} \in \mathbb{F}^{r_1}$ et on définie la deuxièeme cible $t_2^{(2)} = s_{[r_1]} + t_1^{(1)}$. Donc, on demandera que

$$\mathrm{wt}(e_i^{(1)}) \in \mathbb{F}_2^{k+l} \quad \text{et} \quad Q^{[r_1]}e_i^{(1)} = t_i^{(1)}, \qquad \text{pour } i = 1,2.$$

Parce que nous souhaitons que $\mathrm{wt}(e_1^{(1)} + e_2^{(1)}) = p$, le nombre de representations sera

$$R_1(p,l,\epsilon_1) = \binom{p}{p/2}\binom{k+l-p}{\epsilon_1}.$$

Donc, si on veut avoir de la chance d'atteindre les mots $t_1^{(1)}$ et $t_2^{(1)}$ par ces répresentations et ne pas avoir de trop de répétitions, on fait $2^{r_1} = R_1$ ou bien $r_1 = \log(R_1)$. On peut voir qu'un paire e_1 et e_2 satisfait $Q^{[r_1]}(e_1^{(1)} + e_2^{(1)}) = s_{[r_1]}$, donc on s'approche à la solution.

Maintenant, on analyse la suivante étape, comment créer les mots e_1 et e_2, on répétera la même procedure avec $e_1^{(2)}, e_2^{(2)}$ et $e_3^{(2)}, e_4^{(2)}$. On choisit ϵ_2 la taille

de l'intersection des supports et on fait $p_2 = \frac{p_1}{2} + \epsilon_2$ et $r_2 \in [\,0, k+l\,]$; en plus, on choisit aléatoirement les cibles $t_1^{(2i-1)}$ et on défine $t_{2i}^{(2i)} = t_{2i-1}^{(2)} + s_{[r_2]}$. Donc, on demandera à nouveau que

$$\mathrm{wt}(e_i^{(2)}) \in \mathbb{F}_2^{k+l} \quad \text{et} \quad Q^{[r_2]} e_i^{(2)} = t_i^{(2)}, \qquad \text{pour } i = 1, 2, 3, 4.$$

Parce que nous souhaitons que $\mathrm{wt}(e_1^{(2)} + e_2^{(2)}) = p_1$, le nombre de representations sera

$$R_2(p, l, \epsilon_1, \epsilon_2) = \binom{p_1}{p_1/2} \binom{k+l-p_1}{\epsilon_2}.$$

Donc, pour le même raisons que dans l'étape précédente, on fait à nouveau $r_2 = \log(R_2)$. On peut voir qu'un paire e_1^2 et e_2^2 satisfait $Q^{[r_2]}(e_1^{(2)} + e_2^{(2)}) = s_{[r_2]}$, donc on s'approche aux mots $e_1^{(1)}, e_2^{(1)}$.

Finalement, pour obtenir des listes des mots $e_i^{(2)}$, on utilise le décodage par collision. Maintenant, on recapituile les pas du méthode *AssocierColonnes* et la construction de ces listes et, pour faciliter la notation, on suppose que se listes sont ordenées par son produit respecte à Q ou $Q^{[r_i]}$, selon le cas, et que l'on fait colissioner comme dans l'algorithme MMT.

1. On choisit les paramètres $p_1 = p/2 + \epsilon_1$, $p_2 = p_1/2 + \epsilon_2$ et on défine $r_1 = \log R_1$ et $r_2 = \log R_2$ comme avant.
2. On choisit aléatoirement $t_1^{(1)} \in \mathbb{F}_2^{r_1}$, $t_1^{(2)}$, $t_3^{(2)} \in \mathbb{F}_2^{r_2}$ et on défine

$$t_2^{(1)} = s_{[r_1]} + t_1^{(1)}, \quad t_1^{(2)} = s_{[r_2]} + t_1^{(2)} \quad \text{et} \quad t_4^{(2)} = s_{[r_2]} + t_3^{(2)}.$$

3. Pour $i = 1, 2, 3, 4$, on divise aléatoirement $[k+l]$ en deux parties disjoint P_1, P_2 telles que $|P_{i,1}| = |P_{i,2}| = \frac{k+l}{2}$. On construit les listes initiales

$$\begin{aligned} \mathcal{B}_{i,1} &= \big\{ y_i \in \mathbb{F}_2^{k+l} \;;\; \mathrm{supp}(y_1) \subset P_{i,1} \;,\; \mathrm{wt}(y_1) = \frac{p_2}{2} \big\} \\ \mathcal{B}_{i,2} &= \big\{ z_1 \in \mathbb{F}_2^{k+l} \;;\; \mathrm{supp}(y_1) \subset P_{i,1} \;,\; \mathrm{wt}(z_1) = \frac{p_2}{2} \big\}. \end{aligned}$$

4. Pour $i = 1, 2, 3, 4$, on construit les listes de deuxième niveau

$$\mathcal{L}_i^{(2)} = \big\{ e_i^{(2)} = y_i + z_i \;;\; y_i \in \mathcal{B}_{i,1},\, z_i \in \mathcal{B}_{i,2},\, Q^{[r_2]} e_i^{(2)} = t_i^{(2)} \big\}.$$

5. Pour $i = 1, 2$, on construit les listes de premier niveau

$$\mathcal{L}_i^{(1)} = \big\{ e_i^{(1)} = e_{2i-1}^{(2)} + e_{2i}^{(2)} \;;\; e_{2i-1}^{(2)} \in \mathcal{L}_{2i-1}^{(2)},\, e_{2i}^{(2)} \in \mathcal{L}_{2i}^{(2)},\, \mathrm{wt}(e_i^{(1)}) = p_1,\, Q^{[r_1]} e_i^{(1)} = t_i^{(1)} \big\}.$$

6. On obtient la liste des possibles solutions

$$\mathcal{L} = \big\{ e = e_1^{(1)} + e_2^{(1)} \;;\; e_1^{(1)} \in \mathcal{L}_1^{(1)},\, e_2 \in \mathcal{L}_2^{(1)}, \mathrm{wt}(e) = p,\, Qe = s \big\}.$$

Algorithme : Décodage de Becker, Joux, May et Meurer
Input : $H \in \mathbb{F}_2^{n-k \times n}$, $s \in \mathbb{F}_2^{n-k}$, $s \in \mathbb{F}_2^{n-k}$, $w \in \mathbb{Z}$ positif
Output : $e \in \mathbb{F}_2^n$ tel que $He = s$ et $\mathrm{wt}(e) = w$
Paramètres p, l et $p_1 \in]0, p/2[$, $p_2 \in]0, p_1/2[$.

Répéter
Choisir aléatoirement une matrice $P_1 \in \mathbb{F}_2^{n\times n}$
Calculer $P_2 \in \mathbb{F}_2^{n\times n}$ et $U \in \mathbb{F}_2^{n-k-l\times n-k-l}$ tel que
$H' = UHP_1P_2 = \left(\begin{smallmatrix}\mathrm{Id}_{n-k-l}\\ 0\end{smallmatrix}|Q\right)$
Calculer $s' = Us$,
On fait $I = [n-k-l+1, n]$ et $J = [n-k-l+1, n-k]$.
Calculer $\mathcal{L} = \mathbf{AssocierColonnes}(Q^J, s'_I, p, p_1, p_2)$.
Pour tout $e \in \mathcal{L}$ **faire**
Si $\mathrm{wt}(Qe + s') = w - p$
Retourner $P_1P_2(Qe + s', e)$

Finalement, on fait l'analyse de complexité de cet algorithme

1. On ignore le coût de l'élimination de Gauss, donc, il reste analyser le coût de la nouvelle méthode "Associercolonnes". On commence par faire 4 fois le décodage par collision avec les listes $\mathcal{B}_{1,1}, \mathcal{B}_{1,2}, \ldots, \mathcal{B}_{4,1}, \mathcal{B}_{4,2}$ de taille
$$S_3 = \binom{(k+l)/2}{p_2/2},$$
cela donne des nouvelles listes $\mathcal{L}_1^{(2)}, \ldots, \mathcal{L}_4^{(2)}$ de taille en moyenne
$$C_3 = \binom{(k+l)/2}{p_2/2}^2 /2^{r_2} = \binom{k+l}{p_2}/2^{r_2}.$$
Dans la étape de niveau 2, on attends que les listes que nous avons crée $\mathcal{L}_i^{(2)}$ soient, en moyenne, toutes les solutions à cet étape avec la taille de
$$S_2 = \binom{k+l}{p_2}/2^{r_2};$$
donc après son collision on aura des listes $\mathcal{L}_i^{(1)}$, avec des possibles répétitions, de taille
$$C_2 = S_2^2/2^{r_1-r_2} = \binom{k+l}{p_2}/2^{r_2+r_1}$$
en moyenne. Dans l'étape de niveau 1, on attends que les listes soient, en moyenne, toutes les solutions à cet étape et que leurs taille soient
$$S_1 = \binom{k+l}{p_1}/2^{r_1};$$
donc après son collision on aura des listes $\mathcal{L}$, avec des possibles répétitions, de taille
$$C_1 = S_1^2/2^{l-r_1} = \binom{k+l}{p_1}^2 /2^{l+r_1}.$$
Finalement, le nombre moyenne des solutions existants sera
$$S_0 = \binom{k+l}{p}/2^l.$$

Dans chaque étape, on a $S_i + C_i$ itérations, donc le nombre des itérations à faire est sa somme

$$\mathcal{K} = S_1 + C_1 + S_2 + C_2 + S_3 + C_3;$$

cependant on peut simplifier cette expression en montrant une relation entre les tailles C_i et S_{i-1} pour $i = 1, 2, 3$. En effet,

$$\begin{aligned}
\binom{k+l}{p_1} 2^{r_2} &= \binom{p_1}{p_1/2}\binom{k+l-p_1}{\epsilon_2}\binom{k+l}{p_1} \\
&= \frac{1}{\frac{p_1}{2}!\frac{p_1}{2}!}\frac{1}{\epsilon_2!(k+l-p_1-\epsilon_2)!}(k+l)! \\
&= \frac{(k+l-p_2)!}{\frac{p_1}{2}!(k+l-p_2-\frac{p_1}{2})!}\frac{p_2!}{\frac{p_1}{2}!\epsilon_2!}\frac{(k+l)!}{p_2!(k+l-p_2)!} \\
&= \binom{p_2}{\epsilon_2}\binom{k+l-p_2}{p_2-\epsilon_2}\binom{k+l}{p_2} \\
&= \mu_2\binom{k+l}{p_2}^2
\end{aligned}$$

où μ_2 est la probabilité que deux mot en $\mathbb{F}_2^{k+l}$ de poids p_2 coïncident exactement en ϵ_2 coordonnées non nulles. Donc, on obtient

$$S_1 = \mu_2 C_2;$$

de la même manière, on obtient que $S_0 = \mu_1 C_1$, où μ_1 est la probabilité que deux mot en $\mathbb{F}_2^{k+l}$ de poids p_1 coïncident exactement en ϵ_1 coordonnées non nulles. Finalement, on a, sauf termes polynomiaux et non dominants,

$$\begin{aligned}
\mathcal{K} &= S_3 + C_3 + C_2 + C_1 \\
&= \sqrt{\tbinom{k+l}{p_2}} + \frac{\binom{k+l}{p_2}^2}{2^{r_2}} + \frac{\binom{k+l}{p_1}^2}{2^{r_2+r_1}} + \frac{\binom{k+l}{p_2}^2}{2^{l+r_1}} \\
&= \sqrt{\tbinom{k+l}{p_2}} + \frac{\binom{k+l}{p_1}}{\mu_2\binom{k+l}{p_2}} + \frac{\binom{k+l}{p}}{\mu_2\mu_1\binom{k+l}{p_1}} + \frac{\binom{k+l}{p}}{\mu_1 2^l}
\end{aligned}$$

En plus, on minimisera le nombre de paramètres dans l'expression de μ_1 et μ_2, en faisant

$$\mu_1 = \frac{\binom{p_1}{p/2}\binom{k+l-p_1}{p/2}}{\binom{k+l}{p_1}} \quad \text{et} \quad \mu_2 = \frac{\binom{p_2}{p_1/2}\binom{k+l-p_2}{p_1/2}}{\binom{k+l}{p_2}}$$

2. Grâce au choix des paramètres r_1 et r_2, on attend que la probalité de réussite de la méthode *AssocierColonnes* est aussi bonne que sa version précedente, donc on peut ignorer cette probabilité dans nos calculs. La probabilité de réusite sera la même. Pour les listes $\mathcal{L}_1^{(2)}$ et $\mathcal{L}_2^2$ ont des mots de poids p_2 qui coïncident dans ϵ_2 éléments de leurs supports, donc on aura

$$\binom{k+l}{p_1/2}\binom{k+l-p_1/2}{p_1/2}\binom{k+l-p_1}{\epsilon_2},$$

représentations possibles d'un mot de longer $k+l$ et poids p. On note que chacun de ces mots sont représentes R_2 fois dans cette manière. Donc il y aura $\binom{k+l}{p_1}$ mots en moyenne pour la liste $\mathcal{L}_1^{(1)}$, de la même manière on obtiendra le même pour $\mathcal{L}_2^{(1)}$, donc on peut compter avec tous les mots possibles. Pour les listes $\mathcal{L}_1^{(1)}$ et $\mathcal{L}_2^{(1)}$, on aura aussi

$$\binom{k+l}{p/2}\binom{k+l-p/2}{p/2}\binom{k+l-p}{\epsilon_1},$$

représentations possibles d'un mot de longueur $k+l$ et poids p, et R_1 représentations pour chaque de ces mots. Alors, on aura, en moyenne, $\binom{k+l}{p}$ possibles mots et la probabilité sera

$$\mathcal{P} = \frac{\binom{k+l}{p}\binom{n-k-l}{w-p}}{\binom{n}{w}}.$$

3. On obtient

$$\mathrm{WF}_{BJMM} = \frac{\binom{n}{w}}{\binom{n-k-l}{w-p}}\Big(\frac{\sqrt{\binom{k+l}{p_2}}}{\binom{k+l}{p}} + \frac{\binom{k+l}{p_1}}{\mu_2\binom{k+l}{p_2}\binom{k+l}{p}} + \frac{1}{\mu_2\mu_1\binom{k+l}{p_1}} + \frac{1}{\mu_1 2^l}\Big).$$

CHAPITRE 3

Borne inférieure pour la complexité

On commence par calculer la complexité de l'algorithme de Prange, c'est qui nous donnera le valeur asymptotique à comparer avec les autres algorithmes. À un facteur polynomial près, on a

$$\mathrm{WF}_{\mathrm{Prange}} = \frac{\binom{n}{w}}{\binom{n-k}{w}}.$$

Donc, si on assume que $w = \mathbf{o}(n)$, on obtiendra

$$\begin{aligned}
\log \mathrm{WF}_{\mathrm{Prange}} &\approx nh(w/n) - (n-k)h(w/(n-k)) \\
&= -n\Big(\frac{w}{n}\log\big(\frac{w}{n}\big) - \big(1-\frac{w}{n}\big)\log\big(1-\frac{w}{n}\big)\Big) \\
&\quad +(n-k)\Big(\frac{w}{n-k}\log\big(\frac{w}{n-k}\big) + \big(1-\frac{w}{n-k}\big)\log\big(1-\frac{w}{n-k}\big)\Big) \\
&= -w\log\big(\frac{w}{n}\big) + w\log\big(\frac{w}{n-k}\big) \\
&\quad -n\big(1-\frac{w}{n}\big)\Big(\frac{w}{n} + \mathcal{O}\big(\frac{w}{n}^2\big)\Big) + (n-k)(1-\frac{w}{n-k})(\frac{w}{n-k} + \mathcal{O}((\frac{w}{n-k})^2)) \\
&= w\log\Big(\frac{1}{1-\frac{k}{n}}\Big) - w(1-\tau)\Big(1+\mathcal{O}(\frac{w}{n})\Big) + w\Big(1-\frac{w}{n-k}\Big)\Big(1+\mathcal{O}(\frac{w}{n-k})\Big) \\
&= w\log\Big(\frac{1}{1-\frac{k}{n}}\Big) + w\Big(-1+\mathcal{O}\Big(\frac{w}{n}\Big) + 1 + \mathcal{O}(\frac{w}{n-k})\Big) \\
&= w\left(\log\Big(\frac{1}{1-\frac{k}{n}}\Big) + \mathbf{o}(1)\right)
\end{aligned}$$

On déduit que pour des grands valeurs de n, on a

$$\mathrm{WF}_{\mathrm{Prange}} = 2^{w(-\log(1-R)+\mathbf{o}(1))},$$

où $R = \lim_{n\to\infty} \frac{k}{n}$. Dans les sections suivantes, on montrera que cette complexité, respecte a grands valeurs de w, est la meilleur possible pour certains cas.

3.1 La famille des fonctions borne

On définie une famille de fonctions qui nous aidera énormément dans notre but de borner inférieurement certaines complexités algorithmes. On fait

$$\mathfrak{B}_a(l,p) = \frac{\binom{n}{w}}{\binom{n-k-l}{w-p}}\Big(\frac{1}{2^l} + \frac{1}{\binom{k+l}{ap}}\Big),$$

où a sera une constante qui dépende que de l'algorithme et l, p seront variables qui émulent les paramètres des algorithmes.

Lemme 3.1.1 *Soient* $\mathcal{D} = \{(l,p) \in [\,0, n-k] \times [\,0, w\,], w - p \leq n - k - l\,,\ ap \leq l + k\}$ *et* $a \in]0,1[$. *Si* $w < \frac{n-k}{2}$ *donc*

$$\min_{(l,p)\in\mathcal{D}} \mathfrak{B}_a(l,p) = \min_{(l,p)\in\mathcal{V}} \mathfrak{B}_a(l,p) \qquad \textit{avec} \qquad \mathcal{V} = \Big\{(l,p) \in \mathcal{D}\,,\ 2^l = \binom{k+l}{ap}\Big\}$$

On place la démonstration de ce lemme dans l'annexe pour être très grande et elle n'apporte pas de la compréhension au contenu. Grâce à ce lemme, on peut simplifier la borne inférieure $\mathfrak{B}_a$ quand on cherche le minimum ; maintenant, on ajoutera des hypothèses pour simplifier encore la borne inférieure

Lemme 3.1.2 *Si* $\lim_{n\to\infty} \frac{w}{n} = 0$, $\lim_{n\to\infty} \frac{k}{n} = R$ *et* $2^l = \binom{k+l}{ap}$, *alors* $\lim_{n\to\infty} \frac{l}{n} = 0$.

*D*émonstration
On prend le logarithme en base 2 sur la troisième hypothèse et obtient

$$\begin{aligned} l &= (k+l)\frac{ap}{k+l}\log\Big(\frac{ap}{k+l}\Big) + (k+l)\Big(1-\frac{ap}{k+l}\Big)\log\Big(1-\frac{ap}{k+l}\Big) \\ &= ap\log\Big(\frac{ap}{k+l}\Big) + ap\Big(1-\frac{ap}{k+l}\Big)\Big(1+\mathbf{O}(\frac{ap}{k+l})\Big). \end{aligned}$$

Donc, si on prend en compte les autres restrictions, on obtient

$$\begin{aligned} l/n &\leq a\frac{w}{n}\log\Big(\frac{w}{k}\Big) + a\frac{w}{n}\Big(1+\mathbf{O}(\frac{w}{k})\Big) \\ &= a\frac{w}{n}\Big(\log(\frac{w}{n}) + \log(\frac{n}{k})\Big) + a\frac{w}{n}\Big(1+\frac{n}{k}\mathbf{O}(\frac{w}{n})\Big) \\ &= a\mathbf{o}(1) + a\log\Big(\frac{n}{k}\Big)\mathbf{o}(1) + a\mathbf{o}(1) = \mathbf{o}(1). \end{aligned}$$

□

Théorème 3.1.3 *Soient* $\mathcal{A}$ *un algorithme de décodage générique et* $\mathscr{F}$ *une famille de codes linéaires* $[n,k,w]$ *tels que*

$$\lim_{n\to\infty} \frac{k}{n} = R \qquad \lim_{n\to\infty} \frac{w}{n} = 0.$$

Si $l \in]\,0, n-k[$ *et* $p \in]\,0, w\,[$ *son paramètres du décodage de* $\mathcal{A}$ *en un code de* $\mathscr{F}$ *et le facteur de travail*

$$\mathrm{WF}_{\mathcal{A}} := \mathrm{WF}_{\mathcal{A}}(l,p) \geq \frac{\binom{n}{w}}{\binom{n-k-l}{w-p}\binom{k+l}{ap}} \qquad \textit{avec} \qquad 2^l = \binom{k+l}{ap}$$

où $0 < a < 1$ *est une constante, alors*

$$\min \log(\mathrm{WF}_{\mathcal{A}}) \geq w\big(-\log(1-R) + \mathbf{o}(1)\big).$$

*D*émonstration
Grâce au lemme, on peut simplifier la borne $\mathfrak{B}_a$ par le prix d'un facteur $2^{\mathbf{o}(w)}$.
On étudie le quotient

$$\begin{aligned}
\log\Big(\binom{n-k-l}{w-p}/\binom{n-k}{w-p}\Big) &= (n-k-l)h(\frac{w-p}{n-k-l}) - (n-k)h(\frac{w-p}{n-k}) \\
&= -(w-p)\Big(\log\Big(\frac{w-p}{n-k-l}\Big) + \Big(1+\mathbf{O}(\frac{w-p}{n-k-l})\Big)\Big) \\
&\quad +(w-p)\Big(\log\Big(\frac{w-p}{n-k}\Big) + \Big(1+\mathbf{O}(\frac{w-p}{n-k})\Big)\Big) \\
&\leq (w-p)\Big(\log\big(1-\frac{l}{n-k}\big) + \mathbf{O}(\frac{w-p}{n-k-l})\Big) \\
&\leq w\Big(\log\Big(1-\frac{l}{n}\Big) + \frac{n}{n-k-l}\mathbf{O}\Big(\frac{w}{n}\Big)\Big) \\
&= w\mathbf{o}(1)
\end{aligned}$$

De la même manière, on obtient

$$\log\Big(\binom{k+l}{ap}/\binom{k}{ap}\Big) \leq ap\Big(\log(1+\frac{l}{k}) + \frac{n}{k+l}\mathbf{O}(\frac{w}{n})\Big) \leq w\mathbf{o}(1).$$

On déduit que

$$\mathrm{WF}_{\mathcal{A}} \geq 2^{\mathbf{o}(w)}\frac{\binom{n}{w}}{\binom{n-k}{w-p}\binom{k}{ap}}.$$

Si on veut minimiser le cote droite de cet expression, il faut maximiser la fonction

$$f(p) = (n-k)h(\frac{w-p}{n-k}) + kh(\frac{ap}{k})$$

Si on évalue la dérive respecte à p on obtient

$$\begin{aligned}
\frac{df}{dp} &= -\Big(-\log(\frac{w-p}{n-k}) + \log(1-\frac{w-p}{n-k})\Big) + a\Big(-\log(\frac{ap}{k}) + \log(1-\frac{ap}{k})\Big) \\
&= \log\big(\frac{w-p}{n-k-(w-p)}\big) - a\log\big(\frac{ap}{k-ap}\big)
\end{aligned}$$

Si on fait $\frac{df}{dp} = 0$, on obtient

$$\frac{w-p}{n-k-(w-p)} = \frac{(ap)^a}{(k-ap)^a}$$

$$\frac{(k-p)^a p^{1-a}}{n-k-(w-p)} = \frac{a^a p}{w-p}$$

Donc,

$$\frac{k^a w^{1-a}}{n-k-w} \geq a^a \frac{p}{w}$$

$$\frac{(k/n)^a(w/n)^{1-a}}{1-k/n-w/n} \geq a^a \frac{p}{w}.$$

On déduit que $\frac{p}{w} = \mathbf{O}(\frac{w}{n}^{1-a})$; en particulier $p = \mathbf{o}(w)$. Finalement, on analyse $\log(\mathrm{WF}_{\mathcal{A}})$

$$\log(\mathrm{WF}_{\mathcal{A}}) \geq 2^{\mathbf{o}(w)} \frac{\binom{n}{w}}{\binom{n-k}{w-p}\binom{k}{ap}},$$

Alors,

$$\log \mathrm{WF}_{\mathrm{Prange}} 2^{\mathbf{o}(w)} \geq \underbrace{nh(\frac{w}{n})}_{(1)} - \underbrace{(n-k)h(\frac{w-p}{n-k})}_{(2)} - \underbrace{kh(\frac{ap}{k})}_{(3)}.$$

Et, on les développe

$$\begin{aligned}
(1) \quad &: \quad nh(\tfrac{w}{n}) = -n\Big(\frac{w}{n}\log\big(\frac{w}{n}\big) + \big(1-\frac{w}{n}\big)\log(1-\frac{w}{n})\Big) = -w\log\big(\frac{w}{n}\big) - n\big(1-\frac{w}{n}\big)\log\big(1-\frac{w}{n}\big) \\
(2) \quad &: \quad -(n-k)h\big(\frac{w-p}{n-k}\big) = (w-p)\log(\frac{w-p}{n-k}) + (n-k)(1-\frac{w-p}{n-k})\log(1-\frac{w-p}{n-k}) \\
(3) \quad &: \quad -kh\big(\frac{ap}{k}\big) = ap\log\big(\frac{ap}{k}\big) + k\big(1-\frac{ap}{k}\big)\log\big(1-\frac{ap}{k}\big)
\end{aligned}$$

Comme dans le calcul de $\mathrm{WF}_{\mathrm{Prange}}$, on groupe dans deux sommes et on les développe en prenant en compte que $\frac{w-p}{n-k}$ et $\frac{p}{k}$ sont petits (plus petits que $\frac{w}{n-k}$ et $\frac{w}{k}$, respectivement) :

$$\begin{aligned}
(I) &= -w\log\big(\frac{w}{n}\big) + (w-p)\log\big(\frac{w-p}{n-k}\big) + ap\log\big(\frac{ap}{k}\big) \\
(II) &= -w\Big(1+\mathcal{O}(\frac{w}{n})\Big) + (w-p)\Big(1+\mathbf{O}(\frac{w-p}{n-k})\Big) + ap\Big(1+\mathbf{O}(\frac{ap}{k})\Big)
\end{aligned}$$

On continue par la partie facile

$$\begin{aligned}
(II) &= -w\mathbf{O}\big(\frac{w}{n}\big) + (a-1)p + (w-p)\mathbf{O}\big(\frac{w-p}{n-k}\big) + ap\mathbf{O}\big(\frac{p}{k}\big) \\
&= -w\mathbf{o}(1) + (a-1)\mathbf{o}(w) + (w-p)\mathbf{o}(1) + \mathbf{o}(w) \\
&= \mathbf{o}(w)
\end{aligned}$$

Finalement,

$$\begin{aligned}
(I) &= w\Big(\log(\frac{w-p}{n-k}) - \log(\frac{w}{n})\Big) + p\Big(a\log(\frac{ap}{k}) - \log(\frac{w-p}{n-k})\Big) \\
&= w\log\big(\frac{w-p}{w}/\frac{n-k}{n}\big) + p\log\big(a^a\frac{p^a}{w-p}\frac{n-k}{k^a}\big) \\
&= w\log\Big(\frac{1-p/w}{1-k/n}\Big) + pa\log(a) + p\log\Big(\frac{p^a}{w^a}\Big) + p\log(\frac{w^a}{(w-p)^a}) + p\log\Big(\frac{n-k}{k^a(w-p)^{1-a}}\Big) \\
&= w\log\Big(\frac{1-p/w}{1-k/n}\Big) + w\Big(a\frac{p}{w}\log(a) + a\frac{p}{w}\log(\frac{p}{w}) + a\frac{p}{w}\log(\frac{w}{w-p}) + a\frac{p}{w}\log\big(\frac{n-k}{k}\big) + \frac{p}{w}\log\Big(\frac{(n-k)^{1-a}}{(w-p)^{1-a}}\Big)\Big) \\
&= w\log\Big(\frac{1-p/w}{1-k/n}\Big) + w\Big(a\mathbf{o}(1) + a\mathbf{o}(1) + a\mathbf{o}(1)\mathbf{o}(1) + a\mathbf{o}(1)\mathbf{O}(1) + (1-a)\frac{p}{w}\log\big(\frac{n-k}{w-p}\big)\Big) \\
&= w\log\Big(\frac{1-p/w}{1-k/n}\Big) + w\Big(\mathbf{o}(1) - (1-a)\frac{p}{w}\big(\mathbf{O}(1) + \log(\frac{n}{w})\big)\Big) \\
&= w\log\Big(\frac{1-p/w}{1-k/n}\Big) + w\Big(\mathbf{o}(1) + \frac{p}{w}\log\big(\frac{w}{n}^{1-a}\big)\Big) \\
&= w\log\Big(\frac{1-p/w}{1-k/n}\Big) + w\big(\mathbf{o}(1) + \mathbf{o}(1)\big)
\end{aligned}$$

On conclut que pour des grandes valeurs de n, on a

$$\log(\mathrm{WF}_{\mathcal{A}}) \geq (I) + (II) = w(\log(\frac{1}{1-R}) + \mathbf{o}(1)).$$

□

3.2 Applications de la borne

On dans cette section on utilisera cette borne pour comparer la complexité asymptotique des algorithmes examinés avec l'algorithme de Prange. On avait déjà comparé la complexité de l'algorithme de Lee et Brickell avec ce de Prange et on avait trouvé que asymptotiquement ces complexités sont égaux ; on verra que c'est le même cas pour le reste des algorithmes.

On commencer pour mentionner que la complexité algorithmique optimal des autres algorithmes est plus petit que la de Prange. Par exemple, il faut faire les paramètres égaux tous à 0 pour obtenir, un facteur polynomial près, la même complexité de Prange ; donc pour les paramètres optimaux, ils seront plus petit.

On continue cet analyse en utilisant le corollaire 1.2.6, donc il faudra indiquer que l'exactitude de ce corollaire. Cependant, cet exactitude et notre objectif sont d'accord, donc les calculs prochains seront à un facteur polynomial près sans perte de généralité.

Algorithme de Stern et l'algorithme de Dumer :
Pour des valeurs modérés de w, c'est-à-dire $w \leq k/2$, on a normalement $\mathrm{WF}_{\mathrm{Stern}} \geq \mathrm{WF}_{\mathrm{Dumer}}$ (pour des valeurs différents on peut procéder de

la même manière). Donc, on obtient

$$\begin{aligned}\mathrm{WF}_{\mathrm{Dumer}} &\geq \frac{\binom{n}{w}}{\binom{n-k-l}{w-p}}\left(\frac{1}{\binom{(k+l)/2}{p/2}}+\frac{1}{2^l}\right)\\ &\geq \frac{\binom{n}{w}}{\binom{n-k-l}{w-p}}\left(\frac{1}{\sqrt{\binom{k+l}{p}}}+\frac{1}{2^l}\right)\\ &\geq \frac{\binom{n}{w}}{\binom{n-k-l}{w-p}}\left(\frac{1}{\binom{k+l}{\frac{1}{2}\cdot p}}+\frac{1}{2^l}\right)\end{aligned}$$

En conséquence, $\mathrm{WF}_{\mathrm{Dumer}}(l,p) \geq \mathfrak{B}_{\frac{1}{2}}(l,p)$,

Algorithme de May Meurer et Thomae :

On utilisera fait que $\binom{p}{p/2} = \tilde{O}(2^p)$ et la condition $l_2 < p$ pour que la probabilité de réussite de la méthode *AssocierColonnes* soit au moins $1/2$ (décrit dans le théorème 2 en [3]). Donc, on obtient

$$\begin{aligned}\mathrm{WF}_{\mathrm{MMT}}(l,p) &\geq \frac{\binom{n}{w}}{\binom{n-k-l}{w-p}}\left(\frac{\sqrt{\binom{k+l}{p/2}}}{\binom{k+l}{p}}+\frac{\binom{k+l}{p/2}^2}{\binom{k+l}{p}2^{l+l_2}}\right)\\ &\geq \frac{\binom{n}{w}}{\binom{n-k-l}{w-p}}\left(\frac{\sqrt[4]{\binom{k+l}{p}}}{\binom{k+l}{p}}+\frac{\binom{k+l}{p/2}\binom{p}{p/2}}{\binom{k+l-p/2}{p/2}2^{l+l_2}}\right)\\ &\geq \frac{\binom{n}{w}}{\binom{n-k-l}{w-p}}\left(\frac{1}{\binom{k+l}{p}^{\frac{3}{4}}}+\frac{2^p}{2^{l+l_2}}\right)\\ &\geq \frac{\binom{n}{w}}{\binom{n-k-l}{w-p}}\left(\frac{1}{\binom{k+l}{\frac{3}{4}p}}+\frac{1}{2^l}\right).\end{aligned}$$

En conséquence, $\mathrm{WF}_{\mathrm{MMT}}(l,p) \geq \mathfrak{B}_{\frac{3}{4}}(l,p)$.

Algorithme de Becker, Joux, May et Meurer :

On utilisera le fait que $p_2 \leq \frac{p}{4}$ et que μ_1 peut être une probabilité. Donc, on obtient

$$\begin{aligned}\mathrm{WF}_{\mathrm{BJMM}}(l,p) &\geq \frac{\binom{n}{w}}{\binom{n-k-l}{w-p}}\Big(\frac{\sqrt{\binom{k+l}{p_2}}}{\binom{k+l}{p}}+\frac{1}{\mu_1 2^l}\Big)\\ &\geq \frac{\binom{n}{w}}{\binom{n-k-l}{w-p}}\Big(\frac{\sqrt[8]{\binom{k+l}{p}}}{\binom{k+l}{p}}+\frac{1}{2^l}\Big)\\ &\geq \frac{\binom{n}{w}}{\binom{n-k-l}{w-p}}\left(\frac{1}{\binom{k+l}{\frac{7}{8}p}}+\frac{1}{2^l}\right).\end{aligned}$$

En conséquence, $\mathrm{WF}_{\mathrm{BJMM}}(l,p) \geq \mathfrak{B}_{\frac{7}{8}}(l,p)$.

Grâce au théorème, on conclut

$$\lim_{n\to\infty} \min_{l,p} \log(\mathrm{WF}_{\mathcal{A}}) \geq \log(\mathrm{WF}_{\mathrm{Prange}}),$$

pour $\mathcal{A} =$ Stern, Dumer, MMT, BJMM ; en conséquence, ces algorithmes ont la même complexité à un facteur polynomial près.

CHAPITRE 4

Logiciel de calcul de complexité asymptotique

On a déjà obtenu des formules pour la complexité algorithmique des algorithmes présentés pour ces paramètres respectifs, donc on trouve naturellement le problème : trouver les paramètres pour lesquels la complexité est minimale et cette complexité pour un longuer fixé n des mots. À cause de la nature des formules trouvées ce problème est difficile et appartient à la branche de la Combinatoire et Optimization Discrète ; donc on opte pour transformer ce problème en un problème de optimization en $\mathbb{R}$. Cette nouvelle problème aura la même valeur minimal à un facteur polynomial près ; ce défaut ne sera pas important si on considére que la complexité des algorithmes est déjà exponentielle respecte à n .

Pour faire cette transformation on utilisera la fonction entropie binaire et le corollaire 1.2.3 et les paramètres seront divisés par n. De cette manière, la complexité sera exprimée directement une fonction exponentielle en base 2 et les paramètres seront dans $[0, 1]$. Après avoir obtenu les paramètres optimals on exprimera la complexité d'un algorithme $\mathcal{A}$ de la forme $\tilde{O}(2^{c_{\mathcal{A}}n})$ où $c_{\mathcal{A}}$ dépende de l'algorithme en question. Donc, l'objectif fondamental de cet algorithme sera trouver le coefficient exponentielle $\mathcal{A}$.

Libraire scientifique GNU (GSL)

GSL est une collection de routines de calcul numérique écrits dans le langage de programmation C sous license GNU. On utilisera cette libraire pour résoudre des problèmes de minimization à plusiers variables et aussi calculer la solution de certaines équations à une variable réelle (que l'on discutera après). Comme les fonctions seront au moins continues, on aura toujours une solution pour trouver.

On décrira le cas pour la minimization d'une fonction à plusiers variables parce que c'est la plus complique. GSL exige un modèle stardant pour les fonctions à plusiers variables avec lesquelles se feront les calculs

```
double ma_fonction( const gsl_vector *v, void * params)
```

où gsl_vector est le type qui émule les vecteurs en GSL et la variable params est un point à void qui permet utiliser n'import quel type de paramètre pour la fonction. Donc, la méthode de minimization aura toujours la structure suivante

```
main()
{
  //déclaration et initialisation de méthode de minimisation
  const gsl_multimin_fminimizer_type *T=gsl_multimin_fminimizer_nmsimplex2;
  //déclaration du mininisateur
  gsl_multimin_fminimizer *s;
  //déclaration de la variable qui garde les spécifications de la fonction à optimiser
  gsl_multimin_function F;

  //déclaration et initialisation du point du départ
  gsl_vector *x=gsl_vector_alloc(n);
  gsl_vector_set(x,0,x0);
      :
      :
  gsl_vector_set(x,0,xn);
  //déclaration et initialisation de la taille du pas dans la minimisation
  gsl_vector *ss=gsl_vector_alloc(n);
  gsl_vector_set(ss,0,ss0);
      :
      :
  gsl_vector_set(ss,0,ssn);
  //initialisation de la variable qui garde les spécifications de la fonction à optimiser
  F.n=n;
  F.f =ma_fonction;
  F.params=params;

  //initialisation du mimimisateur
  s= gsl_multimin_fminimizer_alloc(T,n);
  gsl_multimin_fminimizer_set(s,&F,x,ss);

  //boucle de minimisation
  do
    {
     //incrementation de la variable de control
     iter++;
     //itération du minimisateur
     gsl_multimin_fminimizer_iterate(s);
     //récuperation du taille caracteristique de la minimisation
     size=gsl_multimin_fminimizer_size(s);
    }//évaluation de la taille de size respecte à delta ou nombre de itération
     while (gsl_multimin_test_size(size,delta) == GSL_CONTINUE && iter<max_iter);

  //Récuperation du valeur mimimal
  min=s->fval;

  //Libération de la mémoire asignée
  gsl_multimin_fminimizer_free(s);
  gsl_vector_free(ss);
  gsl_vector_free(x);

}
```

La recherche de la solution d'une équation à une variable réelle du type $f(x) = 0$ garde une structure très similaire avec une initialisation des spécifications du résolution, itération de solutioneur et un controle du boucle.

4.1 Le logiciel ACCDA

ACCDA est logiciel écrit en C qui utilise la libraire GLS pour calculer le coefficient exponentiel de la complexité asymptotique des algorithmes présentés pour des paramètres optimaux. Il conste d'un programme principal ACCDA.c

et trois fichiers en-tête entropy_tools.h, work_factor.h et low_bound.h (avec ses respectifs fichiers d'implementation) ; de plus, il a aussi une application bound.c qui établis le lien entre les résultats calculés de la minimisation et les bornes théorique déjà réconnues.

```
/ACCDA
   |
   |-/bound_test
   |    |-bound.c
   |    \-Makefile
   |-/include
   |    |-entropy_tools.h
   |    |-low_bound.h
   |    \-work_factor.h
   |-/src
   |  |-accda.c
   |  |-entropy_tools.c
   |  |-Makefile
   |  |-low_bound.c
   |  \-work_factor.c
   \Makefile
```

Le programme accda calcule plus petit coefficient exponentiel d'un des algorithmes étudiés, soit de Prange, Stern, Dumer, MMT ou BJMM. Pour cela, les fonctions implémentées en entropy_tools.c aident à implémenter les fonctions que calculent la complexité de ces algorithmes pour des paramètres spécifiques dans work_factor.c. Donc, on effectue la minimisation de ces fonctions et on obtient les paramètres optimaux pour ces valeurs minimals. De cette façon on obtientla rélation de du coefficient exponentiel minimal en rapport au taux de transmission k/n pour les algorithmes présentés ; pour cela on a décide de choisir w/n tel que $k/n = 1 - h(k/n)$, c'est-à-dire, le taux de transmission atteint la borne de Gilbert-Varshamov (voir le diagramme 4.2).

Implementation de la fonction de borne

Dans le chapitre précédente, on avait trouvé des fonctions bornes $\mathfrak{B}_a$ pour les comparer avec les algorithmes présentés. Donc, on se pose la question si la complexité d'un algorithme peut être exprime pour une fonction borne $\mathfrak{B}_a$ pour un certain $a \in]\,0, 1\,[$. Dans le ficher low_bound.c on a implémente cette fonction, sa minimisation et la résolution du problème

$$\text{Pour } k/n \text{ choisi, trouver } a \in]0, 1[\text{ tel que } \min_{p,l} \mathfrak{B}_a = \min \mathrm{WF}_{\mathcal{A}}(k/n).$$

On a résolu ce problème dans le programme bound.c pour tous les algorithmes précédentes qui utilise le fichier en tête low_bound.h.

Mise en œuvre

La construction du programme a commencé par résoudre séparément le problème de minimisation de chaque algorithme dans l'ordre qu'ils sont présentés dans ce mémoire.

Problèmes et ses solutions

On trouvait presque toujours une approximation au minimum grâce au fonctions en GSL, mais les premières n'étaient jamais très proche du minimum. À continuation, on expose les problèmes et solutions que nous avons obtenu pendant la préparation ce logiciel.

Le bon point de départ Le premier problème de la minimisation a été trouver des bons paramètres pour initialiser le point de départ et la taille des pas dans l'itération. Pour cela, on a opté pour établir les valeurs maximal des arguments de la fonction qui implémentait la coefficient exponentiel et on choisi un point départ en proportion à ce valeur maximal. Après, les points de départs on été corriges en remplaçant par des points plus proche à pseudo-solution obtenue avant (toujours en proportion à maximum valeur possible de l'argument).

La bonne taille du pas Un problème en rapport au premier était combien grand on faisait le pas pour choisir les nouveaux points où itérer la minimisation. Une taille trop petite empêchait de avancer entre les points intermédiaires au résultat, une taille trop grande ne permettant pas de la précision au moment de trouver le minimum.

La bonne région d'évaluation Étant donné que nous utilisons des formules avec des logarithmes, on obtient des restrictions sur les arguments des fonctions. Car la procédure de minimisation avance de façon indistincte, on a opte pour assigner un valeur no minimisant quand la fonction prend ces arguments. De cette façon, on continue la minimisation et on s'éloigne des arguments interdits.

Le bon nombre des arguments Ce problème apparait pour les algorithme de MMT et BJMM parce qu'ils utilisent 3 et 4 paramètre, respectivement. Donc, il devient très difficile à résoudre les deux premiers problèmes et la minimisation obtient fausses résolutions récurrentes. On a résolu ce problème en faisant des optimisation par niveau, c'est-à-dire, on a obtenu le minimum respecte à deux paramètre et après respecte aux paramètres restants, cela a permis de choisir le minimum entre les minimum trouvés. Par contre, cela demande beaucoup plus des itérations.

Le bon minimum Le problème final est vérifier si le minimum obtenu était le vrai minimum. Pour cela, on a opté pour comparer de trois façon différentes :

1. La relation entre les points minimum pour chaque taux de transmission. Si on avait de l'uniformité, cela indiquait que le choix des paramètres était stable.
2. La relation entre les algorithmes précédentes. Si la courbe était inférieur à une courbe de l'algorithme précédente, l'optimisation était bonne.
3. La relation entre le paramètre a obtenu par le programme `bound.c` pour chaque taux transmission. Une courbe plate indique que les valeurs sont homogénes et correspondent au vrai minimum.

Limitations, problèmes subsistants et futurs améliorations

Étant donné la nature des formules utilises et le processus de minimisation, le logiciel est plus limité de ce que l'on désire. À continuation, on expose les limites et problèmes subsistent que nous avons jusque maintenant.

Exactitude L'actuelle version a une exactitude de 0.01 pour le taux de transmission. Si on exige une exactitude 0.001, on n'obtiendra pas toujours des bons valeurs (5% des valeurs fausses).

Capacité de correction On a utilise que $w/n = 1 - h^{-1}(k/n)$ pour la capacité de correction des erreurs du code. Cependant, on peut utiliser w/n tel que $k/n = 1 - h(2w/n)$, comme on suggère en [3] et [1] avec le même succès et pour valeurs compris entre eux. Mais résoudre tous les difficultés déjà discutés pour tous les coupes de valeurs n'est pas toujours possible.

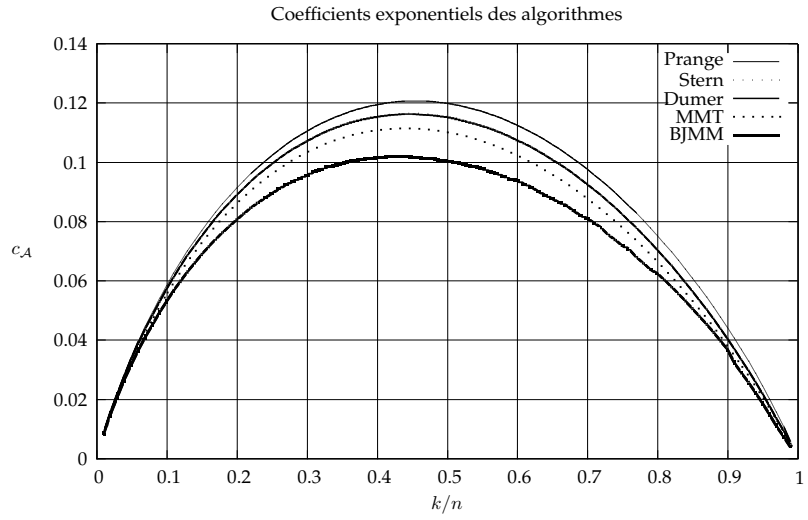
Région de conflit Surtout pour l'implémentation de la fonction borne $\mathcal{B}_a$ les régions où le calcul est difficile n'est pas toujours la même. L'argument a ne donne pas seulement des valeurs différents sinon aussi des régions de définition différentes, cela impose un problème récurrents pour la minimisation.

On envisage résoudre ces limitations et problèmes par un étude statistique des valeurs à majeur profondeur. De plus, on veut calculer aussi la complexité algorithme pour la mémoire utilisée dans chaque algorithme pour avoir une majeur vision de coût de ces algorithmes.

4.2 Résultats

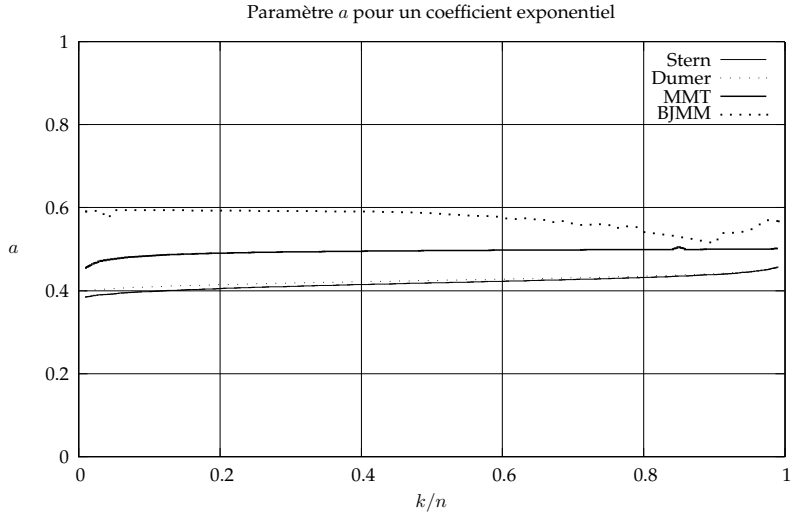
À continuation, on montre le diagramme du coefficient exponentiel respect au taux de transmission pour chaque algorithme.

On signale que la courbe correspondante au algorithme de Stern est couverte par la courbe de Dumer, parce que les valeurs minimales sont très simi-



laire (avec des paramètres optimaux différentes). On vérifie l'évolution des méthodes de décodage vers une meilleure complexité algorithmique, c'est-à-dire, un coefficient exponentiel plus petit. Il faut mentionner que les articles [3] et [1] calculent le coefficient exponentiel pour un taux de transmission k/n et un taux d'erreurs $2 \cdot w/n = h^{-1}(1 - k/n)$ parce qu'ils veulent évaluer des paramètres qui servent pour des cryptosystèmes. Le logiciel a bien vérifié les valeurs obtenus dans ces articles

Le prochaine diagramme montre la relation entre le paramètre a obtenu par le programme `bound.c` et le taux de transmission. Il présente encore des défauts sur un de les extrêmes de la courbe correspondent à BJMM, cela est causé pour la manque de précision du minimum obtenu de $\mathfrak{B}_a$.



Si on prend en content la définition de $\mathcal{B}_a$ et lemme 3.1.1, on trouve que le valeur du minimum devra se réduire quand a accroît. Pour cela, on obtient un paramètre a plus grand pour l'algorithme de BJMM et le plus petit pour Stern et Dummer. Aussi, chaque courbe respecte les valeurs théoriques obtenus pour leurs respectives fonctions borne trouvées dans le chapitre précédent, par exemple, la courbe de MMT est dessous de $\frac{3}{4}$.

Conclusion

Après 6 mois de stage dans l'équipe-projet SECRET de INRIA Paris - Rocquencourt sous la direction de Nicolas Sendrier, on a obtenu un précieux résultat théorique décrit dans le troisième chapitre et aussi des résultats avec un logiciel de calcul décrit dans le quatrième chapitre. Maintenant, je les résume

1. Pour toute fonction de la famille $\mathfrak{B}_a$ le valeur minimal est attendu les deux termes additifs sont égaux.
2. Quand le nombre des erreurs à corriger dans mot est négligeable respect à sa longueur, les algorithmes plus récents ont une complexité asymptotique égal à la complexité de Prange.
3. Les algorithmes décrits ont évalué selon l'ordre présenté quand le nombre des erreurs est proportionnel à la longueur des mots.
4. Pour estimer la sécurité du cryptosystème de McEliece contre de le décodage générique quand l'erreur aléatoire ajouté a un poid petit, il suffit de regarder la complexité algoritmique de Prange pour obtenir un valeur estime des itérations
5. Les cryptosystèmes basés sur des codes comme Goppa ou les codes MPDC sont forts contre l'attaque de décodage générique.
6. Le coefficient exponentiel estimé dans les article sur l'algorithme de MMT et l'algorithme de BJMM est exact.

Annexes

ANNEXE A

Preuve du lemme 3.1.1

On calcule

$$\begin{aligned}\log(\mathfrak{B}_a(l,p)) &= \log\Big(\frac{\binom{n}{k}}{\binom{n-k-l}{w-p}2^l} + \frac{\binom{n}{k}}{\binom{n-k-l}{w-p}\binom{k+l}{ap}}\Big)\\ &= \max\Big\{\log\Big(\frac{\binom{n}{k}}{\binom{n-k-l}{w-p}2^l}\Big), \log\Big(\frac{\binom{n}{k}}{\binom{n-k-l}{w-p}\binom{k+l}{ap}}\Big)\Big\} + c_a(l,p)\end{aligned}$$

où $c(l,p) \in]\,0,1\,]$, donc on peut oublier $c_a(l,p)$ dans notre analyse asymptotique. On fait

$$A(l,p) = \log\Big(\frac{\binom{n}{k}}{\binom{n-k-l}{w-p}2^l}\Big) \quad \text{et} \quad B(l,p) = \log\Big(\frac{\binom{n}{k}}{\binom{n-k-l}{w-p}\binom{k+l}{ap}}\Big).$$

On notre but est de montrer qu'il existe $(\hat{l},\hat{p}) \in D$ tel que $A(\hat{l},\hat{p}) = B(\hat{l},\hat{p})$ qui minimise $\mathfrak{B}_a$. On commence par étudier les points dans l'intérieur de D et vérifier que si $\mathfrak{B}_a$ atteint son minimum en (l^*,p^*) alors il y existe $(\hat{l},\hat{p}) \in \mathcal{V}$ où $\mathfrak{B}_a$ a le même valeur ; après on vérifiera que l'unique possible minimum dans la frontière ∂D est atteint que aux points de $\mathcal{V}$; ces deux affirmations nous permettent conclure le résultat.

On suppose que $(l^*,p^*) \notin \partial D$ minimise $\mathfrak{B}_a$ et que $B(l^*,p^*) > A(l^*,p^*)$. Dans ce cas, $B(l,p) = \max\{A(l,p), B(l,p)\}$ pour tout (l,p) dans une voisinage U de (l^*,p^*). Donc,

$$\min_{(l,p)\in D} \mathfrak{B}_a(l,p) = \min_{(l,p)\in U} \mathfrak{B}_a(l,p) = B(l,p),$$

et en particulier $\nabla B(l^*,p^*) = (0,0)$. On analyse les dérivées partielles de l'approximation asymptotique de $B(l,p)$

$$\begin{aligned}\frac{\partial B}{\partial l} &= -\log(1-\frac{w-p}{n-k-l}) + \log(1-\frac{ap}{k+l})\\ \frac{\partial B}{\partial p} &= h'(\frac{w-p}{n-k-l}) - ah'(\frac{ap}{k+l})\end{aligned}$$

Si on fait $\nabla B(l^*, p^*) = 0$ alors

$$\frac{w-p^*}{n-k-l^*} = \frac{ap^*}{k+l^*} \quad \text{et} \quad h'(\frac{w-p^*}{n-k-l^*}) = ah'(\frac{ap^*}{k+l^*}),$$

et comme $a \in]0,1[$, ces équations ont comme unique solutions

$$\frac{w-p^*}{n-k-l^*} = \frac{p^*}{k+l^*} = 0,$$

ce qui est absurde.

Si au contraire on a $A(l^*, p^*) > B(l^*, p^*)$, alors $\nabla A = (0,0)$. Donc

$$\begin{aligned} \frac{\partial A}{\partial l} &= -\log(1 - \frac{w-p^*}{n-k-l^*}) - 1 = 0. \\ \frac{\partial A}{\partial p} &= h'(\frac{w-p^*}{n-k-l^*}) = 0 \end{aligned}$$

En conséquence,

$$\mathscr{L}^* : \quad \frac{w-p^*}{n-k-l^*} = \frac{1}{2},$$

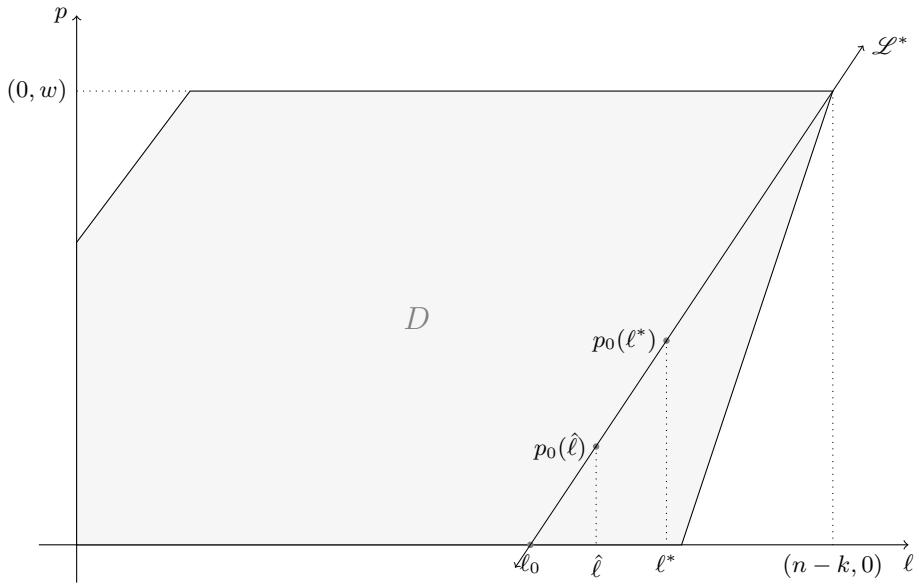
cet équation définit une droite dans laquelle $A(l,p)$ ne change pas de valeur. On define la fonction $p_0 : [\, n-k-2w, n-k\,] \to \mathbb{R}$ par $p_0(l) = w - \frac{n-k-l}{2}$ pour décrire les points de cette droite. Donc, notre objectif est montrer qu'il y existe un point qui appartient à $\mathcal{V}$ dans la droite $\mathscr{L}^*$. Par hypothèse, $\ell_0 = n-k-2w > 0$, donc $(\ell_0, p_0(\ell_0)) = (\ell_0, 0) \in \mathscr{L}^* \cap \mathcal{D}$ et

$$A(\ell_0, p_0(\ell_0)) = nh\big(\frac{w}{n}\big) - (n-k-\ell_0) - \ell_0 < nh\big(\frac{w}{n}\big) - (n-k-\ell_0) = B(\ell_0, p_0(\ell_0)).$$

Parce que $A(\ell^*, p_0(\ell^*)) < B(\ell^*, p_0(\ell^*))$ et le segment de droite entre $(\ell_0, p_0(\ell_0))$ et $(\ell^*, p_0(\ell^*))$ est en $\mathcal{V}$, il y a $\hat{\ell} \in]\ell_0, \ell^*[$ tel que $A(\hat{\ell}, p_0(\hat{\ell})) = B(\hat{\ell}, p_0(\hat{\ell}))$. Car $A(\hat{\ell}, p_0(\hat{\ell})) = A(\ell*, p(\ell^*))$, on conclut que $(\hat{\ell}, p_0(\hat{\ell}))$ est le point minimal pour $\mathfrak{B}_a$ et il appartient à $\mathcal{V}$.

Maintenant, on suppose que le point (l^*, p^*) qui minimise $\mathfrak{B}_a$ appartient à ∂D et on cherchera des possibles candidats dans la frontière. Pour cela, on divise la frontière de D dans 5 segments de droite et on analyse la croissance ou décroissance de A et B.

- Dans le segment $p = 0$, $B \geq A$, A et B es croissante par rapport à l, donc $\min \mathfrak{B}_a = B(0,0)$.
- Dans le segment $\frac{w-p}{n-k-l} = 1$, A et B décroisent respecte à l, donc $\min \mathfrak{B}_a = \max\{A(n-k, w), B(n-k, w)\}$.
- Dans le segment $p = w$, A et B décroisent aussi par rapport à l , donc $\min \mathfrak{B}_a = \max\{A(n-k, w), B(n-k, w)\}$.
- Dans le segment $\frac{ap}{k+l} = 1$, $B \geq A$ et B est croissant respecte à l, donc $\min \mathfrak{B}_a = B(k/a, 0)$.
- Dans le segment $l = 0$, $A \geq B$ et A est croissant respecte a p, donc $\min \mathfrak{B}_a = A(0,0)$.



On obtient que

$$\min_{\partial D} \mathfrak{B}_a(l,p) = \min\{A(0,0), B(0,0), B(k/a,0), \max\{A(n-k,w), B(n-k,w)\}\};$$

Car $B(0,0) = A(0,0) \leq A(k/a,0) = B(k/a,0)$, on résume notre analyse aux points $(0,0)$ et $(n-k,w)$, donc il suffit étudier le cas $(l^*,p^*) = (n-k,w)$. Pour tout $m \leq 1$, on peut analyser B sur la droite

$$\mathscr{L}_m : \frac{w-p}{n-k-l} = m.$$

On a la dérivée

$$\begin{aligned}\frac{\partial B}{\partial l}(l,p) &= h(m) - (h(\frac{ap(l)}{k+l}) - h'(\frac{ap(l)}{k+l})(ap'(l) - \frac{ap(l)}{k+l})) \\ &= h(m) - (-am\log(\frac{ap}{k+l}) - (1-am)\log(1-\frac{ap}{k+l}))\end{aligned}$$

Si on fait $m_0 = \frac{w}{n-k} < \frac{1}{2}$, on décrit aussi la droite par

$$\mathscr{L}_{m_0} : \frac{p}{k+l} = \frac{w}{n},$$

cela nous conduit à

$$\frac{\partial B}{\partial l} = h(\frac{w}{n}) - (-a\frac{w}{n}\log(a\frac{w}{n}) - (1-a\frac{w}{n})\log(1-a\frac{w}{n})) = h(\frac{w}{n}) - h(a\frac{w}{n}) > 0,$$

parce que $w/n < 1/2$. Donc, B est décroisent, par rapport à l, dans cette droite et B n'atteint pas son minimum local en $(n-k,w)$, donc le minimum n'est pas en $\mathfrak{B}_a(n-k,w)$ si $B(n-k,w) > A(n-k,w)$.

Dans le cas où $\mathfrak{B}_a(l^*,p^*) = A(n-k,w) > B(n-k,w)$, alors on prendre n'importe quelle point (l^{**},p^{**}) de l'intérieur de D et à la droite $\mathscr{L}^*$ (avant décrit), et on obtient un point $(\hat{\cdot},\hat{l}) \in \mathcal{V}$ comme avant.

ANNEXE B

Minimisation de la fonction borne

On montre partie du code écrit dans le fichier `low_bound` correspondante à la minimisation d'une fonction borne. On peut voir comme les paramètres sont utilisés dans le modèle de fonction de GSL. La variable `wfp` garde les paramètres k/n et w/n, et elle est définie dans le fichier `accda.c`

```
extern wf_params wfp;

double
wf_BOUND( const gsl_vector *v,void * paramsa)
{
  double l, p,min,a;

  a = *((double *) paramsa);
  l = gsl_vector_get(v, 0);
  p = gsl_vector_get(v, 1);

  //We verify the paramètres before évaluation
  if ((l<0) || (l>(1-wfp.k-wfp.w))|| (wfp.w<p)|| (p>(wfp.k+l)/a)|| (p<0) )
    return 1;

  min=(wfp.k+l)*entropy(a*p/(l+wfp.k));
  if (min>l)
    min=l;

  return entropy(wfp.w)-(1-wfp.k-l)*entropy((wfp.w-p)/(1-wfp.k-l))-min;
}


double
wf_min_BOUND(double a, minimization_params *mp)
{
  int iter=0;
  const gsl_multimin_fminimizer_type *T=gsl_multimin_fminimizer_nmsimplex2;
  double size, wf,aa;

  gsl_multimin_fminimizer *s=NULL;
  gsl_multimin_function F;
  gsl_vector *ss,*x;

  //We initialize the parameters for minimization
  mp->l=0.5*(1-wfp.k-wfp.w);
  if (wfp.w<(wfp.k + mp->l)/a)
    mp->p=0.5*wfp.w;
  else
```

```
    mp->p=0.5*(wfp.k + mp->l)/a;

  //We choose positon vector for minimization
  x=gsl_vector_alloc(2);
  gsl_vector_set(x,0,mp->l);
  gsl_vector_set(x,1,mp->p);


  //We choose the step size for minimization
  ss=gsl_vector_alloc(2);
  gsl_vector_set_all(ss,0.01*mp->p);

  //We define function and its parameters
  F.n=2;
  F.f =wf_BOUND;
  //We transform the value a to a parameter variable aa
  aa=a;
  F.params=&aa;

  //We initialize the mimimizer
  s= gsl_multimin_fminimizer_alloc(T,2);
  gsl_multimin_fminimizer_set(s,&F,x,ss);

  do
    {
     iter++;
     //We iterate the minimization procedure
     gsl_multimin_fminimizer_iterate(s);
     //We obtain the characteristic size of minimization procedure
     size=gsl_multimin_fminimizer_size(s);
    }while (gsl_multimin_test_size(size,1e-10) == GSL_CONTINUE && iter<600);

  //We save the optimal parameters
  mp->l=gsl_vector_get(s->x,0);
  mp->p=gsl_vector_get(s->x,1);

  wf=s->fval;

  gsl_multimin_fminimizer_free(s);
  gsl_vector_free(ss);
  gsl_vector_free(x);

  return wf;
}
```

Bibliographie

[1] 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 D. Pointcheval and T. Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 520–536. Springer, 2012.

[2] I. Dumer. On minimum distance decoding of linear codes. In *Proc. 5th Joint Soviet-Swedish Int. Workshop Inform. Theory*, pages 50–52, Moscow, 1991.

[3] A. May, A. Meurer, and E. Thomae. Decoding random linear codes in $\tilde{O}(2^{0.054n})$. In D.H. Lee and X. Wang, editors, *Advances in Cryptology - ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 107–124. Springer, 2011.

[4] R.J. McEliece. A public-key cryptosystem based on algebraic coding theory. *DSN Prog. Rep., Jet Prop. Lab., California Inst. Technol., Pasadena, CA*, pages 114–116, January 1978.

[5] Rafael Misoczki, Jean-Pierre Tillich, Nicolas Sendrier, and Paulo S. L. M. Barreto. MDPC-McEliece : New McEliece variants from moderate density parity-check codes. In *IEEE Conference, ISIT 2013*, pages 2069–2073, Instanbul, Turkey, July 2013.

[6] E. Prange. The use of information sets in decoding cyclic codes. *IRE Transactions*, IT-8 :S5–S9, 1962.

[7] J. Stern. A method for finding codewords of small weight. In G. Cohen and J. Wolfmann, editors, *Coding theory and applications*, volume 388 of *LNCS*, pages 106–113. Springer, 1989.