Étude et conception d'algorithmes quantiques pour le décodage de codes linéaires

Ghazal Kachigar

▶ **To cite this version:**

Ghazal Kachigar. Étude et conception d'algorithmes quantiques pour le décodage de codes linéaires . Théorie de l'information [cs.IT]. 2016. hal-01371018

HAL Id: hal-01371018
https://inria.hal.science/hal-01371018

Submitted on 23 Sep 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Étude et conception d'algorithmes quantiques pour le décodage de codes linéaires

Rapport de stage de M2

Ghazal Kachigar

15/03/2016 - 15/09/2016

Résumé

J'ai effectué mon stage de fin de master 2 au sein de l'équipe-projet SECRET de l'INRIA de Paris. Mon stage a été encadré par Jean-Pierre Tillich.

Problématique du stage Bien que les ordinateurs quantiques de grande taille ne soient pas encore une réalité, de grandes avancées théoriques ont été faites dans le domaine de l'information quantique. En particulier, nous savons qu'il est possible d'obtenir une amélioration exponentielle du temps de résolution de certains problèmes tels que la factorisation et le logarithme discret. Par conséquent, l'avènement des ordinateurs quantique rendra des cryptosystèmes répandus qui sont basés sur ce type de problèmes, comme RSA et les courbes elliptiques, obsolètes. Pour cette raison, il y a actuellement une compétition organisée par le NIST afin de standardiser les cryptosystèmes dits post-quantiques puisqu'ils se fondent sur des problèmes qui auraient une meilleure résistance face à l'ordinateur quantique [14]. En vue de cela, il est nécessaire d'étudier l'impact des ordinateurs quantiques sur ces cryptosystèmes. Le cryptosystème de McEliece (1978) et le cryptosystème de Niederreiter (1986) qui est son dual sont de bons candidats de cryptosystèmes post-quantiques. Leur sécurité est donnée par la complexité du meilleur algorithme de résolution du problème de décodage générique que nous allons définir.

On se donne un code linéaire binaire C de paramètres $[n, k]$ ainsi qu'une matrice $G \in \mathcal{M}_{k,n}(\mathbb{F}_2)$ telle que $C = \{mG \mid m \in \mathbb{F}_2^k\}$, appelée la matrice génératrice du code C. On se donne également une matrice $H \in \mathcal{M}_{n-k,n}(\mathbb{F}_2)$ telle que $GH^T = 0$, appelée matrice de partié du code. Enfin, on considère un paramètre t.

Le problème du décodage générique est le suivant :

Problème 1 (Décodage générique). *Étant donné un mot de code $y = m + e$ où $m \in C$ et $w_H(e) \leq t$, trouver m (ou e).*

Ce problème est équivalent au problème suivant, appelé problème de décodage par syndrome :

PROBLÈME 2 (Décodage par syndrome). *Étant donné un syndrome* $s = Hy^T$ *où* $y = m + e$ *avec* $m \in C$ *et* $w_H(e) \leq t$*, trouver* e*.*

Dans ce mémoire, nous allons nous concentrer sur le problème du décodage par syndrome, mais il peut être plus intéressant de choisir l'une ou l'autre de ces approches en fonction de la fraction $\frac{k}{n}$, appelé taux de transmission du code et noté R. En effet, on peut montrer que le temps d'exécution des algorithmes de décodage générique (par syndrome ou non) dépend de R. Plus précisément, ils sont en temps $O(2^{\alpha(R)n})$. Nous nous intéresserons à l'analyse asymptotique (c'est-à-dire pour $n \to \infty$) de l'exposant α : dans ce cas, la distance minimale du code et par conséquent t est sur la borne de Gilbert-Varshamov $d_{GV} = nH^{-1}(1 - R)$. En effet, cela correspond à la plus grande valeur de t pour laquelle on peut espérer avoir une unique solution au problème du décodage.
Plus précisément, on distingue entre deux cas de figures :

1. *Half-distance decoding* : $t \leq \frac{d_{GV}}{2}$.
   Dans ce cas de figure, on a une solution unique que l'on obtient avec probabilité 1.
2. *Full-distance decoding* : $t \leq d_{GV}$.
   Dans ce cas de figure, en théorie la solution n'est plus unique, mais en pratique elle est unique avec une probabilité très proche de 1.

Le premier cas de figure est une situation idéale et c'est le deuxième cas de figure qui se rencontre dans la majorité des situations en pratique.

Le but est de voir jusqu'où on peut améliorer l'exposant en se servant d'algorithmes quantiques, en particulier l'algorithme de recherche de Grover et la marche aléatoire quantique.
Nous utiliserons par la suite la notation $\tilde{O}$: $f(n) = \tilde{O}(g(n)) \Leftrightarrow \exists k, f(n) = O(g(n) \log^k g(n))$. Cela nous permet en effet de faire abstraction du facteur polynomial et de nous concentrer uniquement sur le facteur exponentiel donné par la fonction α.
Pour cette même raison, nous nous concentrerons également sur les vecteurs d'erreurs de poids exactement t. En effet, en passant de ce cas de figure au cas général, la complexité augmente d'un facteur qui est au maximum égal à t et qui est donc polynomial.

État de l'art En décodage ISD classique, les meilleurs algorithmes de décodage générique de codes linéaires sont les algorithmes de décodage par ensemble d'information ou Information Set Decoding, dits algorithmes ISD. Ce sont des algorithmes probabilistes de type Las Vegas qui voient le décodage de codes linéaires comme la résolution d'un système linéaire sous-déterminé en raison de la contrainte de poids sur la solution. Le premier algorithme ISD est l'algorithme de Prange (1962). Il y a eu ensuite l'algorithme de Lee et Brickell (1988) qui n'améliore pas l'exposant asymptotique de l'algorithme de Prange, suivi des algorithmes de Stern (1989) et de Dumer (1991) qui améliorent l'exposant asymptotique en mettant en évidence un sous-problème de décodage par syndrome qu'il est possible de résoudre grâce à une recherche de collision. Les prochaines avancées sont venues à partir de 2011, en exploitant les ressemblances entre le problème de la somme des sous-ensembles et le problème du décodage

afin de transposer des techniques développées pour le premier problème au second. C'est ainsi que les algorithmes de May, Meurer et Thomae (2011) et de Becker, Joux, May et Meurer (2012) ont été développés. Enfin, en 2015, May et Ozerov ont adopté une autre approche en considérant le problème à résoudre comme un problème de recherche du voisin le plus proche.

Outre ces algorithmes "standards", il y a toute une série d'algorithmes qui n'ont pas fait l'objet de publications à part car ils consistent principalement en des modifications et des peaufinages des algorithmes "standards". Cependant, nombre de ces algorithmes ont été fondamentaux pour obtenir des améliorations en quantique, parmi lesquels l'algorithme de Shamir-Schroeppel et un algorithme que l'on a appelé $MMT^\sharp$ dans ce mémoire.

Dans le but de développer des algorithmes ISD quantiques, nous nous sommes intéressés principalement aux deux algorithmes quantiques suivants : l'algorithme de recherche non-structurée de Grover et la marche aléatoire quantique (Quantum Walk) ainsi que les différentes façons de les imbriquer et de se servir de structures de données. Ces algorithmes apportent une amélioration quadratique par rapport à leurs équivalents classiques puisque avec l'algorithme de Grover il est possible de trouver un élément en la racine carrée de l'espace de recherche et les temps d'arrêt et de retour de la marche aléatoire quantique sont la racine carrée de ceux de la marche classique correspondante.

Avant ce travail, il existait dans la littérature un seul algorithme ISD quantique : une version quantique de l'algorithme de Prange qui se sert de l'algorithme de Grover pour diviser l'exposant de la complexité par deux [3]. Il existait aussi des algorithmes quantiques pour la résolution du subset sum ayant une meilleure complexité que les algorithmes classiques connus [4].

Les deux tableaux ci-dessous résument la complexité des algorithmes déjà existants en donnant la valeur asymptotique de $\max_{0\leq R\leq 1} \alpha(R)$ à 10^{-7} près.

Algorithme (année)	$t = d_{GV}$	$t = \frac{d_{GV}}{2}$
Prange (1962)	0.1207019 ($R = 0.4537$)	0.0575129 ($R = 0.4689$)
Dumer (1991)	0.1162451 ($R = 0.4470$)	0.0555714 ($R = 0.4668$)
Shamir-Schroeppel	0.1162451 ($R = 0.4470$)	0.0555714 ($R = 0.4668$)
MMT (2011)	0.1114837 ($R = 0.4397$)	0.0536291 ($R = 0.4661$)
BJMM (2012)	0.1018866 ($R = 0.4275$)	0.0493326 ($R = 0.4548$)
$MMT^\sharp$	0.1052837 pour ($R = 0.4271$)	0.0504501 ($R = 0.4601$)
BJMM+MO (2015)	0.0966448 ($R = 0.4174$)	0.0472414 ($R = 0.4577$)
Prange+Grover (2009)	0.0603509 ($R = 0.4537$)	0.0287564 ($R = 0.4689$)

TABLE 1: Algorithmes ISD classiques et quantique

Nouveautés Les nouveaux résultats (la valeur de $\max_{0\leq R\leq 1} \alpha(R)$ à 10^{-7} près pour les nouveaux algorithmes) sont résumés sous forme de tableaux comme dans la section précédente.

Algorithme (année)	$t = d_{GV}$	$t = \frac{d_{GV}}{2}$
Dumer+Grover	0.0603227 ($R = 0.4654$)	0.0287563 ($R = 0.4689$)
Shamir-Schroeppel+Grover	0.0603509 (0.4537)	0.0287563 ($R = 0.4689$)
Shamir-Schroeppel+QuantumWalk	0.0596979 (0.4537)	0.0285339 ($R = 0.4689$)
MMT+QuantumWalk	0.0590365 ($R = 0.4514$)	0.0283123 ($R = 0.4689$)
BJMM+QuantumWalk	0.0591839 (0.4537)	0.0284857 ($R = 0.4675$)
MMT$^\sharp$+QuantumWalk	0.0586953 ($R = 0.4514$)	0.0282012 ($R = 0.4682$)

TABLE 2: Nouveaux algorithmes ISD quantiques

À première vue, ces améliorations peuvent sembler minimes par rapport à celles qui existent dans le cas classique. Néanmoins, en regardant les dates de publication des algorithmes classiques, on se rend compte qu'il a fallu du temps pour obtenir des améliorations considérables et que jusqu'à très récemment, le meilleur algorithme était celui de Dumer. À cela se rajoutent quelques difficultés intrinsèques aux algorithmes quantiques : les nuances de l'imbrication d'algorithmes quantiques, mais aussi des conditions très spécifiques sur les paramètres qui doivent être réunies. Cela fait que malgré les apparences, les améliorations obtenues sont non-triviales.

Par souci de concision, dans la suite de ce mémoire on présentera uniquement les résultats obtenus dans le contexte du *full-distance decoding*.

Table des matières

1 Quelques résultats mathématiques **1**
- 1.1 Théorie de l'information et des codes . . . 1
  - 1.1.1 Le coefficient binomial et la fonction d'entropie . . . 1
  - 1.1.2 Codes poinçonnés . . . 2
  - 1.1.3 Dénombrement . . . 3
  - 1.1.4 Distance de Gilbert-Varshamov . . . 4
- 1.2 Combinatoire et probabilités . . . 5
  - 1.2.1 Combinatoire . . . 5
- 1.3 Algorithmique . . . 6
  - 1.3.1 Un algorithme classique : Merge-Join . . . 6
- 1.4 Théorie des graphes . . . 7
  - 1.4.1 Graphes d-réguliers . . . 7
  - 1.4.2 Valeurs propres de la matrice d'adjacence d'un graphe . . . 7
  - 1.4.3 Graphes de Johnson . . . 8
  - 1.4.4 Produits de graphe . . . 9

2 Algorithmique quantique **12**
- 2.1 Trois mesures de complexité . . . 12
- 2.2 Algorithme de Grover . . . 13
  - 2.2.1 Analyse de complexité . . . 14
- 2.3 Marche aléatoire sur un graphe . . . 15
  - 2.3.1 Marche aléatoire classique . . . 15
  - 2.3.2 Marche aléatoire quantique (QW) . . . 15
  - 2.3.3 Quantum Walk avec structure de données . . . 17
- 2.4 Applications . . . 17
  - 2.4.1 Recherche de collisions (QW) . . . 17
  - 2.4.2 Recherche de collisions (Grover) . . . 18
- 2.5 Considérations plus poussées . . . 19
  - 2.5.1 Quantum Walk sur $J(N, r)$ VS Grover . . . 19
  - 2.5.2 Imbrication d'algorithmes . . . 20

3 Décodage par ensemble d'information (ISD) **22**
- 3.1 Algorithme de Prange . . . 22
  - 3.1.1 Version classique . . . 22
  - 3.1.2 Version quantique avec Grover . . . 23
- 3.2 Squelette d'un algorithme ISD . . . 24
- 3.3 Décodage par ensemble d'information quantique . . . 26
- 3.4 Algorithme de Lee-Brickell . . . 27
  - 3.4.1 Version classique . . . 27
- 3.5 Algorithme de Dumer/Stern . . . 29
  - 3.5.1 Version classique . . . 29
  - 3.5.2 Version quantique avec Grover . . . 34

4 ISD avancé : technique des représentations **37**
- 4.1 Algorithme de Shamir-Schroeppel . . . 38
  - 4.1.1 Version classique . . . 38
  - 4.1.2 Version quantique avec Grover . . . 42
  - 4.1.3 Les limites de l'algorithme de Grover . . . 44

4.2 Algorithme de May, Meurer, Thomae classique 45
4.2.1 Représentations VS collision 50
4.3 ISD avancé avec Quantum Walk 50
4.3.1 Squelette d'un algorithme ISD avancé avec Quantum Walk 51
4.3.2 Algorithme de Shamir-Schroeppel quantique avec Quantum Walk 55
4.3.3 Algorithme de May, Meurer et Thomae quantique avec Quantum Walk 57

5 ISD avancé 2 : technique des représentations étendue 59
5.1 Algorithme MMT^* classique 59
5.2 Algorithme de Becker, Joux, May et Meurer classique 61
5.3 Algorithme de Becker, Joux, May et Meurer quantique avec Quantum Walk 66
5.3.1 Prélude : représentations VS collision 66
5.3.2 Description de l'algorithme 69
5.3.3 Algorithme $BJMMQW^*$ 74
5.3.4 Explication du résultat 74
5.4 Retour aux sources : algorithme $MMT^\sharp$ 75
5.4.1 Version classique 75
5.4.2 Version quantique avec Quantum Walk 76

6 Algorithme de May et Ozerov 77
6.1 Version classique 77
6.1.1 Algorithme de résolution du problème Nearest Neighbour 77
6.1.2 Application au décodage : $BJMM + MO$ 89
6.2 Version quantique 91
6.2.1 Avec l'algorithme de Grover 91
6.2.2 Avec Quantum Walk ? 94

7 Développements limités pour les formules de complexité 95

8 Conclusion 97

Références 98

A Le formalisme quantique 100
A.1 Environnement mathématique 100
A.2 Mécanique et information quantique : états, observables, mesure, évolution, systèmes composés 101
A.3 Algorithmique quantique 103
A.3.1 Circuits quantiques 103
A.3.2 Quelques portes quantiques élémentaires 103
A.4 Quelques algorithmes quantiques 105

B Preuves des développements limités 107
B.0.1 Préliminaires et résultats communs 107
B.0.2 Algorithmes sans technique des représentations 108
B.0.3 Algorithme MMT 111
B.0.4 Algorithme MMTQW 115
B.0.5 Algorithme MMT^* 117

B.0.6 Algorithme $BJMMQW^*$ 118
B.0.7 Tableau récapitulatif des valeurs de c et s 120

1 Quelques résultats mathématiques

Pour les preuves, voir appendice B.

1.1 Théorie de l'information et des codes

1.1.1 Le coefficient binomial et la fonction d'entropie

THÉORÈME 1. *Si H désigne la fonction d'entropie de Shannon, on a :* $\frac{2^{nH(\frac{t}{n})}}{n+1} \leq \binom{n}{t} \leq 2^{nH(\frac{t}{n})}$
Nous écrirons pour cela $\binom{n}{t} \approx 2^{nH(\frac{t}{n})}$: en effet, cela est vrai à un facteur polynomial près.

Démonstration. On a :

$$\sum_{i=0}^{n} \binom{n}{i} (\frac{i}{n})^i (1-\frac{i}{n})^{n-i} = (\frac{i}{n} + (1-\frac{i}{n}))^n = 1$$

Et

$$\begin{aligned}(\frac{t}{n})^t (1-\frac{t}{n})^{n-t} &= e^{n(\frac{t}{n}\ln(\frac{t}{n}) + (1-\frac{t}{n})\ln(1-\frac{t}{n}))} \\ &= e^{ln(2)n(\frac{t}{n}\log_2(\frac{t}{n}) + (1-\frac{t}{n})\log_2(1-\frac{t}{n}))} \\ &= 2^{-nH(\frac{t}{n})}\end{aligned}$$

On en dérive la borne supérieure :

$$\begin{aligned}\binom{n}{t} 2^{-nH(\frac{t}{n})} &= \binom{n}{t} (\frac{t}{n})^t (1-\frac{t}{n})^{n-t} \\ &\leq \sum_{i=0}^{n} \binom{n}{i} (\frac{t}{n})^i (1-\frac{t}{n})^{n-i} \\ &= 1\end{aligned}$$

$$\Rightarrow \binom{n}{t} \leq 2^{nH(\frac{t}{n})}$$

Ainsi que la borne inférieure :

$$\begin{aligned}(n+1)\binom{n}{t} 2^{-nH(\frac{t}{n})} &= (n+1)\binom{n}{t} (\frac{t}{n})^t (1-\frac{t}{n})^{n-t} \\ &\geq \sum_{i=0}^{n} \binom{n}{i} (\frac{t}{n})^i (1-\frac{t}{n})^{n-i} \\ &= 1\end{aligned}$$

$$\Rightarrow \binom{n}{t} \geq \frac{2^{nH(\frac{t}{n})}}{(n+1)}$$

□

COROLLAIRE 1. $\binom{an}{bt} \approx \binom{\frac{a}{b}n}{bt}^b \approx \binom{n}{\frac{b}{a}t}^a$

Démonstration. Premièrement :

$$\begin{aligned}\binom{an}{bt} &\approx e^{anH(\frac{bt}{an})} \\ &\approx e^{b\frac{a}{b}nH(\frac{t}{\frac{a}{b}n})} \\ &\approx \binom{\frac{a}{b}n}{t}^b\end{aligned}$$

Deuxièmement :

$$\begin{aligned}\binom{an}{bt} &\approx e^{anH(\frac{bt}{an})} \\ &\approx e^{anH(\frac{\frac{b}{a}t}{n})} \\ &\approx \binom{n}{\frac{b}{a}t}^a\end{aligned}$$

□

1.1.2 Codes poinçonnés

DÉFINITION 1. *Étant donné un code C de matrice génératrice G et de paramètres $[n, k]$, le poinçonnage consiste à supprimer des colonnes de la matrice génératrice.*

Soit I un ensemble d'indice, $|I| = i$. On note C_I le code poinçonné dont la matrice génératrice G_I est la matrice de G dont on a enlevé les colonnes indicées par I. Alors C_I est de paramètres $[n - i, k]$. On notera H_I la matrice de parité de C_I.

THÉORÈME 2. *Soit m un message de longueur n et $m_I := m_{|I}$. $|I| = i$. Soit $c_I = m_I G_I$. Alors s'il y a une sous-matrice carrée inversible de G_I, alors il est possible de dériver $c := mG$ de manière unique à partir de c_I.*

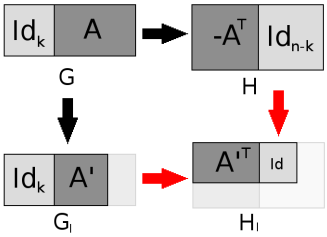
Démonstration. On supposera pour simplifier que l'on a mis G_I est sous forme systématique.
On suppose aussi que $i = 1$ et que $I = \{n\}$.
Alors G s'écrit (à des permutations de colonnes et d'opérations de pivot de Gauss près) $(G_I|g_n)$, plus précisément :

$$\begin{bmatrix} 1 & .. & 0 & | & .. & .. & .. & |g_{1,n} \\ .. & .. & .. & | & .. & .. & .. & | \ .. \\ 0 & .. & 1 & | & .. & .. & .. & |g_{n,k} \end{bmatrix}$$

Par conséquent si l'on écrit $m = (m_1, ..., m_n)$, $m_I = (m_1, ..., m_{n-1})$, $c_I = (c_{I_1}, ..., c_{I_k})$, c s'écrit $c = (c_{I_1} + g_{n,1}m_n, ..., c_{I_k} + g_{n,k}m_n)$.
Il suffit donc de connaître m_n et la colonne g_n de la matrice pour pouvoir calculer c à partir de c_I.
Ce résultat se généralise facilement au cas où $i > 2$, il faudrait alors connaître les $(m_j)_{j\in I}$ et les colonnes $(g_j)_{j\in I}$. □

THÉORÈME 3. *Soit le code C ayant pour matrice génératrice $G = [Id|A]$ et donc pour matrice de parité $H = [-A^T|Id]$ et soit C_I le code obtenu en poinçonnant les $i < k$ dernières de G, alors une matrice de parité H_I de C_I est donnée par la sous-matrice $(n-i) \times (n-k-i)$ "en haut à gauche" de H.*

Illustration de ce théorème :



Démonstration. On peut écrire $G = [Id_k|A] = [Id_k|A_1|A_2]$ où $A_1 \in \mathcal{M}_{k,n-k-i}(\mathbb{F}_2)$ et $A_2 \in \mathcal{M}_{k,i}(\mathbb{F}_2)$ correspond à la partie qui sera poinçonnée.
Alors, d'un côté $H = [-A^T|Id_{n-k}]$ avec $A^T = \begin{bmatrix} A_1^T \\ A_2^T \end{bmatrix}$, donc au final, H s'écrit de manière plus détaillée :

$$\begin{bmatrix} -A_1^T & Id_{n-k-i} & 0_{n-k-i,n-i} \\ A_2^T & 0_{i,n-k-i} & Id_i \end{bmatrix}$$

D'un autre côté, $G_I = [Id_k|A_1] \in \mathcal{M}_{k,n-i}(\mathbb{F}_2)$ donc $H_I = [A_1^T|Id_{n-k-i}]$, ce qui correspond à la sous-matrice de taille $(n-i) \times (n-k-i)$ "en haut à gauche" de H. □

1.1.3 Dénombrement

THÉORÈME 4. *Soit C un code linéaire de paramètres $[n,k]$. Alors pour un syndrome $s \neq 0$ donné, le nombre moyen de mots $e \notin C$ de poids p ayant pour syndrome s est $\frac{\binom{n}{p}}{2^{n-k}}$.*

Démonstration. Il y a clairement $\binom{n}{p}$ mots de longueur n et de poids p. On peut donc les numéroter et on définit pour le $i^{\text{ème}}$ mot e_i la variable aléatoire suivante :

$$X_i = \begin{cases} 1 & \text{si } He_i^T = s, \\ 0 & \text{sinon.} \end{cases}$$

On a alors pour tout i :

$$\begin{aligned} \mathbb{P}(X_i = 1) &= \mathbb{P}(He_i^T = s) \\ &= \mathbb{P}(\bigwedge_{j=1}^{n-k} (h_{k_0,j} e_{i,k_0} = s_j - \sum_{k=1,k\neq k_0}^{n} h_{k,j} e_{k,1})) \text{ où } e_{i,k_0} = 1 \end{aligned}$$

Le terme à droite est soit 1 ou 0, dans tous les cas la probabilité que $h_{k_0,j}$ y soit égal est $\frac{1}{2}$.

Par conséquent :

$$\begin{aligned}\mathbb{P}(X_i = 1) &= \mathbb{P}(\bigwedge_{j=1}^{n-k} (H_{k_0,j} e_{i,k_0} = s_j - \sum_{k=1,k\neq k_0}^{n} H_{k,j} e_{k,1}) \\ &= \prod_{j=1}^{n-k} \mathbb{P}((H_{k_0,j} e_{i,k_0} = s_j - \sum_{k=1,k\neq k_0}^{n} H_{k,j} e_{k,1})) \\ &= \frac{1}{2^{n-k}}\end{aligned}$$

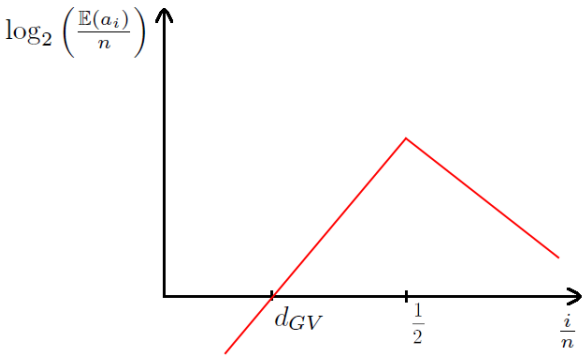
Alors $X := \sum_i X_i$ est la variable aléatoire qui donne le nombre de mots de poids p ayant pour syndrome s. On calcule :

$\mathbb{E}(X) = \sum_i \mathbb{E}(X_i) = \sum_i \frac{1}{2^{n-k}} = \frac{\binom{n}{p}}{2^{n-k}}$. □

1.1.4 Distance de Gilbert-Varshamov

THÉORÈME 5. *Soit C un code linéaire de paramètres $[n, k]$. Soit a_i le nombre moyen de mots de poids i dans C. Alors :*

1. $\mathbb{E}(a_i) = 1$ *ssi* $H\left(\frac{i}{n}\right) = 1 - R \Leftrightarrow \frac{i}{n} = H^{-1}(1 - R)$. *On appelle cette valeur de i distance de Gilbert-Varshamov et on note cette fraction* $d_{GV} := H^{-1}(1 - R)$.
2. *Plus généralement, le graphique suivant montre l'évolution du rapport* $\log_2\left(\frac{\mathbb{E}(a_i)}{n}\right)$ *en fonction de* $\frac{i}{n}$ *:*



Démonstration. On a $a_i = \sum_{x, w_H(x)=i} \mathbb{1}_{x\in C}$.

Par conséquent :

$$\begin{aligned}
\mathbb{E}(a_i) &= \sum_{x, w_H(x)=i} \mathbb{E}\left(\mathbb{1}_{x\in C}\right) \\
&= \sum_{x, w_H(x)=i} \mathbb{P}\left(x \in C\right) \\
&= \sum_{x, w_H(x)=i} \frac{1}{2^{n-k}} \\
&= \frac{\binom{n}{i}}{2^{n-k}} \\
&\approx 2^{nH\left(\frac{i}{n}\right)-(n-k)} \\
&= 2^{n\left(H\left(\frac{i}{n}\right)-1+R\right)}
\end{aligned}$$

On pose $f(x) = H(x) - 1 + R$, on a alors $f(\frac{i}{n}) = \log_2\left(\frac{\mathbb{E}(a_i)}{n}\right)$ et :

1. $f(x) = 0 \Leftrightarrow x = H^{-1}(1-R)$ donc $\mathbb{E}(a_i) = 1$ ssi $\frac{i}{n} = d_{GV}$
2. $\max_{x\in\{0,1\}} f(x) = R$ pour l'argument $x = \frac{1}{2}$.

□

1.2 Combinatoire et probabilités

1.2.1 Combinatoire

THÉORÈME 6. *Soient S un ensemble, M un ensemble de données, $f : S \to M$ une fonction aléatoire ou pseudo-aléatoire. On considère $m \in M$, $P_m : S \mapsto \{0,1\}$ tel que $P_m(s) = 1$ si $f(s) = m$ et $P_m(s) = 0$ sinon. Alors le nombre moyen d'éléments s de S tels que $P_m(s) = 1$ est $\frac{|S|}{|M|}$.*

Démonstration. On numérote les éléments de S et on définit pour s_i, le $i^{\text{ème}}$ éléments, la variable aléatoire suivante :

$X_i = P_m(s_i)$
Comme f est aléatoire ou pseudo-aléatoire, on a $\mathbb{P}(X_i = 1) = M^{-1}$ pour tout i. La variable aléatoire $X := \sum_i X_i$ donne alors le nombre d'éléments de S tels que $P_m(s) = 1$. On calcule $\mathbb{E}(X)$:

$$\mathbb{E}(X) = \sum_i \mathbb{E}(X_i) = |S|\mathbb{P}(X_i = 1) = \frac{|S|}{|M|}$$

□

THÉORÈME 7. *Soient S_1 et S_2 des ensembles, M un ensemble de données, $f_1 : S_1 \to M$ et $f_2 : S_2 \to M$ des fonctions aléatoires ou pseudo-aléatoires. Alors la probabilité de collision des valeurs de f_1 et de f_2 est $\frac{|S_1||S_2|}{|M|}$.*

Démonstration. On considère le produit cartésien $S = S_1 \times S_2$ et on numérote ses $|S_1||S_2|$ éléments. On définit pour son $i^{\text{ème}}$ élément $(x, y)_i$ la variable aléatoire suivante :

$X_i = 1$ si $f_1(x) = f_2(y)$, $X_i = 0$ sinon. Comme f_1 et f_2 sont aléatoires ou pseudo-aléatoires, on a $\mathbb{P}(X_i = 1) = M^{-1}$ pour tout i. La variable aléatoire $X := \sum_i X_i$ donne alors le nombre d'éléments (x, y) de S tels que $f_1(x) = f_2(y)$, i.e. le nombre de collisions. On calcule $\mathbb{E}(X)$:

$$\mathbb{E}(X) = \sum_i \mathbb{E}(X_i) = |S_1||S_2|\mathbb{P}(X_i = 1) = \frac{|S_1||S_2|}{|M|}$$

□

THÉORÈME 8. *Soient $X \sim \mathrm{B}(p)$,$X \sim \mathrm{B}(q)$ des variables aléatoires. Alors $X + Y \sim \mathrm{B}(p + q - 2pq)$.*
En particulier, si $p = q$, $X + Y \sim \mathrm{B}(2p(1 - p))$.

Démonstration.

$$\begin{aligned}\mathbb{P}(X \oplus Y = 1) &= \mathbb{P}(X = 0, Y = 1) + \mathbb{P}(X = 1, Y = 0)\\ &= (1 - p)q + (1 - q)p\\ &= p + q - 2pq\end{aligned}$$

□

1.3 Algorithmique

1.3.1 Un algorithme classique : Merge-Join

Soient E_1, E_2, E des ensembles quelconques, $|E_i| \leq M \in \mathbb{N}$ et $f_i : E_i \to E$ des fonctions pour $i = 1, 2$. On trouve dans [1] un algorithme, appelé $Merge - Join$, qui permet de résoudre le problème suivant :

PROBLÈME 3 ($Merge - Join$). *Étant donné deux listes $L_1 = ((x_i, f_1(x_i)))_{i \in I_1}$ et $L_2 = ((y_i, f_2(y_i)))_{i \in I_2}$, donner la liste L des éléments $(x_i, x_j) \in L_1 \times L_2$ qui vérifient $f_1(x_i) = f_2(x_j)$ et éventuellement d'autres conditions.*

En pratique, les E_i seront des ensembles de vecteurs binaires de même longueur n, donc $M = 2^n$.

Cet algorithme renvoie une liste dont les éléments sont obtenus en joignant ceux des listes d'entrée et en gardant ceux qui vérifient certaines conditions, tout en prenant soin d'éviter les doublons.

Algorithme 1 : $Merge - Join$

Entrées : $L_1 = ((x_i, f_1(x_i)))_{i \in I_1}$ et $L_2 = ((y_i, f_2(y_i)))_{i \in I_2}$
Sorties : Une liste $L := \{((x_i, y_j), F(f_1(x_i), f_2(y_j))) \mid f_1(x_i) = f_2(y_j)\}$

```
Trier L_1 et L_2 dans l'ordre croissant ou lexicographique en fonction de leurs premiers entrées
L ← Nouvelle liste
i ← 0, j ← |L_2| − 1
tant que i > |L_1| et j ≥ 0 faire
    si f_1(x_i) < f_2(y_j) alors
        i ← i + 1
    si f_1(x_i) > f_2(y_j) alors
        j ← j − 1
    si f_1(x_i) = f_2(y_j) alors
        L ← L ∪ ((x_i, y_j), F(f_1(x_i), f_2(y_j)))
```

La complexité de l'étape de tri est $O(|L_1|\log(|L_1|)+|L_2|\log(|L_2|)) = \tilde{O}(|L_1|+|L_2|)$. Le nombre attendu de tours de boucles est égal au nombre de collisions, donc il y en a $O(\frac{|L_1||L_2|}{M})$.

On obtient une complexité en $\tilde{O}(|L_1|+|L_2|+\frac{|L_1||L_2|}{M}) = \tilde{O}(\max(|L_1|, |L_2|, \frac{|L_1||L_2|}{M}))$ en temps et $\tilde{O}(|L_1| + |L_2| + |L|) = \tilde{O}(\max(|L_1|, |L_2|, |L|))$ en espace.

Remarque : En théorie, $|L| = \frac{|L_1||L_2|}{M}$ et on peut écrire $\tilde{O}(\max(|L_1|, |L_2|, \frac{|L_1||L_2|}{M}))$ pour la complexité en espace aussi. En pratique, on peut exiger d'autres conditions que l'égalité $f_1(x_i) = f_2(y_j)$ pour garder (x_i, y_j) dans L, donc il peut y avoir moins d'éléments dans la liste L qu'il y avait de collisions.

Notation : On utilisera la notation $L_1 \bowtie L_2$ pour indiquer un $Merge-Join$. À chaque utilisation, on ajoutera néanmoins quelques autres symboles en-dessous et au-dessus du papillon $\bowtie$ qui résument les spécificités du cas en question.

Remarque : Cet algorithme reste valable avec quelques petites modifications pour toute structure de données ordonnée.

1.4 Théorie des graphes

1.4.1 Graphes d-réguliers

Définition 2. *un graphe $G = (E, V)$ est dit d-régulier lorsque tous les sommets ont le même degré, i.e. le même nombre de voisins.*

1.4.2 Valeurs propres de la matrice d'adjacence d'un graphe

Soit $G = (E, V)$ un graphe à N sommets et M arêtes. et on considère sa matrice d'adjacence, modifiée de la manière suivante : on divise chaque ligne par le nombre d'entrées non-nulles (par conséquent, par le degré sortant du sommet en question) pour la transformer en une matrice stochastique. On obtient ainsi une matrice de transition d'une chaîne de Markov.
En effet, rappelons que pour définir une chaîne de Markov, il faut une distribution

de probabilité initiale v sur un ensemble (de taille n) ainsi qu'une matrice carrée de transition P contenant les probabilités conditionnelles de transition :

$$P = \begin{bmatrix} P(1|1) & ... & P(1|n) \\ ... & ... & ... \\ P(n|1) & ... & P(n|n) \end{bmatrix}$$

On a $P_{t+1}(j) = \sum_i P(j|i)P_t(i)$
Cette matrice d'adjacence a une valeur propre 1 correspondant au vecteur uniforme $u = (\frac{1}{N}, ..., \frac{1}{N})$.

Si le graphe est connexe, il y a un unique vecteur propre associé à la valeur propre 1.

Trou spectral Soit $G = (E, V)$ un graphe d-régulier, non-dirigé et connexe à N sommets et P la chaîne de Markov donnée par sa matrice d'adjacence. Grâce à cela nous pouvons effectuer une marche aléatoire sur le graphe.
Nous savons qu'il y a un vecteur propre $v_1 = u$ de valeur propre $\lambda_1 = 1$ et tous les $N - 1$ autres vecteurs propres v_i auront pour valeur propre $\lambda_i \in]-1, 1[$.
On définit le trou spectral δ du graphe/de la chaîne : $\delta := 1 - \max_{i\geq 2} |\lambda_i|$.
On peut écrire $v = \sum_{i=1}^{n} \alpha_i v_i$ où v est la distribution de probabilité initiale.

$$\begin{aligned} ||P^k v - u||^2 &= ||\sum_i \alpha_i P^k v_i - u||^2 \\ &= ||\sum_{i\geq 2} \alpha_i \lambda_i^k v_i + u - u||^2 \\ &\leq \sum_{i\geq 2} \alpha_i^2 \lambda_i^{2k} ||v_i||^2 \\ &\leq (1-\delta)^{2k} \sum_i \alpha_i^2 \\ &\leq (1-\delta)^{2k} ||v||^2 \\ &\leq (1-\delta)^{2k} \end{aligned}$$

Par conséquent, en prenant $k = \frac{\ln(\frac{1}{\eta})}{\ln(\frac{1}{\delta})}$ on aura $||P^k v - u|| \leq (1-\delta)^{\frac{\ln(\eta)}{\ln(\frac{1}{\delta})}} \leq \eta$.
Or lorsque δ est petit, $\ln(\frac{1}{\delta}) \approx \delta$. Par conséquent, pour avoir $P^k v$ proche de u il faut $k = O(1/\delta)$ étapes de la marche aléatoire.
En général, il est rare que $\delta \approx 1$. Mais pour un graphe d-régulier aléatoire, cela arrive très souvent : dès que le degré du graphe est supérieur à 2, lorsque $d \to n$, $\delta \to 1$.

1.4.3 Graphes de Johnson

On a vu dans la section précédente que pour connaître le nombre d'étapes de marche aléatoire à faire pour arriver à une distribution uniforme, il faut connaître le trou spectral du graphe/de la chaîne de Markov.

Or il y a peu de graphes pour lesquels on peut connaître facilement les valeurs propres et donc le trou spectral. Quelques exemples seraient les graphes de Johnson, qui nous intéresseront particulièrement, et les n-cubes ou hypercubes.

Définition 3. *Un graphe de Johnson $J(n,r)$ est un graphe non-dirigé dont les sommets sont les sous-ensembles de taille r d'un ensemble de taille n, avec une arête entre deux sommets S et S' ssi $|S \cap S'| = r-1$, autrement dit si pour obtenir S' à partir de S il suffit de supprimer un élément et ajouter un nouvel élément à sa place.*

Les graphes de Johnson ont les propriétés suivantes :

1. Le nombre de sommets est clairement $\binom{n}{r}$.
2. Pour chaque sous-ensemble S de taille r, il y a r éléments qui peuvent être supprimés et chacun peut être remplacé de $(n-r)$ manières. Par conséquent il y a $d = r(n-r)$ sous-ensembles de taille r qui diffèrent d'un élément par rapport à S. Par conséquent $J(n,r)$ est d-régulier.
3. Les graphes de Johnson sont un cas particulier des schémas de Johnson. Sans rentrer dans les détails, il est possible de calculer les valeurs propres d'un schéma de Johnson à l'aide de polynômes d'Eberlein. Pour un graphe de Johnson, ces valeurs propres sont :
   $d\lambda_i = (r-i)(n-r-i) - i = rn - r^2 + i^2 - in - i$ pour $(0 \leq i \leq n-1)$
   Ainsi $\lambda_0 = 1$ et

$$\begin{aligned} d \max_{i \geq 1} |\lambda_i| &= \max_{i \geq 1} |r(n-r) + i(i-n-1)| \\ &= \max_{i \geq 1} |d - i(n-i+1)| \\ &= d - \min_{i \geq 1} |i(n-i+1)| \\ &= d - n \end{aligned}$$

   Par conséquent $\delta = 1 - \max_{i \geq 1} |\lambda_i| = \frac{n}{d} = \frac{n}{r(n-r)}$.

1.4.4 Produits de graphe

Définition 4 (Produit cartésien de graphes). *Étant donné deux graphes $G_1 = (E_1, V_1)$ et $G_2 = (E_2, V_2)$, on définit leur produit cartésien $G_1 \times G_2 = G = (E, V)$:*

1. $V = V_1 \times V_2 \Leftrightarrow V = \{v_1 v_2 \mid v_1 \in V_1, v_2 \in V_2\}$
2. $E = \{(u_1 u_2, v_1 v_2) \mid (u_1 = v_1 \wedge (u_2, v_2) \in V_2) \vee ((u_1, v_1) \in V_1 \wedge u_2 = v_2)\}$

On notera dorénavant $n = |V|$, $m = |E|$ et pour $i = 1, 2$, $n_i = |V_i|$, $m_i = |E_i|$. On va également supposer que G_i est d_i-régulier.

Théorème 9 (Matrice d'adjacence d'un produit cartésien de graphes). *Soit $A_i \in \mathcal{M}_{n_i, n_i}(\mathbb{Z})$ la matrice d'adjacence du graphe G_i pour $i = 1, 2$. Alors la matrice d'adjacence A du graphe $G = G_1 \times G_2$ est :*

$$A = I_{n_1} \otimes A_2 + A_1 \otimes I_{n_2}$$

Démonstration. Admis. □

Théorème 10 (Vecteurs propres et valeurs propres d'un produit cartésien de graphes). *Soient, pour $i = 1, ..., k_1$, $k_1 \leq n_1$, λ_i la valeur propre (non-normalisée) du graphe G_1 associée au vecteur propre f_i.*

Soient, pour $j = 1, ..., k_2$, $k_2 \leq n_2$, μ_j la valeur propre (non-normalisée) du graphe G_2 associée au vecteur propre g_j.

Alors le graphe $G = G_1 \times G_2$ possède $k_1 k_2$ vecteurs propres. Ces vecteurs propres sont :
$h_{i,j} = f_i \otimes g_j$ de valeur propre $\nu_{i,j} = \lambda_i + \mu_j$, pour $i = 1, ..., k_1, j = 1, ..., k_2$.

Démonstration. Il faut prouver que $Ah_{i,j} = \nu_{i,j} h_{i,j}$. Or $I_{n_1} \otimes A_2 + A_1 \otimes I_{n_2}$ d'après le théorème précédent. Ainsi :

$$\begin{aligned} Ah_{i,j} &= (I_{n_1} \otimes A_2 + A_1 \otimes I_{n_2}) (f_i \otimes g_j) \\ &= (I_{n_1} \otimes A_2) (f_i \otimes g_j) + (A_1 \otimes I_{n_2}) (f_i \otimes g_j) \\ &= I_{n_1} f_i \otimes A_2 g_j + A_1 f_i \otimes I_{n_2} g_j \\ &= f_i \otimes \mu_j g_j + \lambda_i f_i \otimes g_j \\ &= (\mu_j + \lambda_i) (f_i \otimes g_j) \\ &= \nu_{i,j} h_{i,j} \end{aligned}$$

□

Dorénavant, on supposera que les valeurs propres des graphes G_1 et G_2 sont indicés du plus grand au plus petit, c'est-à-dire :

$d_1 = \lambda_1 \geq ... \geq \lambda_{k_1}$
$d_2 = \mu_1 \geq ... \geq \mu_{k_2}$

On a alors clairement $\max_{i,j} \nu_{i,j} = \lambda_1 + \mu_1 = d_1 + d_2$.

THÉORÈME 11 (Trou spectral d'un produit cartésien de graphes). *Soient $\delta_1 = \frac{d_1 - \max_{i=2,...,k_1} |\lambda_i|}{d_1}$ le trou spectral de G_1, et $\delta_2 = \frac{d_2 - \max_{j=2,...,k_2} |\mu_j|}{d_2}$ le trou spectral de G_2.*
Alors, si on note $\delta = \frac{d_1 + d_2 - \max_{i,j} |\lambda_i + \mu_j|}{d_1 + d_2}$ le trou spectral du graphe G, on a :
$\delta \geq \frac{d_1 \delta_1 + d_2 \delta_2}{d_1 + d_2}$

Démonstration.

$$\begin{aligned} \delta &= \frac{d_1 + d_2 - \max_{i,j} |\lambda_i + \mu_j|}{d_1 + d_2} \\ &\geq \frac{d_1 + d_2 - \max_i |\lambda_i| - \max_j |\mu_j|}{d_1 + d_2} \\ &= \frac{d_1 - \max_i |\lambda_i|}{d_1 + d_2} + \frac{d_2 - \max_j |\mu_j|}{d_1 + d_2} \\ &= \frac{d_1 \delta_1 + d_2 \delta_2}{d_1 + d_2} \end{aligned}$$

□

THÉORÈME 12 (Produit cartésien de graphes de Johnson). *Soit $J(n, r) = (V, E)$, $m \in \mathbb{N}$ et $J^m(n, r) := \times_{i=1}^m J(n, r) = (V_m, E_m)$. Alors :*

1. *$J^m(n, r)$ a $\binom{n}{r}^m$ arêtes et est d_m-régulier avec $d_m = mr(n - r)$.*

2. *Si on note $\delta(J)$ et $\delta(J^m)$ les trous spectraux de $J(n,r)$ resp. $J^m(n,r)$, on a :*
$\delta(J^m) \geq \delta(J)$

Démonstration. 1. On rappelle que $J(n,r)$ est d-régulier avec $d = r(n-r)$ et on démontre le résultat par récurrence.

(a) $\underline{m = 2}$:
$J^2(n,r) := J(n,r) \times J(n,r)$.
Par définition du produit cartésien de graphes :

-$V_2 = V \times V \Rightarrow |V_2| = |V|^2 = \binom{n}{r}^2$.
- On fixe un sommet R_1R_2. Alors, pour un autre sommet S_1S_2, $(R_1R_2, S_1S_2) \in E_2$ ssi

— $R_1 = S_1$ et (R_2, S_2) est une arête du deuxième graphe de Johnson : il y en a d.
— $R_2 = S_2$ et (R_1, S_1) est une arête du premier graphe de Johnson : il y en a également d.

Par conséquent, $J^2(n,r)$ est d_2-régulier avec $d_2 = 2d$.

(b) $\underline{m \geq 2}$:
On suppose que $J^m(n,r)$ a $\binom{n}{r}^m$ arêtes et est d_m régulier.

$J^{m+1}(n,r) := J^m(n,r) \times J(n,r)$.
Alors, par définition :

-$V_{m+1} = V_m \times V \Rightarrow |V_{m+1}| = |V_m| \times |V| = \binom{n}{r}^{m+1}$.
- On fixe un sommet R_1R_2. Alors, pour un autre sommet S_1S_2, $(R_1R_2, S_1S_2) \in E_2$ ssi

— $R_1 = S_1$ et (R_2, S_2) est une arête de $J(n,r)$: il y en a d.
— $R_2 = S_2$ et (R_1, S_1) est une arête de $J^m(n,r)$: il y en a d_m par hypothèse de récurrence.

Par conséquent, $J^2(n,r)$ est $(d + d_m)$-régulier $\Leftrightarrow = d_{m+1}$-régulier.

2. Par récurrence.

(a) $\underline{m = 2}$:
On applique le théorème 11 :

$$\delta(J^2) \geq \frac{d\delta(J) + d\delta(J)}{d + d} = \delta(J)$$

(b) $\underline{m \geq 2}$:
On suppose que $\delta(J^m) \geq \delta(J)$ et on cherche à montrer que $\delta(J^{m+1}) \geq \delta(J)$.

On applique le théorème 11 et les résultats du point (1) de ce théorème :

$$\begin{aligned}\delta(J^{m+1}) &\geq \frac{d_m\delta(J^m) + d\delta(J)}{d_m + d} \\ &= \frac{md\delta(J) + d\delta(J)}{md + d} \\ &= \delta(J)\end{aligned}$$

□

2 Algorithmique quantique

Deux algorithmes quantiques nous intéresseront principalement en vue d'une application au décodage des codes linéaires : l'algorithme de Grover qui permet d'effectuer une recherche dans un ensemble non-structuré et l'algorithme de marche aléatoire quantique.

Nous allons commencer par dire quelques mots sur la différence entre l'algorithmique classique et l'algorithmique quantique. Nous savons en effet qu'il est possible de calculer toute fonction uniquement à partir de portes NANDs ($(a, b) \mapsto 1 \oplus (a \wedge b)$). Il faut tout d'abord savoir que tout ce qui est calculable avec un ordinateur classique l'est avec un ordinateur quantique, avec la nuance suivante : la porte NAND est irréversible, c'est-à-dire que l'on ne peut pas passer de la sortie à l'entrée, alors que les opérations de calcul quantique sont données par des matrices unitaires et sont donc toutes réversibles. Par conséquent, pour transformer directement un algorithme classique en un algorithme quantique, il faudrait remplacer les portes NANDs par des portes réversibles qui font la même opération. Un tel circuit existe, il s'agit de la porte de Toffoli :
$(a, b, c) \mapsto (a, b, c \oplus (a \wedge b))$
En fixant $c = 1$ le troisième registre donne le NAND des deux premiers et cette porte est clairement réversible.

Il faut savoir qu'un ordinateur quantique ne peut pas calculer ce qui est incalculable pour un ordinateur classique, par exemple le problème de l'arrêt. L'intérêt principal des algorithmes quantiques est qu'ils permettent dans certains cas de réduire la complexité (au sens de l'ordre de grandeur des opérations nécessaires) d'un calcul.

Nous référons à l'appendice A pour plus d'informations sur le formalisme quantique.

2.1 Trois mesures de complexité

De nombreux algorithmes quantiques font intervenir une fonction f à laquelle ils font appel de nombreuses fois lors de leur déroulement. Nous allons nous intéresser à la complexité en moyenne et nous allons distinguer trois types de complexité :

1. On définit **la complexité en requêtes** d'un algorithme comme étant le nombre de fois où il fait appel à la fonction f.

2. **La complexité en temps** ou **complexité temporelle** est définie comme le temps d'exécution moyen de l'algorithme. Par conséquent, elle est fonction de la complexité en requête et du temps d'exécution de la fonction f.
3. **La complexité en espace** ou **complexité spatiale** est définie comme l'espace mémoire occupée par les données internes de l'algorithme. Elle est non-cumulative contrairement à la complexité en temps car on peut souvent réutiliser de l'espace mémoire.

2.2 Algorithme de Grover

L'algorithme de Grover résout le problème suivant : étant donné une fonction $f : E \to \{0, 1\}$, trouver un argument x tel que $f(x) = 1$. La technique à la base de l'algorithme de Grover s'appelle *l'amplification de l'amplitude*. Nous allons considérer le cas où il y a plusieurs solutions au problème.

L'algorithme de Grover nécessite les ingrédients suivants :

Soit $N = 2^n$.

1. Étant donné une fonction $f : \mathbb{Z} \to \{0, 1\}$ (concrètement, une fonction indicatrice ou un oracle de vérification d'une propriété), on définit pour $|z\rangle$ de n qubits $O_f : |z\rangle \mapsto (-1)^{f(z)}|z\rangle$. Si l'on nomme "bons" les $|z\rangle$ tels que $f(z) = 1$ et "mauvais" ceux qui vérifient $f(z) = 0$, l'action de O_f consiste à laisser fixes les "mauvais" éléments et changer le signe de l'amplitude (de la phase) des "bons" éléments. O_f a le même temps de calcul que f.
   S'il y a t bonnes valeurs de z et que l'on note $|B\rangle := \frac{1}{\sqrt{t}} \sum_{z,f(z)=1} |z\rangle$ et $|M\rangle := \frac{1}{\sqrt{N-t}} \sum_{z,f(z)=0} |z\rangle$, alors on a $|B\rangle \perp |M\rangle$ et $O_f = 2|M\rangle\langle M| - Id$.
   Par conséquent, cette opération consiste en une réflexion à travers $|M\rangle$ qui laisse fixe la phase des mauvais éléments et inverse la phase des bons éléments. Ainsi, on peut encore l'appeler *inversion de la phase*.
2. On considère la porte de Hadamard $H^{\otimes n}$, qui renvoie $|0^n\rangle$ sur $\frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle$, une superposition uniforme de tous les élements de la base. Par conséquent, s'il y a t bonnes valeurs de z la probabilité de tomber sur un bon z en mesurant devient $\frac{t}{N}$.

   On note $|U\rangle := H^{\otimes n}|0^n\rangle$ et $R := 2|0^n\rangle\langle 0^n| - Id$.
   On a alors :
   $H^{\otimes n} R \left(H^{\otimes n}\right)^* = 2H^{\otimes n}|0^n\rangle\langle 0^n| \left(H^{\otimes n}\right)^* - H^{\otimes n} \left(H^{\otimes n}\right)^* = 2|U\rangle\langle U| - Id$.
   Cette opération est par conséquent une réflexion par $|U\rangle$, on peut la nommer *inversion par rapport à la moyenne*.

   Nous notons enfin qu'il est possible d'appliquer, à la place de $H^{\otimes n}$, n'importe quel algorithme A qui, appliqué à $|0^n\rangle$, rend un état superposé où en mesurant on obtiendra un bon z avec probabilité p.

L'algorithme de Grover se déroule ainsi, avec $\varepsilon := \frac{t}{N}$:

Algorithme 2 : *Grover*

Entrées : ε, f
Sorties : une superposition $|Z\rangle$ des états $|z\rangle$ tel que $f(z) = 1$

1 Préparer l'état de départ $|U\rangle$
2 **pour** $\frac{1}{\sqrt{\varepsilon}}$ *itérations* **faire**
3 Appliquer O_f à l'état courant (inversion de la phase)
4 Appliquer $H^{\otimes n} R H^{\otimes n^T}$ à l'état courant (inversion par rapport à la moyenne)

On a à la sortie de l'algorithme de Grover une superposition $|Z\rangle$ de bons états. Nous pouvons ensuite soit mesurer le registre contenant $|Z\rangle$ afin d'obtenir la valeur d'un bon état $|z\rangle$, ou bien le laisser tel quel, par exemple pour servir d'entrée à un autre algorithme.

Si nous ne connaissons pas t en avance, on ne saura pas combien de fois il faut boucler dans l'algorithme. Il est possible de pallier à ce problème, sans grande perte de temps d'exécution, en devinant des valeurs de k d'une certaine façon.

2.2.1 Analyse de complexité

On a tout d'abord l'égalité suivante :

$$\begin{aligned}
&\sin(\arcsin(\sqrt{\varepsilon}))|B\rangle + \cos(\arcsin(\sqrt{\varepsilon}))|M\rangle \\
&= \sqrt{\varepsilon}|B\rangle + \sqrt{1-\varepsilon}|M\rangle \\
&= \sqrt{\frac{t}{N}}\frac{1}{\sqrt{t}} \sum_{z,\chi(z)=1} |z\rangle + \sqrt{\frac{N-t}{N}}\frac{1}{\sqrt{N-t}} \sum_{z,\chi(z)=0} |z\rangle \\
&= \frac{1}{\sqrt{N}} \sum_{z=0}^{N-1} |z\rangle \\
&= |U\rangle
\end{aligned}$$

Si l'on note $\theta = \arcsin(\sqrt{\varepsilon})$
On peut voir géométriquement qu'un tour de la boucle dans l'algorithme de Grover augmente l'angle de 2θ degrés. Par conséquent, après k tours de boucle, l'état devient : $\sin((2k+1)\theta)|B\rangle + \cos((2k+1)\theta)|M\rangle$.
La probabilité d'observer les bons états, i.e. $|B\rangle$ est alors de $\sin((2k+1)\theta)$. Pour que cette probabilité soit proche de 1 il faut :

$$(2k+1)\theta \approx \frac{\pi}{2} \Rightarrow k \approx \frac{\pi}{4\theta} - \frac{1}{2} \Rightarrow k = O(\frac{1}{\theta}) = O(\frac{1}{\sqrt{\varepsilon}}) = O(\sqrt{\frac{N}{t}})$$

L'algorithme de Grover a par conséquent une complexité en requêtes en $O(\frac{1}{\sqrt{\varepsilon}})$.
De plus, il a été montré que ce type de recherche nécessite au minimum $O(\sqrt{N})$ étapes. L'algorithme de Grover est par conséquent optimal.

En conclusion, puisque l'on fait une seule requête à la fonction f à chaque itération de la boucle, si l'on désigne par T_f le temps d'exécution de la fonction f, l'algorithme de Grover a une complexité temporelle en $O(\frac{1}{\sqrt{\varepsilon}} T_f)$.

2.3 Marche aléatoire sur un graphe

2.3.1 Marche aléatoire classique

On se donne un graphe $G = (E, V)$ non-dirigé et d-régulier et un ensemble $M \subset V$. On dit que les éléments de M sont marqués. On pose $N := |V|$ et $\varepsilon = \frac{|M|}{N}$.
Pour trouver un élément marqué, on peut faire une marche aléatoire : partant d'un sommet y, tester si y est marqué. S'il l'est, c'est fini, sinon choisir un de ses voisins comme sommet courant.
On peut implémenter une marche aléatoire classique à l'aide d'une chaîne de Markov. On utilise alors la procédure suivante pour trouver un sommet marqué :

Algorithme 3 : *RandomWalk*

Entrées : $G = (E, V)$ et $M \subset V$
Sorties : un e tel que $e \in M$

```
SETUP : Préparer un état initial v (coût S)
pour 1/ε itérations faire
    CHECK : si Le sommet courant est marqué (coût C) alors
        retourner le sommet courant
    sinon
        répéter 1/δ fois
            UPDATE : Faire une étape de la marche aléatoire (coût U)
        jusqu'à arriver à une distribution uniforme
```

Par conséquent la complexité totale de l'algorithme, en fonction de la complexité de ses étapes élémentaires, est :

$$\boxed{S + \frac{1}{\varepsilon}(C + \frac{1}{\delta}U)}$$

Ici comme dans la version quantique, le but sera de faire en sorte que le terme dominant entre parenthèses soit $\frac{1}{\delta}U$. Nous aurons alors dans le cas classique un algorithme en $O(\frac{1}{\delta\varepsilon})$.

2.3.2 Marche aléatoire quantique (QW)

La marche aléatoire quantique (Quantum Walk - QW) est une généralisation de l'algorithme de Grover, il est en effet possible de redériver l'algorithme de Grover à partir de la marche aléatoire quantique [6].

Dans la version quantique de la marche aléatoire, il y a toujours une chaîne de Markov ayant une matrice de transition P, mais les états du système ne sont plus les sommets mais les arêtes que l'on représentera par le produit tensoriel $|x\rangle|y\rangle$ où x est le sommet courant et y le sommet précédent.

On dit qu'une arête $|x\rangle|y\rangle$ est "bonne" lorsque $x \in M$ et "mauvaise" lorsque $x \notin M$. On définit aussi $|p_x\rangle = \sum_y \sqrt{P(y|x)}|y\rangle$.
Comme pour l'algorithme de Grover on définit la superposition uniforme des bonnes et mauvaises arêtes :
$|G\rangle = \frac{1}{\sqrt{M}} \sum_{x \in M} |x\rangle|p_x\rangle$

$|B\rangle = \frac{1}{N-\sqrt{M}} \sum_{x \notin M} |x\rangle|p_x\rangle$
Ainsi que la superposition uniforme sur tous les états :
$|U\rangle := \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle|p_i\rangle = \cos(\theta)|G\rangle + \sin(\theta)|B\rangle$ avec $\theta = \arcsin(\sqrt{\varepsilon})$.

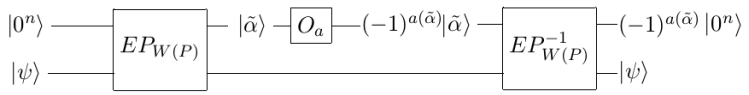
Les étapes de l'algorithme de marche aléatoire quantique sont les mêmes que pour l'algorithme de Grover, à savoir : préparer $|U\rangle$, faire une boucle où l'on reflète par $|B\rangle$ et ensuite par $|U\rangle$ et mesurer le premier registre à la fin de la boucle. Il y a, pour les mêmes raisons que dans l'algorithme de Grover, $P(\frac{1}{\sqrt{\varepsilon}})$ tours de boucles.
La nouveauté est la réflexion par $|U\rangle$ dont nous allons expliquer l'implémentation.

1. Il faut d'abord dériver une matrice unitaire $W(P)$ à partir de la matrice stochastique P. Pour ce faire, on définit $\mathbb{A} := \text{span}\{|x\rangle|p_x\rangle\}$ et $\mathbb{B} := \text{span}\{|p_y\rangle|y\rangle\}$. Soient $R_\mathbb{A}$ et $R_\mathbb{B}$ les réflexions par $\mathbb{A}$ et $\mathbb{B}$, i.e. $R_\mathbb{A} = 2|x\rangle|p_x\rangle\langle p_x|\langle x| - Id$ et $R_\mathbb{B} = 2|p_y\rangle|y\rangle\langle y|\langle p_y| - Id$. On définit $W(P) := R_\mathbb{B}R_\mathbb{A}$. $W(P)$ est une matrice unitaire dont les valeurs propres dépendent de celles de P de la manière suivante : si on écrit $\lambda_i = \cos(\alpha_i)$ pour les valeurs propres de P, les valeurs propres de $W(P)$ seront de la forme $\mu_i = e^{\pm 2i\alpha_i}$. Alors on a en particulier que :

   - $W(P)$ a un vecteur propre de valeur propre 1, car $\lambda_1 = 1 \Rightarrow \alpha_1 = 0 \Rightarrow \mu_1 = e^0 = 1$.
   - Pour tout i, $1 - \delta \geq |\lambda_i| = \cos(\alpha_i) \geq 1 - \frac{\alpha_i^2}{2} \Rightarrow \alpha_i \geq \sqrt{2\delta}$.

   Par conséquent, il suffit de connaître $\alpha \pm \frac{\sqrt{\delta}}{2}$ pour savoir si $\alpha_i = 1$ ou non. Autrement dit, il faut $n = O(-\log(\sqrt{\delta}))$ bits de précision.
2. On note $EP_{W(P)}$ l'algorithme de l'estimation de phase sur $W(P)$. Rappelons que pour une estimation à n bits de précision, il fera $2^n = O(\frac{1}{\sqrt{\delta}})$ fois appel à $W(P)$. On notera $\tilde{\alpha}$ l'estimation de l'angle α.
3. On note a l'oracle qui, appliqué à un angle α, renvoie 0 si $\alpha = 0$ et 1 sinon.

On peut maintenant définir $R(P) : |U\rangle \mapsto |U\rangle, |V\rangle \mapsto -|V\rangle \qquad \forall |V\rangle \perp |U\rangle$.



L'algorithme de recherche d'un sommet marqué à l'aide d'une marche aléatoire est alors analogue dans la forme à sa version classique.

Si l'on note :

- S : coût de préparation de $|U\rangle$ (Setup)
- C : coût de l'opération de vérification (Check) $|x\rangle|y\rangle \mapsto (-1)^{\mathbb{I}_M(x)}|x\rangle|y\rangle$
- U : coût de l'opération de mise à jour $W(P)$ (Update) : il s'agit du coût des opérations $|x\rangle|0\rangle \mapsto |x\rangle|p_x\rangle$ et $|p_y\rangle|y\rangle \mapsto |0\rangle|y\rangle$ ainsi que leurs inverses.

La marche aléatoire quantique a la complexité suivante :

$$\boxed{S + \frac{1}{\sqrt{\varepsilon}}(C + \frac{1}{\sqrt{\delta}}U)}$$

De même que dans le cas classique, on cherche à faire en sorte que C soit dominé par $\frac{1}{\sqrt{\delta}}U$, de sorte à avoir une complexité en $O(\frac{1}{\sqrt{\delta\varepsilon}})$.

2.3.3 Quantum Walk avec structure de données

Étant donné un ensemble Ω on se donne une fonction $\mathcal{D} : \Omega \to \mathcal{H}_D$, qui associe à un élément x de Ω un élément $\mathcal{D}(x)$ de l'espace de Hilbert $\mathcal{H}_D$. On appellera **structure de données** l'ensemble $(\mathcal{D}(x))_{x\in\Omega}$.
L'élément $\mathcal{D}(x)$ dépend souvent d'une fonction $f : \Omega \to \{0,1\}$, dans ce cas on écrira $\mathcal{D}_f(x)$ pour plus de clarté.

On s'intéresse donc au produit tensoriel $|x\rangle|y\rangle|\mathcal{D}(x)\rangle$. Cela change les étapes de l'algorithme de la manière suivante :

- Setup : Préparer $|U_{\mathcal{D}}\rangle := \frac{1}{\sqrt{N}} \sum_{x,y=0}^{N-1} |x\rangle|p_x\rangle|\mathcal{D}(x)\rangle$.
- Check : $|x\rangle|y\rangle|\mathcal{D}(x)\rangle \mapsto (-1)^{\mathbb{I}_M(x)}|x\rangle|y\rangle|\mathcal{D}(x)\rangle$.
- Update : cette étape fera intervenir en plus les fonctions $|x\rangle|0\rangle|\mathcal{D}(x)\rangle \mapsto |x\rangle|y\rangle|\mathcal{D}(x)\rangle$ et $|x\rangle|y\rangle|\mathcal{D}(x)\rangle \mapsto |x\rangle|y\rangle|\mathcal{D}(y)\rangle$ et leurs inverses.

Comme nous le verrons, l'intérêt de l'incorporation des structures de données vient principalement du fait qu'elles permettent d'accélérer la vérification et donc de diminuer C.

2.4 Applications

2.4.1 Recherche de collisions (QW)

On peut appliquer la marche aléatoire quantique pour trouver des arguments d'une fonction qui produisent une collision. Plus précisément, il s'agit de résoudre le problème suivant :

Problème 4 (Recherche de collisions). *Étant donné $f : \{0,1\}^n \to E$ (E est un ensemble quelconque), $N := 2^n$ trouver $0 \leq i \neq j \leq N-1$ tels que $f(i) = f(j)$ ou dire si la fonction est injective.*

Ce problème est le plus diffcile lorsqu'il existe une unique paire (i,j) produisant une collision. Par conséquent on fera cette hypothèse dans l'analyse qui suit.

On prend un paramètre r qui sera optimisé plus tard et on considère le graphe de Johnson $J(N,r)$. On associe à chaque sommet $R \subset \{1,...,N\}$ la structure de données $\mathcal{D}_f : R \mapsto f(R) := (f(k))_{k\in R}$. On peut ainsi voir chaque sommet comme un couple $(R, f(R))$.

On a alors besoin des ingrédients suivants pour calculer la complexité en requêtes d'une marche aléatoire quantique sur ce graphe :

1. On sait que $\delta = \frac{N}{r(N-r)} = \Omega(\frac{1}{r})$.
2. L'ensemble M contient les sommets R tels qu'il existe $i \neq j \in R$, $f(i) = f(j)$. Cela arrive avec probabilité $\frac{r(r-1)}{N(N-1)}$ d'où une proportion $\varepsilon \approx (\frac{r}{N})^2$ de sommets marqués.
3. Setup : Pour préparer $|U_{\mathcal{D}_f}\rangle$ il faut connaître l'évaluation par f de tous les r éléments de R pour tous les sommets R. Cela nécessite donc r requêtes, d'où $S = O(r)$.
4. Check : On voit directement sur $f(R)$ si $f(R) \in M$, ainsi $C = 0$.
5. Update : Pour mettre à jour un état après une étape de la marche aléatoire, pour chaque sommet R, il faut créer une superposition d'arêtes (R, R') et mettre à jour la structure de données, i.e. calculer les $f(R')$ à partir de $f(R)$. Pour ce faire, il faut faire une requête à f sur une superposition des R' pour l'élément ajouté à chacun d'entre eux. Ainsi $U = O(1)$.

Par conséquent, la complexité en requêtes est :

$$\begin{aligned} S + \frac{1}{\sqrt{\varepsilon}}(C + \frac{1}{\sqrt{\delta}}U) &= O(r) + \frac{N}{r}(0 + \sqrt{\frac{r(N-r)}{N}}O(1)) \\ &= O(r + \frac{N}{\sqrt{r}}) \end{aligned}$$

La complexité est minimale lorsque les deux termes de la somme sont égaux, c'est-à-dire $r = N^{\frac{2}{3}}$.

On obtient ainsi une complexité en requêtes de $O(N^{\frac{2}{3}})$ dont il a été montré qu'elle est optimale.

2.4.2 Recherche de collisions (Grover)

Dans [4] nous trouvons une application de l'algorithme de Grover au problème calculatoire de la somme des sous-ensembles afin de le résoudre par une recherche de collisions. Nous allons donner ici une généralisation de ce procédé afin de résoudre le problème suivant :

Problème 5 (Recherche de collisions généralisée). *Étant donné $f_i : S_i \to E$, $i = 1, 2$, S_1, S_2 et E des ensembles quelconques, trouver $(s_1, s_2) \in S_1 \times S_2$ tel que $f_1(s_1) = f_2(s_2)$.*

Dans l'algorithme qui résout ce problème à l'aide de l'algorithme de Grover et que l'on nommera *CollisionGrover*, on commence par construire la liste $L_1 := \{(f_1(s_1), s_1) \mid s_1 \in S_1\}$ et on la trie.
Ensuite, on définit la fonction $\mathcal{C}_{L_1} : S_2 \to \{0, 1\}$ comme suit :

$$\mathcal{C}_{L_1}(s_2) = \begin{cases} 1 & \text{s'il existe } s_1 \text{ tel que} (f_2(s_2), s_1) \in L_1, \\ 0 & \text{sinon.} \end{cases}$$

Enfin, on se sert de l'algorithme de Grover pour trouver $s_2 \in S_2$ tel que $\mathcal{C}_{L_1}(s_2) = 1$ et on cherche le s_1 correspondant et on retourne (s_1, s_2).

On donne le pseudo-code de cet algorithme :

Algorithme 4 : CollisionGrover

Entrées : S_1, S_2, f_1, f_2
Sorties : $(s_1, s_2) \in S_1 \times S_2$ tels que $f_1(s_1) = f_2(s_2)$, $NULL$ s'il n'y a pas de collision

1 $L_1 \leftarrow$ Nouvelle liste
2 Construire et trier la liste L_1
3 Trouver $s_2 \in S_2$ tel que $\mathcal{C}_{L_1}(s_2) = 1$ avec l'algorithme de Grover
4 **retourner** (s_1, s_2) ou $NULL$ si aucun tel élément n'a pu être trouvé

Analyse de complexité On suppose que les temps de calcul de f_1 et de f_2 sont de même ordre de grandeur T_f. Pour construire L_1, on fait $|S_1|$ requêtes à f_1. Par conséquent, cette liste est construite et triée en temps $\tilde{O}(|S_1|T_f)$.
Pour trouver $s_2 \in S_2$ tel que $\mathcal{C}_{L_1}(s_2) = 0$, on effectue une recherche dans un ensemble de taille $|S_2|$, en faisant une requête à f_2 pour chaque élément, et on cherche $(f_1(s_1), s_1) \in L_1$ tel que $f_1(s_1) = f_2(s_2)$. Cela se fait donc en temps $O\left(\sqrt{|S_2|}\left(T_f + \log_2(|S_1|)\right)\right)$.
$\Rightarrow$ La complexité en temps totale est en $\tilde{O}\left(\left(|S_1| + \sqrt{|S_2|}\right)T_f\right)$.
En particulier, pour que l'algorithme de Grover ait un intéret dans ce cas, il faudrait avoir $|S_1| << |S_2|$, le cas optimal étant $|S_1| = O(\sqrt{|S_2|})$.

Quant à la complexité en espace, on stocke L_1 qui est de taille $O(|S_1|)$. Par conséquent la complexité en espace est en $O(|S_1|)$.

2.5 Considérations plus poussées

2.5.1 Quantum Walk sur $J(N, r)$ VS Grover

Il y a plusieurs facteurs qui entrent en jeu lorsque l'on considère si l'usage d'une marche aléatoire quantique est pertinent dans un cas de figure, par exemple le coût de mise à jour.
En général, les marches aléatoires quantiques sont intéressantes pour résoudre les problèmes ayant la forme suivante :

Problème 6 (Recherche de k-uplet). *Étant donné $f : E_1 \times ... \times E_k \to E$ (les E_i et E étant des ensembles quelconques), trouver $(x_1, ..., x_k) \in E_1 \times ... \times E_k$ tels que $f(x_1, ..., x_k) = 0$.*

Par exemple, le problème de la recherche de collisions sur une fonction $g : \{0,1\}^n \to E$, que nous avons déjà vu, peut-être vu comme le problème de recherche suivant : trouver $(i, j) \in (\{0,1\}^n)^2$ tel que $f(i, j) = 0$, où $f(i, j) := g(i) - g(j)$.

Pour simplifier, nous supposerons que les E_i sont de même taille N. Le problème peut alors être vu comme un problème de recherche dans un ensemble de taille N^k, auquel cas nous pouvons utiliser l'algorithme de Grover pour le résoudre en $O(N^{\frac{k}{2}})$ requêtes à la fonction f.
Il est préférable d'utiliser une marche aléatoire quantique sur un graphe de Johnson $J(N, r)$ à la place de l'algorithme de Grover dans les cas où, bien que

la solution cherchée soit un k-uplet, on peut représenter l'espace de recherche de manière plus "compacte", typiquement sous forme de liste, en raison de certaines dépendances entre les éléments. À titre d'exemple, dans le cas de la recherche de collisions, on cherche un couple d'éléments $(i, j) \in \{1, ..., N\}^2$ tel que $f(i) = f(j)$, mais la marche s'effectue sur un sous-ensemble de $\{1, ..., N\}$.

NOTATION : On utilisera la notation suivante pour une marche aléatoire quantique sur le graphe de Johnson $J(N, r)$ qui résout le problème de recherche de k-uplet pour la fonction f : $QW\left(J(N, r), \{f : E_1 \times ... \times E_k \to E\}\right)$.

Les considérations mathématiques suivantes permettent de vérifier cette afirmation :

Dans le cas d'une marche aléatoire sur $J(N, r)$ où ce que l'on cherche est un k-uplet $(x_1, ..., x_k)^k \in \{1, ..., N\}^k$, ε vaut typiquement $O\left(\left(\frac{r}{N}\right)^k\right)$ et le trou spectral δ vaut $\Omega(r^{-1})$, ce qui fait que le produit $\frac{1}{\sqrt{\varepsilon\delta}}$ vaut $\frac{N^{\frac{k}{2}}}{r^{\frac{k-1}{2}}}$.

En revanche, s'il était impossible de représenter l'espace de recherche sous une forme plus compacte que celle d'un ensemble de k-uplets, la marche aléatoire s'effectuerait sur le graphe $J(N^k, r^k)$, auquel cas δ vaudrait $\Omega(r^{-k})$ et par conséquent le produit $\frac{1}{\sqrt{\varepsilon\delta}}$ vaudrait $N^{\frac{k}{2}}$. Or puisque N^k est la taille de l'espace des solutions, ceci est la complexité que l'on aurait en utilisant Grover et donc la marche aléatoire ne ferait pas mieux que Grover.

2.5.2 Imbrication d'algorithmes

Imbrication dans l'algorithme de Grover Il est possible d'imbriquer un algorithme de recherche quantique (Grover ou QW) dans l'algorithme de Grover de deux manières :

1. Soit il est pertinent de s'en servir pour améliorer le temps de calcul de l'oracle χ et la fonction associée O_χ, c'est-à-dire que l'on s'en sert dans l'étape d'inversion de la phase.
2. Soit on remarque que les algorithmes de Grover et QW créent, avant la mesure ou l'étape de vérification, une superposition d'états vérifiant une certaine propriété avec une très forte probabilité. Ainsi on peut choisir pour l'opération notée A dans la description de l'algorithme de Grover la suite d'opérations unitaires qui débouche sur cet état superposé d'un autre algorithme de recherche. Cela peut être vu comme la recherche de solutions partielles d'un problème avant de chercher la solution finale parmi ces solutions partielles. Cette façon d'imbriquer un algorithme de Grover dans un autre a été exposé dans [5], où il a été montré que l'algorithme résultant aura pour complexité $O\left(\sqrt{N^\alpha}\right)$ avec $\alpha < 1$ un paramètre optimisable.
3. Bien sûr, on peut aussi combiner les deux approches.

Imbrication dans la marche aléatoire quantique On rappelle qu'une marche aléatoire (classique comme quantique) a trois étapes principales : SETUP, UPDATE et CHECK. On peut imbriquer une marche aléatoire quantique au sein

d'une autre de deux manières : comme procédure de mise à jour ou comme procédure de vérification [8].

Dans les deux cas, des modifications sont nécessaires pour exploiter la construction de manière non-triviale. On va se concentrer sur les marches aléatoires comme procédure de vérification, et on introduit les terminologies de *marche externe* pour la marche aléatoire principale et *marche interne* pour la marche aléatoire qui lui sert de procédure de vérification.

Si l'on note ε, δ, S, U et C les termes et coût des étapes de la marche externe, et ε', δ', S', U' et C' ceux qui interviennent dans la marche interne, alors le coût d'une imbrication naïve sera :

— Coût de la marche interne :

$$S' + \frac{1}{\sqrt{\varepsilon'}}(C' + \frac{1}{\sqrt{\delta'}}U') = C$$

— Coût de la marche externe :

$$S + \frac{1}{\sqrt{\varepsilon}}(C + \frac{1}{\sqrt{\delta}}U)$$

$$S + \frac{1}{\sqrt{\varepsilon}}(S' + \frac{1}{\sqrt{\varepsilon'}}(C' + \frac{1}{\sqrt{\delta'}}U') + \frac{1}{\sqrt{\delta}}U)$$

Une imbrication moins naïve utilise la notion de **structures de données quantiques** pour avoir un coût en :

$$S + S' + \frac{1}{\sqrt{\varepsilon}}(\frac{1}{\sqrt{\varepsilon'}}(C' + \frac{1}{\sqrt{\delta'}}U') + \frac{1}{\sqrt{\delta}}U^*)$$

Avec $U^* = O(U)$ dans les cas les plus favorables.

Structures de données quantiques On a défini une structure de données associée comme l'ensemble des $\mathcal{D}(x)$ pour tous les sommets x du graphe. Chaque $\mathcal{D}(x)$ a été défini comme un élément d'un espace de Hilbert $\mathcal{H}_D$. Mais on peut aussi envisager une superposition de ces éléments, auquel cas on parle de structures de données quantiques.

Dans le cas de l'imbrication de marches aléatoires quantiques, on peut mettre à l'oeuvre cette idée de la manière suivante :

Dans l'imbrication naïve, il y a une structure de données associée à la marche externe, $(\mathcal{D}(x))_x$, et une autre associée à la marche interne, qui dépend de l'état où se trouve la marche externe, $(\mathcal{D}_x(u))_u$.

Dans l'imbrication plus intelligente, on stocke dans la structure de données de la marche externe l'état inital de la marche interne (y compris la structure de données internes). Autrement dit, on définit $\mathcal{D}(x)$ comme :

$$\mathcal{D}(x) := \mathcal{D}'(x) \otimes \left(\frac{1}{\sqrt{N}} \sum_u |u\rangle |\mathcal{D}_x(u)\rangle \right)$$

Où éventuellement $\mathcal{D}'(x) = \emptyset$.

Il y a aussi une relation entre les ensembles des éléments marqués des deux marches. L'ensemble des éléments marqués de la marche interne dépend aussi de l'état où se trouve la marche externe, c'est pourquoi, si cet état est noté x, on notera l'ensemble des éléments marqués de la marche interne M_x. L'ensemble M des éléments de la marche externe est à son tour définie de la manière suivante en fonction des ensembles M_x :

$$M = \{x \mid M_x \neq \emptyset\}$$

3 Décodage par ensemble d'information (ISD)

Étant donné un code C de paramètres $[n, k]$ et sa matrice de parité H, on dit que $S \subset \{1, ..., n\}$ est un ensemble d'information si $|S| = k$. Une technique qui est basée sur l'utilisation de tels ensembles pour le décodage du code s'appelle décodage par ensemble d'information ou information set decoding (ISD) en anglais.

Les détails des calculs de complexité et des affirmations autour des paramètres optimaux ainsi que le code pour calculer les courbes de complexité se trouvent dans l'appendice C.

3.1 Algorithme de Prange

Le premier algorithme ISD et le seul algorithme ISD à proprement parler est l'algorithme de Prange : un algorithme probabiliste de type Las Vegas qui échantillonne les ensembles d'information en espérant que le vecteur d'erreur solution e de poids t ait toutes ses positions d'erreurs dans **le complémentaire de l'ensemble d'information** $\overline{S}$ ($|\overline{S}| = n-k$). Cela réduit ainsi le problème du décodage à celui de la résolution d'un système linéaire, une tâche de complexité polynomiale.

3.1.1 Version classique

On notera cet algorithme P.

L'algorithme est constitué d'une boucle sur les ensembles d'information possibles et pour chaque ensemble d'information, on procède comme suit :

1. On note A la sous-matrice de H dont les colonnes sont indicées par $\overline{S}$.
2. Si A est inversible, on fait $A^{-1}s = e$ et si $w_H(e) = t$, on retourne e.
3. Si une des conditions sur A ou sur e n'est pas vérifiée, on recommence avec un autre ensemble d'information.

Ainsi le coût des opérations au sein de chaque tour de boucle est polynomial.

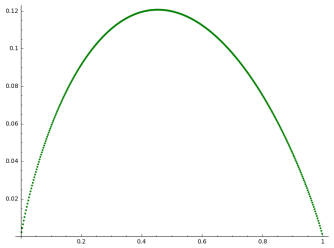
On dit qu'un ensemble d'information est bon si le tour de boucle retourne le

bon vecteur e, ce qui arrive si la sous-matrice A est inversible (cet événement arrive dans 29% des cas [3]) et que $w_H(e) = t$. Si l'on note P_P la probabilité de tomber sur un bon ensemble d'information, la complexité temporelle de cet algorithme est en $\tilde{O}(\frac{1}{P_P})$.

THÉORÈME 13 ([15]). *Si H est une matrice aléatoire, en notant $R := \frac{k}{n}$ et $\tau := \frac{t}{n}$, on a l'expression suivante pour l'exposant α_P de l'algorithme de Prange :*

$$\boxed{\alpha_P(R) = H(\tau) - (1-R)H(\frac{\tau}{1-R})}$$

Et $\max_R(\alpha_P) = 0.1207019$ *pour* $R = 0.4537$.



Courbe de α_P en fonction de R

Démonstration. Rappelons que le vecteur d'erreur devrait avoir la forme suivante : les t positions d'erreur sont parmi les $n-k$ indices de $\overline{S}$.
Par conséquent $P_P = \frac{\binom{n-k}{t}}{\binom{n}{t}}$, et l'algorithme a une complexité en $\tilde{O}\left(\frac{\binom{n}{t}}{\binom{n-k}{t}}\right)$.
Afin d'avoir l'exposant de l'algorithme en fonction de R (et d'autres paramètres), nous utiliserons théorème 1 (page 1) : $\binom{n}{t} \approx 2^{nH(\frac{t}{n})}$ et on écrira $R := \frac{k}{n}$ et $\tau = \frac{t}{n}$.

$$\begin{aligned}\frac{\binom{n}{t}}{\binom{n-k}{t}} &\approx \frac{2^{nH(\frac{t}{n})}}{2^{(n-k)H(\frac{t}{n-k})}} \\ &\approx \frac{2^{nH(\tau)}}{2^{n(1-R)H(\frac{\tau}{1-R})}} \\ &\approx 2^{n(H(\tau)-(1-R)H(\frac{\tau}{1-R}))}\end{aligned}$$

On obtient ainsi l'expression suivante pour l'exposant α_P :

$$\boxed{\alpha_P(R) = H(\tau) - (1-R)H(\frac{\tau}{1-R})}$$

□

3.1.2 Version quantique avec Grover

On donne ici une version quantique de l'algorithme de Prange, donné pour la première fois dans [3], que l'on notera PG.

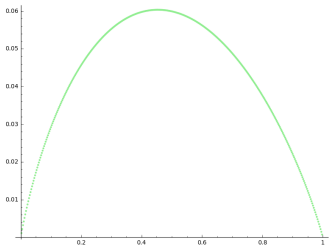
Dans cet algorithme, on part de la même hypothèse sur la répartition des erreurs (donc, si l'on note P_{PG} la probabilité de tomber sur le bon ensemble d'information, on a $P_{PG} = P_P$). On utilise l'algorithme de Grover pour trouver le bon ensemble d'information (l'oracle pour l'algorithme de Grover s'obtient

facilement en modifiant la suite d'étapes détaillée dans la partie précédente pour renvoyer 1 si l'ensemble d'information est bon et 0 sinon). Nous obtenons ainsi un algorithme de complexité temporelle $\tilde{O}(\sqrt{\frac{1}{P_{PG}}})$.

THÉORÈME 14 ([3]). *Si H est une matrice aléatoire, en notant $R := \frac{k}{n}$ et $\tau := \frac{t}{n}$, on a l'expression suivante pour l'exposant α_{PG} de l'algorithme de Prange+Grover :*

$$\boxed{\alpha_{PG}(R) = \frac{H(\tau) - (1-R)H(\frac{\tau}{1-R})}{2}}$$

Et $\max_R(\alpha_{PG}) = 0.0603509$ *pour* $R = 0.4537$



Courbe de α_{PG} en fonction de R

Démonstration. L'algorithme a une complexité temporelle en $\tilde{O}(\sqrt{\frac{1}{P_{PG}}}) = \tilde{O}\left(\sqrt{\frac{\binom{n}{t}}{\binom{n-k}{t}}}\right)$.
On en déduit l'expression suivante pour l'exposant α_{PG} :

$$\boxed{\alpha_{PG}(R) = \frac{H(\tau) - (1-R)H(\frac{\tau}{1-R})}{2}}$$

□

3.2 Squelette d'un algorithme ISD

Il y a d'autres algorithmes inspirés de l'algorithme de Prange et qui ont une meilleure complexité. On introduit un paramètre p en plus. Ces algorithmes ISD "généralisés" échantillonnent des ensembles d'indices un peu plus généraux que les ensembles d'informations, en espérant que la majeure partie, $t - p$, des positions d'erreur soit dans le complémentaire de cet ensemble. Ils cherchent ensuite les p positions d'erreur restantes en se servant de stratégies efficaces mais de complexité exponentielle.

Un algorithme ISD comporte donc une phase de recherche d'un ensemble d'indices convenables suivi d'une phase de recherche d'un vecteur d'erreur vérifiant certaines propriétés. Nous allons formaliser cette structure en donnant un squelette pour ces algorithmes et en introduisant un cadre pour l'analyse de leur complexité.

La différence principale entre les algorithmes ISD réside dans la façon dont ils trouvent le vecteur d'erreur. Chaque algorithme A fait appel à une fonction $Recherche_A$ propre à elle, dont la spécification est la suivante :

$Recherche_A : S, H, s, t, p \rightarrow \{e \mid e$ est de poids p sur S et $t-p$ sur $\overline{S}$ et $s = He^T\} \cup \{NULL\}$ où S est un ensemble d'indices, H la matrice de parité du code et s le syndrome de l'erreur cherché.

Nous pouvons maintenant donner le squelette d'un algorithme ISD :

Algorithme 5 : Squelette ISD

Entrées : H, y, $s = Hy^T$, t, p
Sorties : e de poids t tel que $s = He^T$

```
répéter
    Tirer un ensemble d'indices S ⊂ {1, ..., n}
    e ← Recherche_A(S, H, s, t, p)
jusqu'à w_H(e) = t
retourner e ;
```

Analyse de complexité Comme nous l'avons dit, un tour de boucle de l'algorithme débouche sur le bon vecteur d'erreur si ses positions d'erreurs sont réparties d'une certaine manière (sa forme exacte dépend de l'algorithme A). Si P_A est la probabilité qu'il ait cette forme, il faut donc en moyenne P_A^{-1} répétitions de la boucle avant de le trouver. En réalité, il faut également que la sous-matrice indicée par S soit inversible, mais cela ne fait que multiplier le nombre de tours de boucles par un facteur constant.

Le temps d'exécution de la fonction $Recherche_A$ dépend de l'algorithme A. On le notera T_A. Cette fonction pourrait également avoir une complexité spatiale non-négligeable que l'on notera S_A.

L'algorithme aura par conséquent une complexité temporelle en $\tilde{O}\left(P_A^{-1} T_A\right)$ et une complexité spatiale en S_A.

Pour simplifier les notations Dans ce qui suit, nous allons considérer une version modifiée de l'algorithme présenté ci-dessus afin de faciliter la présentation des algorithmes. En effet, étant donné un ensemble d'indices S de taille $n-k$ tel que les colonnes indicées par cet ensemble forment une sous-matrice inversible, on considère $P \in \mathcal{M}_{n,n}$ la matrice qui permute les colonnes de H indicées par S avec les dernières colonnes de H. Alors, la matrice HP possède une sous-matrice carrée à droite. On peut alors faire un pivot de Gauss T sur cette matrice afin d'obtenir la matrice $THP := \tilde{H} = [A|Id_{n-k}], A \in \mathcal{M}_{n-k,k}$. Alors, si l'on note $\tilde{s} := Ts$ et $\tilde{e} := eP$, on a :

1. Comme P est une permutation, $w_H(\tilde{e}) = w_H(\tilde{e}P^{-1}) = w_H(e) = t$.
2. $\tilde{s} = \tilde{H}\tilde{e}^T \Leftrightarrow Ts = THP(eP)^T = THPP^{-1}e^T \Leftrightarrow Ts = THe^T \Leftrightarrow s = He^T$

Par conséquent, nous allons modifier l'algorithme de la manière suivante : on transforme d'abord H et s en $\tilde{H}$ et $\tilde{s}$, puis on applique $Recherche_A$ à ces arguments et à l'ensemble $\tilde{S}$ des $|S|$ derniers indices, ce qui fait qu'à la fin de l'algorithme, le résultat est $\tilde{e}$ et on retourne ensuite $\tilde{e}P^{-1} = e$.

Cela étant un point de détail qui permet de faciliter la présentation, afin d'alléger la notation, dorénavant nous écrirons directement $H = [A|Id_{n-k}]$.

3.3 Décodage par ensemble d'information quantique

Nous allons généraliser ici l'approche adoptée dans [3] pour l'algorithme de Prange quantique.

On rappelle la spécification et la complexité en requêtes de l'algorithme de Grover.
$Grover : \varepsilon, \{f : E \to \{0,1\}\} \mapsto |X\rangle$, où $|X\rangle$ est une superposition d'états $x \in E$ tel que $f(x) = 1$. Cela s'effectue en $O\left(\frac{1}{\sqrt{\varepsilon}}\right)$ requêtes à la fonction f, où ε est la proportion des $x \in E$ tel que $f(x) = 1$.

Étant donné une matrice H, un syndrome s, des entiers t et p et un algorithme de décodage générique A, on définit la fonction $\mathcal{F}_A : S \mapsto \{0,1\}$ comme la fonction qui exécute $Recherche_A(S, H, s, t, p)$ et renvoie 1 si le vecteur renvoyé n'est pas $NULL$ et est de poids t, et 0 sinon.

Alors, pour trouver une erreur de poids t tel que $s = He^T$, il suffit d'utiliser l'algorithme de Grover sur $\mathcal{F}_A$ pour obtenir un ensemble d'indices S. Il suffit ensuite d'appliquer $Recherche_A$ avec cet ensemble d'indices.

Ainsi, un algorithme ISD quantique a la forme générale suivante :

Algorithme 6 : Squelette ISD quantique

Entrées : H, y, $s = Hy^T$, t, p
Sorties : e de poids t tel que $s = He^T$
1 $|S\rangle \leftarrow Grover(P_A, \mathcal{F}_A)$
2 $|S_0\rangle \leftarrow Mesure(|S\rangle)$
3 $e \leftarrow Recherche_A(S_0, H, s, t, p)$
4 **retourner** e ;

La fonction $\mathcal{F}_A$ a la même complexité temporelle et spatiale que $Recherche_A$, c'est-à-dire respectivement $\tilde{O}(T_A)$ et S_A. L'algorithme de Grover trouve S au bout de $O\left(\sqrt{P_A^{-1}}\right)$ requêtes à $\mathcal{F}_A$, d'où une complexité temporelle totale de $\tilde{O}\left(\sqrt{P_A^{-1}}T_A\right)$. La complexité de l'étape 3 est $\tilde{O}(T_A)$ et donc la complexité temporelle totale d'un algorithme ISD quantique est $\tilde{O}\left(\sqrt{P_A^{-1}}T_A\right)$ (et sa complexité spatiale est $\tilde{O}(S_A)$).

Nous finissons par une remarque générale sur les algorithmes ISD : dans les algorithmes classiques A autres que Prange, $P_A^{-1} \leq P_P^{-1}$ mais $T_A >> P_P$. En effet, T_P est polynomial, mais dans tous les autres algorithmes T_A est exponentiel. Les algorithmes qui ont battu l'algorithme de Prange classique ont réussi à avoir $P_A^{-1}T_A < P_P^{-1}$. Pour battre l'algorithme de Prange quantique, on cherche des algorithmes A tels que $\sqrt{P_A^{-1}}T_A = \sqrt{P_A^{-1}{T_A}^2} < \sqrt{P_{PG}^{-1}}$.

On va donc se focaliser sur l'algorithme $Recherche_A$ et les diverses façons de l'améliorer en se servant d'algorithmes quantiques. De plus, en regardant la

formule de complexité $\sqrt{{P_A^{-1}T_A}^2}$, on se rend compte qu'il faut des améliorations bien plus importantes qu'en classique pour battre PG.

3.4 Algorithme de Lee-Brickell

L'algorithme de Lee-Brickell s'obtient à partir de l'algorithme de Prange en relâchant la contrainte suivante : au lieu d'exiger d'avoir toutes les erreurs dans la partie identité de H, on choisit un paramètre $p < t$ et on fait l'hypothèse qu'il y a $t - p$ erreurs dans la partie identité et p erreurs dans l'autre partie. Cela permettrait de recouvrir un plus grand ensemble d'erreurs possibles.

3.4.1 Version classique

Description de l'algorithme On notera cet algorithme LB.

On écrira indifféremment $e = e_1 + e_2$ et $e = (e_1|e_2)$ avec e_1 de longueur k, e_2 de longueur $n - k$, $w_H(e_1) = p$ et $w_H(e_2) = t - p$.

En considérant la matrice $H = [A|Id_{n-k}]$ on a alors $s := Hy^T = He^T = Ae_1 + I_{n-k}e_2$. Donc $s = Ae_1 + e_2 \Leftrightarrow s - e_2 = Ae_1$.
Ainsi les $n - k$ dernières composantes de e, c'est-à-dire les composantes de e_2, se lisent directement sur celles de s, et pour trouver les autres il faudrait tester tous les candidats possibles pour e_1.

On définit alors $Recherche_{LB}$ de la manière suivante :

Algorithme 7 : $Recherche_{LB}$

Entrées : G, H, y, $s = Hy^T$, t, p
Sorties : e de poids t tel que $y = mG + e$, $NULL$ si aucun tel vecteur n'a été trouvé

```
1 e_2 ← (s_{k+1}, ..., s_n)
2 si w_H(e_2) ≠ t − p alors
3     retourner NULL
4 s_1 ← s − e_2 = Ae_1
5 pour e'_1 de longueur k et de poids p faire
6     si Ae'_1 = s − e_2 alors
7         retourner e := (e'_1|e_2)
8 retourner NULL
```

Preuve de correction Si $e = (e_1|e_2)$ est le bon vecteur d'erreur, on connaît déjà e_2. Il faut par conséquent chercher e_1 qui est de longueur k et de poids p. On fait donc une boucle sur les vecteurs d'erreurs e'_1 ayant cette forme. A l'intérieur de la boucle, on teste si $Ae'_1 = s - e_2$. Or cela est équivalent à $He'_1 + He_2 = s \Leftrightarrow H(e'_1|e_2) = s$.

Complexité

THÉORÈME 15 ([10]). *Si H est une matrice aléatoire, en notant $R := \frac{k}{n}$ et $\tau := \frac{t}{n}$ et $\pi := \frac{p}{n}$, on a l'expression suivante pour l'exposant α_{LB} de l'algorithme de Lee et Brickell :*

$$\boxed{\alpha_{LB}(R) = \min_{\pi}\left(H(\tau) - (1-R)H(\frac{\tau-\pi}{1-R})\right)}$$

Soumise à la contrainte :
$$0 \leq \pi \leq \tau$$

De plus, le minimum est atteint pour $\pi = 0$, et dans ce cas $\alpha_{LB} = \alpha_P$, c'est-à-dire que l'amélioration apportée par l'algorithme de Lee-Brickell est tout au plus polynomiale.

Démonstration.
(1) Calcul de α_{LB}

Rappelons que le vecteur d'erreur devrait avoir la forme suivante : $t-p$ erreurs parmi les $n-k$ indices de $\overline{S}$ et p erreurs parmi les k indices de S.
Dans le pire cas, la fonction $Recherche_{LB}$ fait une boucle sur $\binom{k}{p}$ éléments.
On a par conséquent :

$$1)\ P_{LB} = \frac{\binom{n-k}{t-p}\binom{k}{p}}{\binom{n}{t}}$$
$$2)\ T_{LB} = \tilde{O}\left(\binom{k}{p}\right)$$
$$3)\ S_{LB} = poly(n) = \tilde{O}(1)$$

L'algorithme a donc une complexité en $\tilde{O}\left(\frac{\binom{n}{t}}{\binom{n-k}{t-p}\binom{k}{p}}\binom{k}{p}\right) = \tilde{O}\left(\frac{\binom{n}{t}}{\binom{n-k}{t-p}}\right)$. On utilise encore théorème 1 (page 1) afin d'obtenir l'exposant α_{LB}. On notera $R := \frac{k}{n}$ et $\tau = \frac{t}{n}$ et $\pi = \frac{p}{n}$. On a de plus les égalités suivantes :

$$\frac{n-k}{n} = 1 - \frac{k}{n} = 1 - R$$
$$\frac{t-p}{n-k} = \frac{t}{n(1-R)} - \frac{p}{n(1-R)} = \frac{\tau-\pi}{1-R}$$

On calcule enfin :

$$\begin{aligned}\frac{\binom{n}{t}}{\binom{n-k}{t-p}} &\approx \frac{2^{nH(\frac{t}{n})}}{2^{(n-k)H(\frac{t-p}{n-k})}} \\ &\approx \frac{2^{nH(\tau)}}{2^{n(1-R)H(\frac{\tau-\pi}{1-R})}} \\ &\approx 2^{n(H(\tau)-(1-R)H(\frac{\tau-\pi}{1-R}))}\end{aligned}$$

On en déduit l'expression suivante pour α_{LB} :

$$\boxed{\alpha_{LB}(R) = \min_{\pi} \left(H(\tau) - (1-R)H(\frac{\tau-\pi}{1-R}) \right)}$$
$$0 \leq \pi \leq \tau$$

(2) Preuve de $\alpha_{LB} = \alpha_P$

On cherche à minimiser cette expression en fonction de π (pour R et τ fixés). Cela revient donc à maximiser $H(\frac{\tau-\pi}{1-R})$. Or $H'(x) = log(\frac{1-x}{x})$. H est donc maximale en $x = \frac{1}{2}$.
Il faut donc $\frac{\tau-\pi}{1-R} = \frac{1}{2} \Leftrightarrow 2(\tau-\pi) = 1-R \Leftrightarrow 2\pi = R-1+2\tau \Leftrightarrow 2p = k+2t-n$. Puisque $t \leq \frac{d-1}{2}$ avec d la distance minimale du code, on a alors $2p \leq k-n+d-1 \leq k-n+n-k+1-1 = 0$.
Donc la complexité de l'algorithme est minimale lorsque $p = 0$, c'est-à-dire qu'on revient à l'algorithme de Prange ! □

3.5 Algorithme de Dumer/Stern

Le grand problème de l'algorithme de Lee-Brickell est le facteur $\binom{k}{p}$ qui donne le nombre de tours de boucle de la fonction de recherche et qui est trop grand. Les algorithmes de Dumer et de Stern, qui sont très proches, tentent de pallier à ce problème en utilisant une recherche de collisions à cette étape de l'algorithme.

Les algorithmes de Dumer et de Stern introduisent un paramètre de plus, h, qui est petit. Nous allons considérer l'algorithme de Dumer dont la complexité est un peu meilleure et qui fait la chose suivante pour pallier aux faiblesses de l'algorithme de Lee-Brickell : au lieu de considérer une fenêtre de taille k dans laquelle $p < t$ erreurs seraient réparties, on considère une fenêtre un peu plus grande, de taille $h + k$.

3.5.1 Version classique

On notera cet algorithme D.

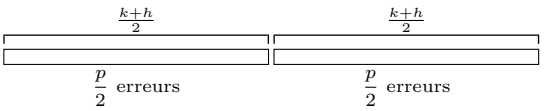
Considérations préliminaires

1. On rappelle que l'on avait supposé, pour simplifier les notations, que $H = [\, A | I_{n-k} \,]$. Notons qu'on peut également écrire la matrice H sous la forme $\begin{bmatrix} H' | 0_{h,n-k-h} \\ B | I_{n-k-h} \end{bmatrix}$ avec $H' \in \mathcal{M}_{h,k+h}$, $B \in \mathcal{M}_{n-k-h,k+h}$.

   Nous nous référons maintenant aux définitions et aux résultats relatifs aux codes poinçonnés à la page 2. On considère plus précisément le code C' obtenu en poinçonnant les $n-k-h$ dernières colonnes de $G := [\, I_k | A^T \,]$, qui est une matrice génératrice de C, La matrice génératrice G' de C' est alors donnée par la sous-matrice de G restreinte aux $k+h$ premières colonnes. Alors, d'après le théorème 3, la matrice H' est une matrice de parité du code C'. Qui plus est, puisque G' admet une sous-matrice inversible (la matrice identité), alors le théorème 2 entraîne que si l'on note $c := mG$, il est possible d'obtenir $c' := m_{|k+h}G'$, à partir de c via l'opération de poinçonnage et inversement, il est possible de trouver c à partir de c' de manière unique.

On aura alors $H'c'^T = 0$. On est donc ramené à un problème de décodage par syndrome sur une matrice de taille plus petite, plus précisément une matrice $(h+k)\times h$. Nous appelerons ce problème le problème de décodage par collision, d'après la technique principale qui sera utilisée pour le résoudre. L'intérêt de se ramener à ce problème est que, bien qu'il y ait moins d'équations de parité (car h est petit), le nombre de motifs d'erreurs possibles a diminué car on se restreint aux mots de codes de poids p du code C').

2. De plus, pour optimiser la recherche de collisions, on fera l'hypothèse que les p erreurs sont réparties de la manière suivante dans la fenêtre de taille $k+h$:



Cette hypothèse n'est pas trop restrictive. En effet, avec le théorème 1 (page 1), on voit que :

$$\binom{\frac{k+h}{2}}{\frac{p}{2}}^2 \approx \left(2^{\frac{k+h}{2}H(\frac{2p}{2(k+h)})}\right)^2 = 2^{(k+h)H(\frac{p}{k+h})} \approx \binom{k+h}{p}$$

3. La recherche de collisions est une technique reposant sur un compromis temps-mémoire. Par conséquent, il faudrait calculer la complexité en espace aussi puisque celle-ci n'est plus négligeable.
4. Il est possible de décrire autrement le problème du décodage par collision, à savoir avec des ensembles d'indices. En effet, si l'on note M_ℓ la ℓème colonne d'une matrice M, alors les deux problèmes suivants sont équivalents :
   (a) Trouver e de longueur n tel que $s = He^T$ et $w_H(e) = t$.
   (b) Trouver $I \subset \{1, ..., n\}$, $|I| = t$, tel que $s = \sum_{\ell \in I} H_\ell$.

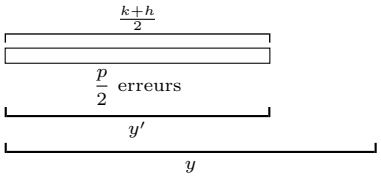
   La deuxième formulation met en lumière des similitudes fortes avec le problème de la somme des sous-ensembles (subset sum) : le problème du décodage par collision peut être vu comme une version vectorielle du problème subset sum. Nous allons en effet commencer à nous servir de cette autre formulation.

Description de l'algorithme

Algorithme de résolution du problème de décodage par collision Afin de pouvoir définir la fonction $Recherche_D$, on doit d'abord définir une fonction qui résout le problème du décodage par collision et que l'on appelera $DecodeCollision_D$.

On fixe d'abord les notations suivantes :

1. Étant donné y, on note y' le vecteur obtenu à partir de y via poinçonnage. Illustration :



2. On écrit $H' = [H'_1|H'_2]$ avec H'_ℓ de longueur $\frac{k+h}{2}$. On écrit aussi $e' = (e_1|e_2) = e_1 + e_2$ avec e_i de longueur $\frac{k+h}{2}$.
3. On écrit H'_ℓ pour la ℓème colonne de la matrice H' et on notera I_i l'ensemble des indices où e_i est non-nul et I' l'ensemble des indices où e' est non-nul. On a alors $|I_i| = \frac{p}{2}, |I'| = p$ et $I' = I_1 \sqcup I_2$.
4. On définit le syndrome s' de longueur h comme la première partie du syndrome s : $s = (s' \mid s")$.

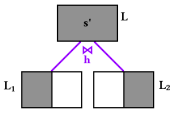
$DecodeCollision_D$ sera une fonction qui trouve e' tel que $s' := H'y'^T = H'e'^T$ (de manière équivalente, I' tel que $s' = \sum_{\ell \in I'} H'_\ell$ via une recherche de collisions). Plus précisément, elle trouve une collision entre les $s' + H'_1 e_1^T$ et les $H'_2 e_2^T$ (de manière équivalente, $s' + \sum_{\ell \in I_1} H'_{1\ell}$ et $\sum_{\ell \in I_2} H'_{2\ell}$).

Algorithme 8 : $DecodeCollision_D$

Entrées : H', y', $s' = H'y'^T$, p
Sorties : une liste L de vecteurs e' tels que $s' = H'e'^T$

$L_1 \leftarrow$ Nouvelle liste
$L_2 \leftarrow$ Nouvelle liste
pour $I_1 \subset \{1, ..., \frac{k+h}{2}\}$ *tel que* $|I_1| = \frac{p}{2}$ **faire**
  $L_1 \leftarrow L_1 \cup (\sum_{\ell \in I_1} H'_{1\ell}, I_1)$
pour $I_2 \subset \{\frac{k+h}{2} + 1, ..., k+h\}$ *tel que* $|I_2| = \frac{p}{2}$ **faire**
  $L_2 \leftarrow L_2 \cup (s' + \sum_{\ell \in I_2} H'_{2\ell}, I_2)$
$L \leftarrow L_1 \overset{s'}{\bowtie} L_2$
$L = (I' := I_1 \sqcup I_2 \mid \sum_{\ell \in I_1} H'_{1\ell} = s' + \sum_{\ell \in I_2} H'_{2\ell})$
retourner L

La liste L contient les unions I' des ensembles I_1 et I_2 tels que $\sum_{\ell \in I_1} H'_{1\ell} = s' + \sum_{\ell \in I_2} H'_{2\ell}$. En définissant $e' = (e_1|e_2)$ comme le vecteur d'erreur dont les composantes non-nulles sont indicées par I', on a ainsi $H'_1 e_1^T = s' + H'_2 e_2^T \Leftrightarrow H'e'^T = s'$.



$DecodeCollision_D$ - illustration sous forme d'arbre
violet : recherche de collisions classique

On obtient ainsi une liste L d'ensembles indices I' correspondant à des vecteurs d'erreurs poinçonnés e'. On définit une opération *Dépoinçonner* : $I' \mapsto e \in \mathbb{F}_2^n$ comme suit :

1. $e' \leftarrow$ vecteur d'erreur dont les composantes non-nulles sont indicées par I'
2. Calculer le message poinçonné $m_{|k+h} = (y' - e')G'^{-1}$ et en déduire m, puis $e \leftarrow y - mG$

L'algorithme $Recherche_D$ On peut maintenant définir $Recherche_D$:

Algorithme 9 : $Recherche_D$

Entrées : H, y, $s = Hy^T$, t, p
Sorties : e tel que $s = mG + e$ et $w_H(e) = t$, $NULL$ si aucun tel vecteur n'a pu être trouvé

```
Préparer G',H',y' et calculer s' = H'y'^T
L ← DecodeCollision_D(H', y', s', p)
pour I' ∈ L faire
    e ← Dépoinçonner(I')
    si w_H(e) = t alors
        retourner e
retourner NULL
```

Complexité

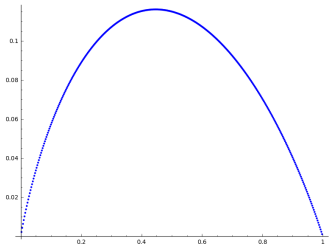
THÉORÈME 16 ([7]). *Si H est une matrice aléatoire, en notant $R := \frac{k}{n}$, $\tau := \frac{t}{n}$, $\pi := \frac{p}{n}$ et $\eta = \frac{h}{n}$, on a l'expression suivante pour l'exposant α_D de l'algorithme de Dumer :*

$$\boxed{\alpha_D(R) = \min_{\pi,\eta}\left(H(\tau) - (1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta}) - \frac{(R+\eta)}{2}H(\frac{\pi}{R+\eta})\right)}$$

Soumise aux contraintes :

$$\pi = (R+\eta)H^{-1}(\tfrac{2\eta}{(R+\eta)})$$
$$0 \leq \pi \leq \min(\tau, R+\eta)$$
$$0 \leq \eta \leq 1-R-\tau+\pi \leq 1-R$$

Et $\max_R(\alpha_D) = 0.1162451$ *pour* $R = 0.4470$*, valeur obtenue pour* $\pi \approx 0.0102646977$ *et* $\eta \approx 0.035839347$.



Courbe de α_D en fonction de R

Démonstration. P_D est la proportion de vecteurs d'erreur de poids t et de longueur n ayant la forme souhaitée ($\frac{p}{2}$ erreurs dans "chaque moitié" des $k+h$ dernières positions et $t-p$ erreurs dans les $n-k-h$ premières positions).
T_D est le temps d'exécution de l'algorithme $Recherche_D$, qui est le temps d'exécution de $DecodeCollision_D$ puis un parcours de la liste L. Or $DecodeCollision_D$ fait un Merge-Join de deux listes de taille environ $\sqrt{\binom{k+h}{p}}$ de vecteurs binaires de longueur h. Par conséquent la taille de la liste L est $O\left(\binom{k+h}{p}2^{-h}\right)$.
On en déduit :

$$1)\ P_D = \frac{\binom{\frac{k+h}{2}}{\frac{p}{2}}\binom{\frac{k+h}{2}}{\frac{p}{2}}\binom{n-k-h}{t-p}}{\binom{n}{t}}$$

$$2)\ T_D = \tilde{O}\left(\max\left(\sqrt{\binom{k+h}{p}}, \binom{k+h}{p}2^{-h}\right)\right)$$

On considère deux cas de figures :

— $2^h << \sqrt{\binom{k+h}{p}} \Leftrightarrow p >> (k+h)H^{-1}(\frac{2h}{(k+h)})$
Alors $T_D = \tilde{O}\left(\binom{k+h}{p}2^{-h}\right)$.

— $2^h \approx \sqrt{\binom{k+h}{p}} \Leftrightarrow p \approx (k+h)H^{-1}(\frac{2h}{(k+h)})$
Alors $T_D = \tilde{O}\left(\sqrt{\binom{k+h}{p}}\right)$

Par conséquent, pour minimiser T_D, il faut minimiser $\binom{k+h}{p}2^{-h}$ avec la contrainte $p >> (k+h)H^{-1}(\frac{2h}{(k+h)})$ et $\sqrt{\binom{k+h}{p}}$ avec la contrainte $p \approx (k+h)H^{-1}(\frac{2h}{(k+h)})$.

Avec ces contraintes sur p en fonction de h et de k, la complexité est par conséquent :

$$\tilde{O}\left(\min\left(\frac{\binom{n}{t}}{\binom{k+h}{p}\binom{n-k-h}{t-p}}\sqrt{\binom{k+h}{p}}, \frac{\binom{n}{t}}{\binom{k+h}{p}\binom{n-k-h}{t-p}2^h}\binom{k+h}{p}\right)\right)$$

Autrement dit, en posant $R := \frac{k}{n}$ et $\tau = \frac{t}{n}$ et $\pi = \frac{p}{n}, \pi := \frac{p}{n}$ et $\eta := \frac{h}{n}$:

$$\tilde{O}\left(2^{n(\min(H(\tau)-(R+\eta)H(\frac{\pi}{R+\eta})-(1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta})+\frac{(R+\eta)}{2}H(\frac{\pi}{R+\eta}), H(\tau)-(1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta})-\eta))}\right)$$

On en déduit l'exposant $\alpha_D = \min(\alpha_{D_1}, \alpha_{D_2})$ avec :

$$\boxed{\alpha_{D_1}(R) = \min_{\pi,\eta}\left(H(\tau) - (1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta}) - \frac{(R+\eta)}{2}H(\frac{\pi}{R+\eta})\right)}$$

Soumise aux contraintes :

$$\pi = (R+\eta)H^{-1}(\tfrac{2\eta}{(R+\eta)})$$
$$0 \le \pi \le \min(\tau, R+\eta)$$
$$0 \le \eta \le 1-R-\tau+\pi \le 1-R$$

$$\boxed{\alpha_{D_2}(R) = \min_{\pi,\eta}\left(H(\tau) - (1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta}) - \eta\right)}$$

Soumise aux contraintes :

$$\pi > (R+\eta)H^{-1}(\tfrac{2\eta}{(R+\eta)})$$
$$0 \le \pi \le \min(\tau, R+\eta)$$
$$0 \le \eta \le 1-R-\tau+\pi \le 1-R$$

En cherchant à optimiser numériquement, nous constatons que $\alpha_{D_1} \le \alpha_{D_2}$, par conséquent $\alpha_D = \alpha_{D_1}$.

Quant à la complexité en espace, le seul endroit où on utilise de l'espace mémoire de façon non-négligeable est dans la fonction $DecodeCollision_D$: on effectue en effet un $Merge-Join$ de deux listes de taille $\sqrt{\binom{k+h}{p}}$ pour construire une liste de taille $\tilde{O}\left(\binom{k+h}{p}2^{-h}\right)$. Par conséquent $S_D = \tilde{O}\left(\min(\sqrt{\binom{k+h}{p}}, \binom{k+h}{p}2^{-h}\right)$. Ainsi la même analyse s'y applique que pour T_D. □

3.5.2 Version quantique avec Grover

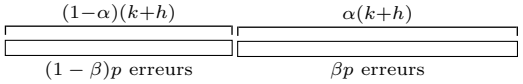
On notera DG cet algorithme.

Nous allons montrer ici une manière d'améliorer l'étape $Recherche_D$ de l'algorithme de Dumer classique. En effet, $DecodeCollsion_D$ est essentiellement un algorithme de recherche de collisions.

Or, on a vu qu'il est possible de faire une recherche de collisions à l'aide de l'algorithme de Grover. On rappelle brièvement la spécification de l'algorithme $CollisionGrover$.
$CollisionGrover : S_1, S_2, f_1, f_2 \mapsto (s_1, s_2) \in S_1 \times S_2, f_1(s_1) = f_2(s_2)$

Une différence qu'il faut prendre en compte est que $CollisionGrover$ renvoie une seule collision, alors que $DecodeCollsion_D$ renvoie une liste de collisions (par conséquent, toutes les collisions). Or, une seule de ces collisions correspond au vecteur d'erreur cherché. Il est donc pertinent de faire une recherche imbriquée où la recherche interne trouve une collision et la recherche externe trouve la bonne collision.

Considérations préliminaires Afin que la recherche de collisions avec l'algorithme de Grover soit efficace vis-à-vis de la recherche de collisions classique, la meilleure stratégie n'est pas, a priori, de diviser de manière uniforme. Nous allons donc prendre des paramètres $\alpha, \beta \in [0, 1]$ qui seront optimisés plus tard, et faire l'hypothèse que les erreurs sont réparties de la manière suivante :



Nous verrons lors des calculs de complexité qu'il est optimal de prendre $\alpha = \beta = 2/3$, et alors, comme dans le cas classique, cette hypothèse fait perdre au plus un facteur polynomial.

Description de l'algorithme La recherche de collisions avec Grover se fera avec les paramètres suivants :

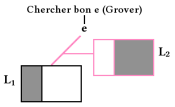
- $e' = (e_1|e_2)$ avec la répartition d'erreur ci-dessus et les ensembles $I' = I_1 \sqcup I_2$ correspondants.
- $S_i =$ Les valeurs de I_i possibles $\Rightarrow |S_1| = \binom{\alpha(k+h)}{\beta p} \approx \binom{k+h}{p}^{1/3}, |S_2| = \binom{(1-\alpha)(k+h)}{(1-\beta)p} \approx \binom{k+h}{p}^{2/3}$.
- $f_1 : I_1 \mapsto \sum_{\ell \in I_1} H'_{1_\ell}$.
- $f_2 : I_2 \mapsto s' + \sum_{\ell \in I_2} H'_{2_\ell}$.

Ainsi l'image des f_i, $i = 1, 2$ est l'ensemble $\{0, 1\}^h$.
On définit ensuite $K := \max\left(1, \binom{k+h}{p} 2^{-h}\right)$ et $Recherche_{DG}$ de la manière suivante :

Algorithme 10 : $Recherche_{DG}$

Entrées : G, H, y, $s = Hy^T$, t, p, K
Sorties : e tel que $y = mG + e$ et $w_H(e) = t$, $NULL$ si aucun tel vecteur n'a pu être trouvé

Préparer G',H',y' et calculer $s' = H'y'^T$
tant que $w_H(e) \neq t$ **faire**
  $(I_1, I_2) \leftarrow CollisionGrover(S_1, S_2, f_1, f_2)$
  $I' \leftarrow I_1 \sqcup I_2$
  $e \leftarrow Dépoinçonner(I')$
retourner e



$Recherche_{DG}$ - illustration sous forme d'arbre
rose : $CollisionGrover$

Preuve de correction $CollisionGrover$ renvoie (I_1, I_2) tels que $f_1(I_1) = f_2(I_2) \Leftrightarrow \sum_{\ell \in I_1} H'_{1_\ell} = s' + \sum_{\ell \in I_2} H'_{2_\ell} \Leftrightarrow \sum_{\ell \in I'} H'_\ell = s' \Leftrightarrow H'e'^T = s'$. Or e' défini ainsi est un vecteur d'erreur parmi K ayant pour syndrome s', donc ce n'est pas nécessairement celui qui donnera le bon vecteur e. C'est pour cette raison que l'on répète cette opération jusqu'à avoir trouvé le bon e. On peut voir cela comme un problème de recherche d'un élément parmi K. Par conséquent, on peut y appliquer l'algorithme de Grover aussi et obtenir ainsi une imbrication de deux algorithmes de Grover. Dans ce cas, on obtient la réponse avec $O(\sqrt{K})$ requêtes à $CollisionGrover$.

Complexité

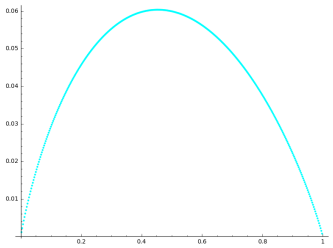
THÉORÈME 17. *Si H est une matrice aléatoire, en notant $R := \frac{k}{n}$, $\tau := \frac{t}{n}$, $\pi := \frac{p}{n}$ et $\eta = \frac{h}{n}$, on a l'expression suivante pour l'exposant α_{DG} de l'algorithme de Dumer+Grover :*

$$\boxed{\alpha_{DG}(R) = \min_{\pi,\eta} \left(\frac{1}{2}(H(\tau) - (1 - R - \eta)H(\frac{\tau - \pi}{1 - R - \eta}) - \frac{(R + \eta)}{3} H(\frac{\pi}{R + \eta})) \right)}$$

Soumise aux contraintes :

$$\pi = (R + \eta)H^{-1}(\tfrac{\eta}{(R+\eta)})$$
$$0 \leq \pi \leq \tau$$
$$0 \leq \pi \leq R + \eta$$
$$0 \leq \eta \leq 1 - R - \tau + \pi \leq 1 - R$$

Et $\max_R(\alpha_{DG}) = 0.0603227$ *pour* $R = 0.4654$, *pour* $\pi \approx 4.654e - 13$ *et* $\eta \approx 1.01451e - 11$.



Courbe de α_{DG} en fonction de R

Démonstration. On a :

$$1)\ P_{DG} = \frac{\binom{(1-\alpha)(k+h)}{(1-\beta)p}\binom{\alpha(k+h)}{\beta p}\binom{n-k-h}{t-p}}{\binom{n}{t}}$$

$$2)\ T_{DG} = \tilde{O}\left(\sqrt{K}(|S_1| + \sqrt{|S_2|})\right) = \tilde{O}\left(\sqrt{K}\left(\binom{(1-\alpha)(k+h)}{(1-\beta)p} + \sqrt{\binom{\alpha(k+h)}{\beta p}}\right)\right)$$

L'algorithme a donc une complexité en :

$$\tilde{O}\left(\sqrt{\frac{\binom{n}{t}}{\binom{(1-\alpha)(k+h)}{(1-\beta)p}\binom{\alpha(k+h)}{\beta p}\binom{n-k-h}{t-p}}}\sqrt{K}\left(\binom{(1-\alpha)(k+h)}{(1-\beta)p} + \sqrt{\binom{\alpha(k+h)}{\beta p}}\right)\right)$$

$$\tilde{O}\left(\sqrt{K}\left(\sqrt{\frac{\binom{n}{t}\binom{(1-\alpha)(k+h)}{(1-\beta)p}}{\binom{\alpha(k+h)}{\beta p}\binom{n-k-h}{t-p}}} + \sqrt{\frac{\binom{n}{t}}{\binom{(1-\alpha)(k+h)}{(1-\beta)p}\binom{n-k-h}{t-p}}}\right)\right)$$

Pour minimiser cette expression, on pourrait essayer de rendre égaux les deux termes de la somme. Cela arrive ssi :

$$\binom{\alpha(k+h)}{\beta p} \approx \binom{(1-\alpha)(k+h)}{(1-\beta)p}^2$$
$$\Leftrightarrow$$
$$\binom{\alpha(k+h)}{\beta p} \approx \binom{2(1-\alpha)(k+h)}{2(1-\beta)p}$$
$$\Leftrightarrow$$

$2(1-\alpha) = \alpha$ et $2(1-\beta) = \beta$, i.e. $\alpha = \beta = 2/3$

On a alors :

$$\tilde{O}\left(\sqrt{K}\sqrt{\frac{\binom{n}{t}}{\binom{k+h}{p}^{1/3}\binom{n-k-h}{t-p}}}\right)$$

$$\tilde{O}\left(\max\left(\sqrt{\frac{\binom{n}{t}}{\binom{k+h}{p}^{1/3}\binom{n-k-h}{t-p}}}, \sqrt{\frac{\binom{n}{t}}{\binom{k+h}{p}^{1/3}\binom{n-k-h}{t-p}}\frac{\binom{k+h}{p}}{2^h}}\right)\right)$$

Il y a alors deux cas de figures :

- $2^h \approx \binom{k+h}{p}$, auquel cas la complexité sera en $\tilde{O}\left(\sqrt{\frac{\binom{n}{t}}{\binom{k+h}{p}^{1/3}\binom{n-k-h}{t-p}}}\right)$.
- $2^h << \binom{k+h}{p}$, auquel cas la complexité sera en $\tilde{O}\left(\sqrt{\frac{\binom{n}{t}\binom{k+h}{p}^{2/3}}{\binom{n-k-h}{t-p}2^h}}\right)$

On en déduit l'exposant $\alpha_{DG} = \min(\alpha_{DG_1}, \alpha_{DG_2})$ avec :

$$\boxed{\alpha_{DG_1}(R) = \min_{\pi,\eta}\left(\frac{1}{2}(H(\tau) - (1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta}) - \frac{(R+\eta)}{3}H(\frac{\pi}{R+\eta}))\right)}$$

Soumise aux contraintes :

$$\pi = (R+\eta)H^{-1}(\tfrac{\eta}{(R+\eta)})$$
$$0 \leq \pi \leq \tau$$
$$0 \leq \pi \leq R+\eta$$
$$0 \leq \eta \leq 1-R-\tau+\pi \leq 1-R$$

$$\boxed{\alpha_{DG_2}(R) = \min_{\pi,\eta}\left(\frac{1}{2}(H(\tau) + \frac{2(R+\eta)}{3}H(\frac{\pi}{R+\eta}) - (1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta})) - \eta)\right)}$$

Soumise aux contraintes :

$$\pi > (R+\eta)H^{-1}(\tfrac{\eta}{(R+\eta)})$$
$$0 \leq \pi \leq \tau$$
$$0 \leq \pi \leq R+\eta$$
$$0 \leq \eta \leq 1-R-\tau+\pi \leq 1-R$$

Comme pour l'algorithme de Dumer classique, nous trouvons que $\alpha_{DG} = \alpha_{DG_1}$. Quant à la complexité en espace, la seule étape où l'on utilise de l'espace mémoire de manière considérable est lors des appels à la fonction $CollisionGrover$ afin de construire une liste de longueur $|S_1| = O\left(\binom{k+h}{p}^{1/3}\right)$. On ne garde pas cette liste d'une itération de la fonction à la suivante, par conséquent $S_{DG} = O\left(\binom{k+h}{p}^{1/3}\right)$. □

L'amélioration n'est pas considérable par rapport à PG, de plus, elle s'obtient pour des valeurs de π et de η très petites. Par exemple, pour $\pi = O(10^{-13})$. Or, si l'on se rappelle que $\pi = \frac{p}{n}$, cette valeur de π signifie en pratique que $p = 0$ lorsque $n < 10^{13}$.

4 ISD avancé : technique des représentations

Comme nous l'avons vu, le problème du décodage par collision peut être vu comme une version vectorielle du problème de la somme des sous-ensembles (subset sum). En effet, il y a déjà eu des travaux pour appliquer les techniques avancées de résolution du subset sum à ce problème ([12], [2], [1]).

Nous reprenons les mêmes notations que dans l'algorithme de Dumer, en écrivant $H = \begin{bmatrix} H'|0_{h,n-k-h} \\ B|I_{n-k-h} \end{bmatrix}$ avec $H' \in \mathcal{M}_{h,k+h}$ la matrice de parité du code poinçonné, etc. Nous prendrons aussi comme hypothèse de répartition des erreurs celle de l'algorithme de Dumer : p erreurs dans l'ensemble S de taille $k+h$ et $t-p$ erreurs dans $\overline{S}$ de taille $n-k-h$. Par conséquent, pour tous les algorithmes A qui suivent, $P_A^{-1} \approx 2^{n(H(\tau)-(R+\eta)H(\frac{\pi}{R+\eta})-(1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta}))}$. Nous supposerons également que la fonction $Recherche_A$ comporte une sous-partie de résolution du problème de décodage par collision, $DecodeCollision_A$. Le but est d'accélérer cette étape.

4.1 Algorithme de Shamir-Schroeppel

L'algorithme de Shamir-Schroeppel est à la base un algorithme pour résoudre le problème du subset sum. C'est un algorithme dont la complexité spatiale est la racine carrée de celle de l'algorithme de Dumer mais dont la complexité temporelle est en théorie identique à celle de Dumer (en pratique, le fait que la complexité spatiale soit inférieure implique une réduction du temps d'exécution en raison du coût des accès à la mémoire).

L'idée derrière cette algorithme est d'ajouter un niveau de recherche de collisions en plus pour la construction des listes L_1 et L_2, ce qui permet de réduire l'espace mémoire utilisé. Par contre, pour que cette recherche de collisions fonctionne, il faut chercher un vecteur intervenant dans une somme intermédiare, ce qui fait qu'au final la complexité en temps asymptotique reste la même.

Enfin, comme nous le verrons, l'intérêt de cet algorithme est qu'il s'adapte bien au cadre quantique.

4.1.1 Version classique

On notera cet algorithme SS.

Considérations préliminaires On fixe les notations suivantes : on écrira $h = h_1 + h_2$ et on va "diviser" H' en deux sous-matrices ayant h_1 et h_2 lignes chacune. Pour simplifier les notations, nous allons définir $H'^{(1)}$ comme étant la matrice H' restreinte aux h_1 premières lignes et $H'^{(2)}$ comme étant la matrice H' restreinte aux lignes $h_1 + 1$ jusqu'à $h_1 + h_2 = h$. Ainsi :

$$H' = \begin{bmatrix} H'^{(1)} \\ H'^{(2)} \end{bmatrix}$$

De même, pour un vecteur colonne v de longueur h, on notera $v^{(1)}$ et $v^{(2)}$.

Si on écrit $J = \{1, ..., h + k\}$ et $S := \{I \subset J \mid |I| = p\}$, S est isomorphe à l'ensemble des vecteurs d'erreurs possibles pour le code poinçonné. on a $|S| = \binom{k+h}{p}$. On considère :

$$\begin{cases} J_{1,1} = \{1, ..., \frac{k+h}{4}\} \\ J_{1,2} = \{\frac{k+h}{4} + 1, ..., \frac{k+h}{2}\} \\ J_{2,1} = \{\frac{k+h}{2} + 1, ..., \frac{3(k+h)}{4}\} \\ J_{2,2} = \{\frac{3(k+h)}{4} + 1, ..., k + h\} \end{cases}$$

Alors, $\sqcup_{i,j=1,2} J_{i,j} = J$. Si l'on définit $S_{i,j} := \{I_{i,j} \subset J_{i,j} \mid |I_{i,j}| = p/4\}$, alors $|S_{i,j}| = \binom{\frac{k+h}{4}}{\frac{p}{4}} \approx \binom{k+h}{p}^{1/4}$.
Et alors proportion des $I \in S$ s'écrivant $I = \sqcup_{i,j=1,2} I_{i,j}$ est donc :

$$\frac{\binom{\frac{k+h}{4}}{\frac{p}{4}}^4}{\binom{k+h}{p}} = \tilde{O}(1)$$

Algorithme *DecodeCollision*$_{SS}$

Algorithme 11 : *DecodeCollision*$_{SS}$

Entrées : H', y', $s' = H'y'^T$, p, h_1, h_2, r vecteur de longueur h tel que $r^{(2)} = 0$, $(S_{i,j})_{i,j=1,2}$

Sorties : une liste L de d'ensembles I', $|I'| = p$ tels que $\sum_{\ell \in I'} H'_\ell = s'$

$r_1 \leftarrow r$

$r_2 \leftarrow s' - r$

Initialiser les listes $L_{1,1}$, $L_{1,2}$, $L_{2,1}$, $L_{2,2}$

pour $i = 1, 2$ **faire**

  pour $I_{i,1} \in S_{i,1}$ **faire**

    $L_{i,1} \leftarrow L_{i,1} \cup (\sum_{\ell \in I_{i,1}} H'^{(1)}_\ell, I_{i,1})$

  pour $I_{i,2} \in S_{i,2}$ **faire**

    $L_{i,2} \leftarrow L_{i,2} \cup (r_i^{(1)} + \sum_{\ell \in I_{i,2}} H'^{(1)}_\ell, I_{i,2})$

  $L_i(r) \leftarrow L_{i,1} \overset{r_i}{\underset{h_1}{\bowtie}} L_{i,2}(*)$

$L \leftarrow L_1(r) \overset{s'}{\underset{h_2}{\bowtie}} L_2(r)$ $(**)$

retourner L

$(*)(L_{i,1} \overset{r_i}{\underset{h_1}{\bowtie}} L_{i,2} := ((\sum_{\ell \in I_i} H'^{(2)}_\ell, I_i) \mid I_i := I_{i,1} \sqcup I_{i,2}$ tel que $\sum_{\ell \in I_{i,1}} H'^{(1)}_\ell = r_i^{(1)} + \sum_{\ell \in I_{i,2}} H'^{(1)}_\ell)$

$(**) L_1(r) \overset{s'}{\underset{h_2}{\bowtie}} L_2(r) = (I' \mid I' := I_1 \sqcup I_2$ et $|I'| = p$ tel que $\sum_{\ell \in I_1} H'^{(2)}_\ell = s'^{(2)} + \sum_{\ell \in I_2} H'^{(2)}_\ell)$

Preuve de correction On fait l'hypothèse que le vecteur r est bon, c'est-à-dire qu'en cherchant dans la liste finale L on a la solution au problème.

- La liste L_1 contient les $I_1 = I_{1,1} \sqcup I_{1,2}$ tels que $r^{(1)} = r_1^{(1)} = \sum_{\ell \in I_{1,1}} H'^{(1)}_\ell + \sum_{\ell \in I_{1,2}} H'^{(1)}_\ell = \sum_{\ell \in I_1} H'^{(1)}_\ell$. Par conséquent $|I_1| = p/2$.
- La liste L_2 contient les $I_2 = I_{1,2} \sqcup I_{2,2}$ tels que $s'^{(1)} - r^{(1)} = r_2^{(1)} = \sum_{\ell \in I_{2,1}} H'^{(1)}_\ell + \sum_{\ell \in I_{2,2}} H'^{(1)}_\ell = \sum_{\ell \in I_2} H'^{(1)}_\ell$. Par conséquent $|I_2| = p/2$.
- La liste L contient les $I' = I_1 \sqcup I_2$ (donc $|I'| = p$) tels que :

$$\begin{cases} I_1 \in L_1, \text{ i.e. } \sum_{\ell \in I_1} H'^{(1)}_\ell = r^{(1)} \\ I_2 \in L_2, \text{ i.e. } \sum_{\ell \in I_2} H'^{(1)}_\ell = s'^{(1)} - r^{(1)} \\ \sum_{\ell \in I_1} H'^{(2)}_\ell + \sum_{\ell \in I_2} H'^{(2)}_\ell = s'^{(2)} \end{cases}$$

$$\Rightarrow \begin{cases} \sum_{\ell \in I_1} H'^{(1)}_\ell + \sum_{\ell \in I_2} H'^{(1)}_\ell = s'^{(1)} \\ \sum_{\ell \in I_1} H'^{(2)}_\ell + \sum_{\ell \in I_2} H'^{(2)}_\ell = s'^{(2)} \end{cases}$$

$\Rightarrow \sum_{\ell \in I'} H'_\ell = s'$ et $|I'| = p$.

Analyse de complexité

1. On parcourt $|S_{i,j}|$ éléments pour construire la liste $L_{i,j}$ pour $i, j = 1, 2$.
   - Complexité en temps : $\tilde{O}(|S_{i,j}|)$.
   - Complexité en espace : $|L_{i,j}| = |S_{i,j}| \Rightarrow \tilde{O}(|S_{i,j}|)$.

2. Pour construire L_1 et L_2, on fait un $Merge - Join$ sur des listes de taille $|L_{i,j}|$ contenant des vecteurs binaires de longueur h_1.
   - Complexité en temps : $\tilde{O}\left(\max\left(|L_{i,j}|, |L_{i,j}|^2 2^{-h_1}\right)\right)$.
   - Complexité en espace : $|L_i| = \tilde{O}\left(|L_{i,j}|^2 2^{-h_1}\right) \Rightarrow \tilde{O}\left(\max\left(|L_{i,j}|, |L_i|\right)\right)$.
3. Pour construire L, on fait un $Merge - Join$ sur des listes de taille $|L_i|$ contenant des vecteurs binaires de longueur h_2.
   - Complexité en temps :

$$\tilde{O}\left(\max\left(|L_i|, |L_i|^2 2^{-h_2}\right)\right) = \tilde{O}\left(\max\left(|L_{i,j}|^2 2^{-h_1}, |L_{i,j}|^4 2^{-2h_1-h_2}\right)\right)$$

   - Complexité en espace : on peut réutiliser de l'espace mémoire $\Rightarrow O(1)$.

Au final :

Complexité en temps

$$\tilde{O}\left(\max(|L_{i,j}|, |L_{i,j}|^2 2^{-h_1}, |L_{i,j}|^4 2^{-2h_1-h_2})\right) = \tilde{O}\left(\max\left|S_{i,j}|, |S_{i,j}|^2 2^{-h_1}, |S_{i,j}|^4 2^{-2h_1-h_2}\right)\right)$$

Complexité en espace

$$\tilde{O}(\max\left(|L_{i,j}|, |L_{i,j}|^2 2^{-h_1}\right)) = \tilde{O}(\max\left(|S_{i,j}|, |S_{i,j}|^2 2^{-h_1}\right))$$

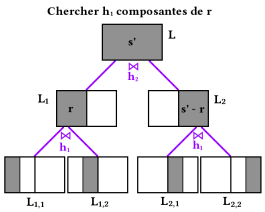
En choisissant $h_1 = h_2 = \frac{h}{2} = \log_2\left(\binom{k+h}{p}^{1/2}\right)$, on a toutes les listes de même taille moyenne égale à $|S_{i,j}| = \binom{\frac{k+h}{4}}{\frac{p}{4}}$. Cela donne la complexité en espace et la complexité en temps de l'algorithme $DecodeCollision_{SS}$.

Algorithme $Recherche_{SS}$ Cet algorithme fait la même chose que $Recherche_D$ avec en plus la recherche d'un bon vecteur r.

Algorithme 12 : $Recherche_{SS}$

Entrées : H, y, $s = Hy^T$, t, p,h_1, h_2,$(S_{i,j})_{i,j=1,2}$
Sorties : e tel que $y = mG + e$ et $w_H(e) = t$, $NULL$ si aucun tel vecteur n'a pu être trouvé

Préparer G',H',y' et calculer $s' = H'y'^T$
pour *tous les vecteurs* r *de longueur* h *tel que* $r^{(2)} = 0$ **faire**
    $L \leftarrow DecodeCollision_{SS}(H', y', s', p, h_1, h_2, r, (S_{i,j})_{i,j=1,2})$
    pour $I' \in L$ **faire**
        $e \leftarrow Dépoinçonner(I')$
        si $w_H(e) = t$ **alors**
            retourner e
retourner $NULL$



$Recherche_{SS}$ - illustration sous forme d'arbre
violet : recherche de collisions classique
$\bowtie$: Merge-Join

Complexité $Recherche_{SS}$ a la même complexité en espace que $DecodeCollision_{SS}$ et pour avoir sa complexité temporelle, il faut prendre en compte en plus la boucle sur 2^{h_1} éléments.

THÉORÈME 18. *Si H est une matrice aléatoire, l'algorithme de Shamir-Schroeppel a son exposant de complexité temporelle α_{SS} égale à α_D, mais sa complexité spatiale est la racine carrée de celle de Dumer.*

Démonstration.
Complexité en temps de $DecodeCollision_{SS}$

$$\tilde{O}\left(\max\left|S_{i,j}\right|, |S_{i,j}|^2 2^{-h_1}, |S_{i,j}|^4 2^{-2h_1-h_2}\right))$$

$$=$$

$$\tilde{O}\left(\max\left(\binom{k+h}{p}^{1/4}, \binom{k+h}{p}^{1/2} 2^{-h_1}, \binom{k+h}{p} 2^{-2h_1-h_2}\right)\right)$$

Complexité en espace de $DecodeCollision_{SS}$

$$\tilde{O}(\max\left(|S_{i,j}|, |S_{i,j}|^2 2^{-h_1}\right))$$

$$=$$

$$\tilde{O}(\max\left(\binom{k+h}{p}^{1/4}, \binom{k+h}{p}^{1/2} 2^{-h_1}\right))$$

Complexité de $Recherche_{SS}$
Dans $Recherche_{SS}$, on fait une boucle sur 2^{h_1} éléments. A l'intérieur de la boucle, on applique $DecodeCollision_{SS}$. Par conséquent, la complexité spatiale est la même que celle de $DecodeCollision_{SS}$ en réutilisant de l'espace mémoire, mais la complexité temporelle devient :

$$\tilde{O}\left(\max\left(\binom{k+h}{p}^{1/4} 2^{h_1}, \binom{k+h}{p}^{1/2}, \binom{k+h}{p} 2^{-h_1-h_2}\right)\right)$$

Donc on ne peut espérer faire mieux que $\binom{k+h}{p}^{1/2}$ et les deux autres termes y deviennent égaux en prenant $h_1 = h_2$ tels que $2^{h_1} = 2^{h_2} = \binom{k+h}{p}^{1/4}$. Alors puisque $h = h_1 + h_2$, on a $2^h = \sqrt{\binom{k+h}{p}}$ et la complexité temporelle sera en $\tilde{O}\left(\sqrt{\binom{k+h}{p}}\right)$. Par contre, la complexité spatiale sera en $\tilde{O}\left(\binom{k+h}{p}^{1/4}\right)$.

On a donc $T_{SS} = T_D$ mais $S_{SS} = \sqrt{S_D}$.

Complexité de l'algorithme
On trouve par conséquent $\alpha_{SS} = \alpha_D$ au niveau du temps d'exécution et $S_{SS} = \sqrt{S_D}$. □

4.1.2 Version quantique avec Grover

On note cet algorithme SSG.
On rappelle la notation $Grover(\varepsilon, f)$ pour la recherche d'un x tel que $f(x) = 1$ en $O\left(\frac{1}{\sqrt{\varepsilon}}\right)$ requêtes à la fonction f, où ε est la proportion des $x \in E$ tel que $f(x) = 1$.

On prend $Recherche_{SSG}$ comme $Recherche_{SS}$ avec la différence que l'on donne en argument un vecteur r au lieu de boucler sur toutes les valeurs possibles. Puis, on définit, étant donné H' et s', la fonction $\mathcal{G}_{SSG} : r \mapsto \{0, 1\}$ comme la fonction qui applique $Recherche_{SSG}$ à H', s' et r et renvoie 1 si la liste L renvoyée contient un vecteur d'erreur poinçonné e' correspondant au bon vecteur d'erreur e, et 0 sinon.

Alors, $Grover(2^{-h_1}, \mathcal{G}_{SSG})$ renvoie le bon vecteur r en $O(2^{h_1/2})$ requêtes à $\mathcal{G}_{SSG}$. Ensuite, on peut utiliser ce r comme argument de $Recherche_{SSG}$ pour trouver le bon vecteur d'erreur e.

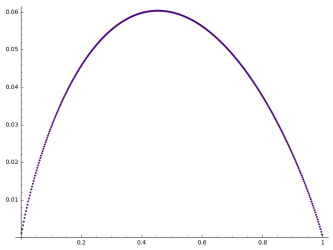
Complexité

THÉORÈME 19. *Si H est une matrice aléatoire, en notant $R := \frac{k}{n}$, $\tau := \frac{t}{n}$, $\pi := \frac{p}{n}$ et $\eta = \frac{h}{n}$, on a l'expression suivante pour l'exposant α_{SSG} de l'algorithme de Shamir-Schroeppel+Grover :*

$$\boxed{\alpha_{SSG}(R) = \min_{\pi,\eta}\left(H(\tau) - (1 - R - \eta)H(\frac{\tau - \pi}{1 - R - \eta}) - \frac{(R + \eta)}{4}H(\frac{\pi}{R + \eta})\right)}$$

Soumise aux contraintes :
$$\pi = (R + \eta)H^{-1}(\tfrac{2\eta}{(R+\eta)})$$
$$0 \leq \pi \leq \min(\tau, R + \eta)$$
$$0 \leq \eta \leq 1 - R - \tau + \pi \leq 1 - R$$

Et $\max_R(\alpha_{SSG}) = 0.0603509$ *pour* $R = 0.4537$, *valeur obtenue pour* $\pi \approx 1.162e - 09$ *et* $\eta \approx 1.742e - 08$



Courbe de α_{SSG} en fonction de R

Démonstration. Rappelons la complexité de $DecodeCollision_{SSG}$:

$$\tilde{O}\left(\max\left(\binom{k+h}{p}^{1/4}, \binom{k+h}{p}^{1/2}2^{-h_1}, \binom{k+h}{p}2^{-2h_1-h_2}\right)\right)$$

Cela donne la complexité suivante pour $Recherche_{SSG}$:

$$\tilde{O}\left(\max\left(\binom{k+h}{p}^{1/4}2^{h_1/2}, \binom{k+h}{p}^{1/2}2^{-h_1/2}, \binom{k+h}{p}2^{-3h_1/2-h_2}\right)\right)$$

Pour optimiser, on cherche à rendre tous les arguments égaux :

$$\binom{k+h}{p}^{1/4}2^{h_1/2} = \binom{k+h}{p}^{1/2}2^{-h_1/2} = \binom{k+h}{p}2^{-3h_1/2-h_2}$$

$$\Leftrightarrow$$

$$2^{h_1} = \binom{k+h}{p}^{1/4} = \binom{k+h}{p}^{3/4}2^{-h_1-h_2}$$

$$\Leftrightarrow$$

$$2^{h_1} = 2^{h_2} = \binom{k+h}{p}^{1/4}$$

$\Rightarrow 2^{h_1/2} = \binom{k+h}{p}^{1/8}$ et $2^h = \binom{k+h}{p}^{1/2}$
On obtient donc la complexité suivante pour $Recherche_{SSG}$:

$$\tilde{O}\left(\binom{k+h}{p}^{3/8}\right)$$

Donc la complexité totale de l'algorithme, $\sqrt{P_{SSG}^{-1}}T_{SSG}$ est :

$$\tilde{O}\left(\sqrt{\frac{\binom{n}{t}}{\binom{k+h}{p}\binom{n-k-h}{t-p}}\binom{k+h}{p}^{3/4}}\right)$$

$$\tilde{O}\left(\sqrt{\frac{\binom{n}{t}}{\binom{k+h}{p}^{1/4}\binom{n-k-h}{t-p}}}\right)$$

En d'autres termes, en posant $R := \frac{k}{n}$ et $\tau = \frac{t}{n}$ et $\pi = \frac{p}{n}, \pi := \frac{p}{n}$ et $\eta := \frac{h}{n}$:

$$\boxed{\alpha_{SSG}(R) = \min_{\pi,\eta}\left(\frac{H(\tau) - (1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta}) - \frac{(R+\eta)}{4}H(\frac{\pi}{R+\eta})}{2}\right)}$$

Soumise aux contraintes :

$$\pi = (R+\eta)H^{-1}(\tfrac{2\eta}{(R+\eta)})$$
$$0 \leq \pi \leq \min(\tau, R+\eta)$$
$$0 \leq \eta \leq 1-R-\tau+\pi \leq 1-R$$

□

La même remarque que pour DG sur la taille des paramètres optimaux s'applique dans ce cas, à savoir qu'en pratique ils sont nuls et que l'on retrouve l'algorithme de Prange quantique.

4.1.3 Les limites de l'algorithme de Grover

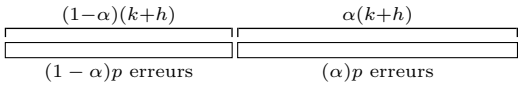
Voir appendice C.8 pour une discussion plus rigoureuse et détaillée de ces points.

Dans l'algorithme SSG, nous cherchons le bon vecteur r à l'aide de l'algorithme de Grover, alors que l'étape de recherche du bon vecteur d'erreur (construction d'une liste L par recherche de collision puis recherche dans cette liste) se fait de manière entièrement classique. Nous allons essayer d'appliquer l'algorithme de Grover à cette étape et voir pourquoi cela ne marche pas.

Tout d'abord, nous pouvons essayer de construire des listes $L_{i,j}$ puis, à partir de ces listes, construire les listes L_i auxquelles on appliquerait alors l'algorithme de Grover dans le but de trouver une collision. Le problème, là, sera celui d'un goulot d'étranglement : les listes L_i ne sont pas construites en temps constant, donc leur temps de construction (qui dépend de la taille des listes de base) l'emporte sur leur taille. Par conséquent, appliquer Grover ne permet pas de réduire la complexité.

L'autre solution consisterait à considérer des listes L_i de tailles inégales (par exemple, $|L_1| < |L_2|$). On construit alors la liste L_1 à partir de sous-listes $L_{1,1}$ et $L_{1,2}$, alors que la liste L_2 est engendré en temps constant. Dans ce cas, on peut appliquer $CollisionGrover$ pour trouver un vecteur d'erreur donnant une collision mais il faudrait, comme pour DG, imbriquer cette procédure de recherche au sein d'une autre qui cherche le bon vecteur d'erreur. Des calculs montrent alors que nous obtenons un algorithme meilleur que SSG mais pire que DG.

Démonstration. On suppose que les p erreurs de e' sont réparties de la manière suivante ($\frac{1}{2} \leq \alpha \leq 1$) :



Une première version consisterait à engendrer les listes $L_{i,j}$ comme dans Shamir-Schroeppel classique et appliquer ensuite $CollisionGrover$ dans le but de trouver une collision. Mais cela n'a aucun intérêt puisque le but de la recherche de collisions à l'aide de l'algorithme de Grover est de faire la recherche en la racine carrée de la taille d'une des listes L_i. Or le fait d'engendrer les listes L_i à partir des listes $L_{i,j}$ a la complexité temporelle $\tilde{O}\left(\max\left(|L_{i,j}|, |L_i|\right)\right)$ qui domine la complexité qu'aurait $CollisionGrover$.

On peut donc engendrer la liste L_1 qui correspond à la première partie du vecteur d'erreur ($1-\alpha$ positions d'erreurs parmi $(1-\alpha)(k+h)$ coordonnées) à partir de deux listes $L_{1,1} := \{(\sum_{k\in I_{1,1}} H'^{(1)}_k, I_{1,1}) \mid I_{1,1} \subset \{1, ..., \frac{(1-\alpha)(k+h)}{2}\}, |I_{1,1}| = \frac{1-\alpha}{2}p\}$ et $L_{1,2} := \{(\sum_{k\in I_{1,2}} H'^{(1)}_k, I_{1,2}) \mid I_{1,2} \subset \{\frac{(1-\alpha)(k+h)}{2} + 1, ..., (1-\alpha)(k+h)\}, |I_{1,2}| = \frac{1-\alpha}{2}p\}$. On aura alors en moyenne $|L_1| \approx \binom{\frac{(1-\alpha)(k+h)}{2}}{\frac{(1-\alpha)p}{2}}^2 2^{-h_1}$. Ensuite, on cherche à l'aide de $CollisionGrover$ un ensemble d'indices correspondant à αp erreurs parmi les $\alpha(k+h)$ dernières positions. La complexité temporelle totale

sera donc $\tilde{O}\left(\max\left(\binom{k+h}{p}^{(1-\alpha)}2^{-h_1}, \binom{\alpha(k+h)}{\alpha p}^{1/2}\right)\right) = \tilde{O}\left(\max\left(\binom{k+h}{p}^{(1-\alpha)}2^{-h_1}, \binom{k+h}{p}^{\alpha/2}\right)\right)$.

Pour simplifier, on prend le cas où les deux termes sont égaux. Cela arrive si $2^{h_1} \approx \binom{(k+h)}{p}^{(1-\frac{3}{2}\alpha)}$. Ceci est le temps moyen pour trouver un vecteur d'erreur. Il faudrait répéter cette étape $K = \max\left(1, \frac{\binom{k+h}{p}}{2^h}\right)$ fois afin d'obtenir le bon vecteur d'erreur. En faisant l'hypothèse $K = 1$ (i.e. $\binom{k+h}{p} = 2^h$) on obtient la complexité temporelle suivante pour tout l'algorithme :

$$\tilde{O}\left(\sqrt{\frac{\binom{n}{t}}{\binom{n-k-h}{t-p}\binom{k+h}{p}}}2^{h_1/2}\binom{k+h}{p}^{\alpha/2}\right)$$

$$\tilde{O}\left(\sqrt{\frac{\binom{n}{t}}{\binom{n-k-h}{t-p}\binom{k+h}{p}}\binom{k+h}{p}^{1-\frac{3}{2}\alpha+\alpha}}\right)$$

$$\tilde{O}\left(\sqrt{\frac{\binom{n}{t}}{\binom{n-k-h}{t-p}\binom{k+h}{p}^{\alpha/2}}}\right)$$

De plus, le fait que l'on devrait avoir $0 < h_1 < h$ impose $1 < 2^{h_1} < 2^h \Rightarrow 1 < \binom{(k+h)}{p}^{(1-\frac{3}{2}\alpha)} < \binom{k+h}{p} \Rightarrow 0 < 1 - \frac{3}{2}\alpha < 1 \Rightarrow 0 < \alpha < \frac{2}{3}$. Par conséquent, la complexité, bien qu'elle puisse être meilleure que celle de SSG, est pire que celle de DG.

Il y a en fait deux choses qui bloquent :

1. La taille des listes de base qui est souvent plus grande que celle obtenue pour les listes construites à partir de $Merge - Join$. Il y a donc un effet de goulot d'étranglement.
2. $CollisionGrover$ trouve un candidat parmi plusieurs et il faut soit choisir les paramètres de telle sorte qu'il y ait un seul candidat, soit imbriquer une autre recherche de Grover afin de trouver le bon candidat, ce qui augmente la complexité.

□

4.2 Algorithme de May, Meurer, Thomae classique

Comme nous l'avons vu, l'algorithme de Shamir-Schroeppel n'améliore pas la complexité temporelle asymptotique de l'algorithme de Dumer. C'est l'utilisation de la technique des représentations, une technique qui a été d'abord utilisée pour accélérer la résolution du subset sum, qui va permettre d'accélérer la recherche de collisions. L'algorithme de May, Meurer et Thomae ([12], [1]), que nous allons décrire dans cette section et que nous noterons MMT, a été le premier algorithme de décodage à utiliser cette technique.

Technique des représentations

1. Dans l'algorithme de Dumer comme dans l'algorithme de Shamir-Schroeppel, nous avions considéré l'ensemble I des indices des composantes non-nulles du vecteur d'erreur e et nous l'avions écrit sous la forme $I = I_1 \sqcup I_2$ avec $I_1 \subset \{1, ..., \frac{k+h}{2}\}$, $I_2 \subset \{\frac{k+h}{2} + 1, ..., k + h\}$. On appelle (I_1, I_2) une **représentation** de l'ensemble I. L'idée de l'algorithme de May, Meurer et Thomae est de relâcher les contraintes sur (I_1, I_2) afin d'avoir un nombre bien plus grand de représentations. En particulier, nous n'exigerons plus que $\frac{p}{2}$ erreurs soient réparties dans chaque moitié du vecteur d'erreur. En faisant cela, on augmente de façon importante la probabilité de tomber sur le bon vecteur d'erreur. Mais alors, un nouveau problème surgit : en augmentant le nombre de représentations, nous augmentons aussi la taille des listes. Nous verrons qu'il est possible de choisir les paramètres de sorte à avoir une seule représentation qui subsiste, ce qui diminue la taille des listes, mais garde l'avantage procuré par la multiplicité des représentations. Cela nous permettra de réduire aussi bien la complexité en l'espace qu'en temps.

2. L'algorithme de May, Meurer et Thomae repose donc sur la méthode des représentations pour améliorer les complexités en temps et en espace. Mais il y a aussi un deuxième niveau d'amélioration dans la mesure où nous nous servons d'une recherche de collisions pour trouver les bonnes représentations. Cela réintroduit des contraintes sur la forme du vecteur d'erreur, mais comme nous le verrons plus tard, ces contraintes font perdre au plus un facteur polynomial.

Description de l'algorithme MMT ressemble à SS, avec les différences que nous allons souligner.

On rappelle que l'ensemble $S := \{I \subset \{1, ..., h + k\} \mid |I| = p\}$ est isomorphe à l'ensemble des vecteurs d'erreur de poids p et de longueur $k + h$. On cherche à représenter l'ensemble I' des coordonnées non-nulles du vecteur d'erreur comme $I_1 \sqcup I_2$ avec $I_i \subset \{1, ..., k + h\}, |I_i| = \frac{p}{2}$ pour $i = 1, 2$. Il y a $\binom{p}{p/2}$ telles représentations.
On prend :
$\begin{cases} J_{1,1} = J_{2,1} = \{1, ..., \frac{k+h}{2}\} \\ J_{1,2} = J_{2,2} = \{\frac{k+h}{2} + 1, ..., k + h\} \end{cases}$
Alors $S_{i,j} = \{I_{i,j} \subset J_{i,j} \mid |I_{i,j}| = p/4\}$ et $|S_{i,j}| = \binom{\frac{k+h}{2}}{\frac{p}{4}} \approx \binom{k+h}{\frac{p}{2}}^{1/2}$.

On construit les listes $L_{i,j}$ à partir des $S_{i,j}$, puis on construit les listes L_i à partir des listes $L_{i,j}$ comme dans SS, en considérant l'union exclusive $\sqcup$ des ensembles d'indices. Par contre, pour construire L à partir des L_i, nous prendrons des ensembles d'indices $I' = I_1 \Delta I_2 := (I_1 \cup I_2) \setminus (I_1 \cap I_2)$, puisqu'il n'y a aucune garantie que les $I_i \in L_i$ soient disjoints.

Cette façon de construire les choses entraîne que I' ne s'écrit tout à fait comme $I_1 \sqcup I_2$ avec $I_i \subset \{1, ..., k + h\}, |I_i| = \frac{p}{2}$, car avec cette construction, les I_i s'écriront comme $I_{i,1} \sqcup I_{i,2}$. Or la proportion des représentations de cette forme

est :

$$\frac{\binom{p/2}{p/4}^2}{\binom{p}{p/2}} = \tilde{O}(1)$$

Donc on ne perd qu'un facteur polynomial en se restreignant aux représentations de cette forme.

Preuve de correction de $DecodeCollision_{MMT}$

— La liste L_1 contient les $I_1 = I_{1,1} \sqcup I_{1,2}$ tels que $r^{(1)} = r_1^{(1)} = \sum_{\ell \in I_{1,1}} H'^{(1)}_\ell + \sum_{\ell \in I_{1,2}} H'^{(1)}_\ell = \sum_{\ell \in I_1} H'^{(1)}_\ell$.

— La liste L_2 contient les $I_2 = I_{1,2} \sqcup I_{2,2}$ tels que $s'^{(1)} - r^{(1)} = r_2^{(1)} = \sum_{\ell \in I_{2,1}} H'^{(1)}_\ell + \sum_{\ell \in I_{2,2}} H'^{(1)}_\ell = \sum_{\ell \in I_2} H'^{(1)}_\ell$.

— On a $|I_i| = \frac{p}{2}$ pour $i = 1, 2$ par construction car $I_{i,1} \cap I_{i,2} = \emptyset$. De plus, $I' = I_1 \Delta I_2$ vérifie $|I'| \leq p$.

— La liste L contient les $I' = I_1 \Delta I_2$ tels que :

$$\begin{cases} I_1 \in L_1, \text{ i.e. } \sum_{\ell \in I_1} H'^{(1)}_\ell = r^{(1)} \\ I_2 \in L_2, \text{ i.e. } \sum_{\ell \in I_2} H'^{(1)}_\ell = s'^{(1)} - r^{(1)} \\ \sum_{\ell \in I_1} H'^{(2)}_\ell + \sum_{\ell \in I_2} H'^{(2)}_\ell = s'^{(2)} \\ |I'| = p, \text{ i.e. } I_1 \cap I_2 = \emptyset \end{cases}$$

$$\Rightarrow \begin{cases} \sum_{\ell \in I_1} H'^{(1)}_\ell + \sum_{\ell \in I_2} H'^{(1)}_\ell = s'^{(1)} \\ \sum_{\ell \in I_1} H'^{(2)}_\ell + \sum_{\ell \in I_2} H'^{(2)}_\ell = s'^{(2)} \\ |I'| = p \end{cases}$$

$$\Rightarrow \sum_{\ell \in I'} H'_\ell = s' \text{ et } |I'| = p.$$

Par conséquent la liste de sortie L est bien comme on souhaite.

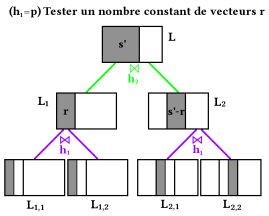
Représentations $Recherche_{MMT}$ consiste en une boucle sur 2^{h_1} éléments au maximum, au sein de laquelle l'étape la plus coûteuse est $SubmatrixMatch_{MMT}$ dont la complexité a été calculée ci-dessus.
Montrons qu'en choisissant $h_1 = \log_2 \binom{p/2}{p/4}^2 \approx \log_2 \binom{p}{p/2} \approx p$, il y aura en moyenne une représentation de chaque ensemble de positions d'erreurs, en particulier de celui qui est solution, qui "survivra" aux opérations de filtrage. En effet, le nombre de couples (I_1, I_2) s'écrivant $I_{i,1} \sqcup I_{i,2}, i = 1, 2$ avec $I_{i,j} \in L_{i,j}, j = 1, 2$ est $2\binom{p/2}{p/4}^2$ et la probabilité pour qu'un seul couple de ces ensembles soit tel que $\sum_{\ell \in I_1} H'^{(1)}_\ell = r^{(1)}$ et $\sum_{\ell \in I_2} H'^{(1)}_\ell = s'^{(1)} - r^{(1)}$ est 2^{-h_1}.

Par conséquent, la probabilité de "survie" d'une représentation est :

$$2\binom{p/2}{p/4}^2 2^{-h_1} \approx 2\frac{\binom{p}{p/2}}{2^{h_1}}$$

Ainsi il y a en moyenne $\tilde{O}\left(\frac{\binom{p}{p/2}}{2^{h_1}}\right) = \tilde{O}(1)$ représentations restantes de chaque ensemble de positions d'erreurs sous l'hypothèse $h_1 \approx \log_2 \binom{p}{p/2}$. Ceci est l'intérêt

de la méthode des représentations : en augmentant le nombre de façons de représenter un ensemble d'indices comme union de sous-ensembles, on arrive à réduire la taille des vecteurs cherchés par un facteur 2^{h_1}, ce qui réduit la taille des listes de recherche et améliore donc la complexité, tout garantissant qu'il suffit de répéter un nombre constant de fois le procédé avant de tomber sur la bonne réponse.



Recherche$_{MMT}$ - illustration sous forme d'arbre
violet : recherche de collisions classique
vert : technique des représentations
$\bowtie$: Merge-Join

Complexité

THÉORÈME 20 ([12]). *Si H est une matrice aléatoire, en notant $R := \frac{k}{n}$, $\tau := \frac{t}{n}$, $\pi := \frac{p}{n}$ et $\eta = \frac{h}{n}$, on a l'expression suivante pour l'exposant α_{MMT} de l'algorithme de MMT :*

$$\alpha_{MMT}(R) = \min_{\pi,\eta}\left(\beta(R,\pi,\eta) + \max_{i=1,2,3}\gamma_i(R,\eta,\pi)\right) \text{ avec :}$$

$$\boxed{\beta(R,\pi,\eta) = H(\tau) - (1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta}) - (R+\eta)H(\frac{\pi}{R+\eta})}$$

$$\boxed{\gamma_1(R,\eta,\pi) = \frac{R+\eta}{2}H(\frac{\pi}{2(R+\eta)})}$$

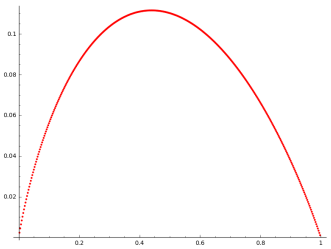
$$\boxed{\gamma_2(R,\eta,\pi) = (R+\eta)H(\frac{\pi}{2(R+\eta)}) - \pi}$$

$$\boxed{\gamma_3(R,\eta,\pi) = 2(R+\eta)H(\frac{\pi}{2(R+\eta)}) - \pi - \eta}$$

Soumise aux contraintes :

$$0 \leq \pi \leq \min(\tau,\eta) \leq R+\eta$$
$$0 \leq \eta \leq 1-R-\tau+\pi \leq 1-R$$

Et $\max_R(\alpha_{MMT}) = 0.1114837$ *pour* $R = 0.4397$, *valeur obtenue pour* $\pi \approx 0.021447363$ *et* $\eta \approx 0.0752051973$



Courbe de α_{MMT} *en fonction de* R

Démonstration.
Complexité en temps de $DecodeCollision_{MMT}$

$$\tilde{O}\left(\max\left|S_{i,j}\right|, |S_{i,j}|^2 2^{-h_1}, |S_{i,j}|^4 2^{-2h_1-h_2}\right))$$

$$=$$

$$\tilde{O}\left(\max\left(\binom{k+h}{\frac{p}{2}}^{1/2}, \binom{k+h}{\frac{p}{2}} 2^{-h_1}, \binom{k+h}{\frac{p}{2}}^2 2^{-2h_1-h_2}\right)\right)$$

Complexité en espace de $DecodeCollision_{MMT}$

$$\tilde{O}(\max\left(|S_{i,j}|, |S_{i,j}|^2 2^{-h_1}\right))$$

$$=$$

$$\tilde{O}(\max\left(\binom{k+h}{\frac{p}{2}}^{1/2}, \binom{k+h}{\frac{p}{2}} 2^{-h_1}\right))$$

Complexité de $Recherche_{MMT}$

1. $P_{MMT} = \frac{\binom{k+h}{p}\binom{n-k-h}{t-p}}{\binom{n}{t}}$
2. Calculons T_{MMT}, le temps d'exécution de $Recherche_{MMT}$.
   On utilise la technique des représentations en prenant $h_1 \approx \log_2 \binom{p}{p/2} \approx p$ (avec $h_2 = h - h_1$). Alors, la complexité en temps de $Recherche_{MMT}$ est la même que celle de $DecodeCollision_{MMT}$, à savoir :

$$T_{MMT} = \tilde{O}\left(\max\left(\binom{k+h}{\frac{p}{2}}^{1/2}, \binom{k+h}{\frac{p}{2}} 2^{-p}, \binom{k+h}{\frac{p}{2}}^2 2^{-2p-(h-p)}\right)\right)$$

On en déduit la complexité de l'algorithme $P_{MMT}^{-1} T_{MMT}$:

$$\tilde{O}\left(\frac{\binom{n}{t}}{\binom{k+h}{p}\binom{n-k-h}{t-p}} \max\left(\binom{k+h}{\frac{p}{2}}^{1/2}, \binom{k+h}{\frac{p}{2}} 2^{-p}, \binom{k+h}{\frac{p}{2}}^2 2^{-h-p}\right)\right)$$

Par conséquent, $\alpha_{MMT}(R) = \min_{\pi,\eta}\left(\beta(R,\tau,\pi,\eta) + \max_{i=1,2,3}\gamma_i(R,\eta,\pi)\right)$ avec :

$$\boxed{\beta(R,\tau,\pi,\eta) = H(\tau) - (1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta}) - (R+\eta)H(\frac{\pi}{R+\eta})}$$

$$\boxed{\gamma_1(R,\eta,\pi) = \frac{R+\eta}{2}H(\frac{\pi}{2(R+\eta)})}$$

$$\boxed{\gamma_2(R,\eta,\pi) = (R+\eta)H(\frac{\pi}{2(R+\eta)}) - \pi}$$

$$\boxed{\gamma_3(R,\eta,\pi) = 2(R+\eta)H(\frac{\pi}{2(R+\eta)}) - \pi - \eta}$$

Soumise aux contraintes :
$0 \leq \pi \leq \min(\tau,\eta) \leq R+\eta$
$0 \leq \eta \leq 1-R-\tau+\pi \leq 1-R$

□

4.2.1 Représentations VS collision

Nous pouvons faire le constat général suivant au sujet de la technique des représentations telle qu'elle est utilisée dans MMT (voir par exemple [1], chapitre 10, section 1.2).

1. Si $h_1 < \log_2 \binom{p}{p/2}$, chaque ensemble de positions d'erreurs possible est représenté de plusieurs manières.
2. Si $h_1 > \log_2 \binom{p}{p/2}$, tous les ensembles de positions d'erreurs possibles n'auront pas de représentation mais s'ils ont une représentation, celle-ci est unique. Autrement dit, quelque soit le vecteur r avec $r^{(2)} = 0$ que l'on prend, l'égalité sur p des h_1 composantes, par exemple les p premières composantes, est garantie par la technique des représentations. Il faut chercher de manière exhaustive une configuration du vecteur r sur les $h_1 - p$ composantes restantes qui fonctionne.

   En effet, le nombre total d'ensembles I qu'il est possible d'obtenir à partir des listes de base et qui vérifient la condition d'égalité au syndrome est $|L_{i,j}|^4 2^{-h}$ éléments. Cela devrait être le nombre total de valeurs que l'algorithme aura mis dans la liste finale à la fin de son déroulement.
   Or, en pratique, cette liste contient $\binom{p}{\frac{p}{2}}|L_{i,j}|^4 2^{-h-h_1}$ pour chaque vecteur r. Si l'on note 2^δ le nombre de vecteurs r sur lesquels l'algorithme se déroule avant de terminer, il y aura un total de $\approx 2^\delta 2^p |L_{i,j}|^4 2^{-h-h_1}$ valeurs passées dans la liste finale.

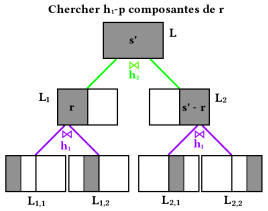
   On doit donc choisir δ de telle sorte à avoir l'égalité :

$$2^\delta 2^p |L_{i,j}|^4 2^{-h-h_1} = |L_{i,j}|^4 2^{-h}$$

$$\Leftrightarrow$$

$$2^\delta = 2^{h_1 - p}$$

   Dans ce deuxième cas, il est possible de faire la chose suivante : on choisit, comme dans SS, les $L_{i,j}$ et L_i et L de taille égale, ce qui revient à choisir : $2^{h_1} = 2^{h_2} = |L_{i,j}| (h = \frac{h_1}{2} = \frac{h_2}{2} = \log_2(|L_{i,j}|)$.



$Recherche_{MMT}$ dans le cas $h_1 > p$ - illustration sous forme d'arbre
violet : recherche de collisions classique
vert : technique des représentations
$\bowtie$: Merge-Join

4.3 ISD avancé avec Quantum Walk

Les algorithmes donnés dans cette section sont inspirés de celui donné dans [4] (section 4) pour le problème du subset sum. Il s'agit de résoudre le problème

du décodage par collision à l'aide d'une marche aléatoire quantique. En effet, nous pouvons reformuler le problème que cherchent à résoudre les fonctions $Recherche_{SS}$ et $Recherche_{MMT}$ comme le problème de recherche de 4-uplet suivant :

Problème 7 (Décodage comme recherche de 4-uplet). *Soit* $f : S_{1,1} \times S_{1,2} \times S_{2,1} \times S_{2,2} \to \mathbb{F}_2^h$, $f : (I_{1,1}, I_{1,2}, I_{2,1}, I_{2,2}) \mapsto \sum_{i,j=1,2} \sum_{\ell \in I_{i,j}} H'_\ell + s$. *Trouver un 4-uplet* $(I_{1,1}, I_{1,2}, I_{2,1}, I_{2,2})$ *tel que* $f(I_{1,1}, I_{1,2}, I_{2,1}, I_{2,2}) = 0$ *et* $|\cup_{i,j=1,2} I_{i,j}| = p$.

Or, nous avons vu qu'il peut être intéressant d'utiliser une marche aléatoire pour résoudre des problèmes qui admettent ce type de formulations. C'est effectivement le cas : comme nous le verrons, si certaines conditions sont réunies, la marche aléatoire quantique permet de contourner le problème du goulot d'étranglement qui se posait avec l'algorithme de Grover. De plus, la recherche du bon vecteur d'erreur parmi les éléments de la liste finale peut se faire au sein de la marche aléatoire, ce qui évite le recours à des recherches imbriquées qui augmentent la complexité comme dans *DG*.

4.3.1 Squelette d'un algorithme ISD avancé avec Quantum Walk

La marche aléatoire et son graphe La solution cherchée est un 4-uplet $(I_{1,1}, I_{1,2}, I_{2,1}, I_{2,2}) \in S_{1,1} \times S_{1,2} \times S_{2,1} \times S_{2,2}$. Pour éviter de parcourir tout l'espace de recherche, on va considérer quatre sous-ensembles $T_{i,j}$ des ensembles $S_{i,j}$, tous de même taille $T \leq |S_{i,j}|$.

On peut clairement mettre en bijection chaque $T_{i,j}$ avec un sous-ensemble de taille T d'un ensemble de taille $|S_{i,j}|$. Par exemple, si nous avons des bijections entre les $T_{1,1}$ et les sous-ensembles de taille T de $\{1, ..., |S_{i,j}|\}$, ... , les $T_{2,2}$ et les sous-ensembles de taille T de $\{3|S_{i,j}| + 1, ..., 4|S_{i,j}|\}$, on peut les utiliser pour mettre en bijection le 4-uplet $(T_{1,1}, T_{1,2}, T_{2,1}, T_{2,2})$ et une partie des sous-ensembles de taille $4T$ de $\{1, ..., 4|S_{i,j}|\}$.

À l'aide de ces identifications, on considère donc une marche aléatoire sur le graphe $J(4|S_{i,j}|, 4T)$ pour trouver un 4-uplet convenable pour une fonction f_A que l'on définira ci-dessous. Il s'agit de la marche $QW(J(4|S_{i,j}|, 4T), f_A)$ en reprenant une notation introduite dans la section sur l'algorithmique quantique.

On a besoin de définir ou expliciter les éléments suivants :

1. Le trou spectral $\delta = \frac{4|S_{i,j}|}{4T(4(|S_{i,j}|-4T)} = \Omega\left(T^{-1}\right)$.
2. La fonction f_A à calculer et la structure de données associée.
3. L'ensemble des éléments marqués M_{f_A}, ce qui donnera la proportion ε des éléments marqués.
4. Le coût S de Setup.
5. Le coût C de Check.
6. Le coût U de Update.

La fonction f_A et la structure de données associée La fonction f_A a à peu de choses près la même structure que les fonctions $DecodeCollision_A$ des algorithmes ISD avancés.

En effet, elle calcule des structures de données ordonnées contenant des couples "$(\sum_J, J)$", où J désigné un ensemble d'indices, sur trois niveaux, où celles du niveau le plus bas sont générées directement et celles du niveau suivant sont obtenues à partir de celles du niveau précédent en faisant un $Merge - Join$ et en exigeant une égalité à un vecteur d'abord sur h_1 coordonnées et ensuite sur les h_2 coordonnées restantes.
Ainsi, la fonction f_A créé quatre structures de données $D_{f_{i,j}}$, $i = 1, 2$ au dernier niveau, puis deux au niveau suivant, D_{f_1} et D_{f_2}. Ces structures de données sont ordonnées en fonction du premier élément du couple. Enfin, au niveau final, une dernière structure de données D est créée à partir de D_{f_1} et D_{f_2}. Dans cette structure de données, toutes les sommes sont égales et donc on peut se contenter de ne stocker que l'ensemble d'indices J.

Ce qui change des fonctions $DecodeCollision_A$, c'est que tout d'abord, f_A stocke en parallèle les ensembles d'indices séparément à chaque niveau dans des structures de données ordonnées. Elle créé ainsi quatre structures de données $D_{I_{i,j}}$, $i, j = 1, 2$ au niveau le plus bas, puis deux, D_{I_1} et D_{I_2} au niveau suivant.

Par conséquent, la structure de données associée à la fonction f_A comporte ainsi treize sous-structures de données $(D_{I_{i,j}})_{i,j=1,2}, (D_{f_{i,j}})_{i,j=1,2}, (D_{I_i})_{i=1,2}, (D_{f_i})_{i=1,2}, D$.

Enfin, f_A diffère des fonctions $DecodeCollision_A$ dans la mesure où, après avoir crée la structure de données finale D, elle cherche s'il existe $I' \in D$ tel que, en y appliquant l'opération *Dépoinçonner*, on tombe sur un vecteur d'erreur e de poids t. Elle renvoie 0 si c'est le cas, 1 sinon.

On va donner cette fonction sous forme de pseudo-code :

Algorithme 13 : f_A

Entrées : $T_{i,j}$ pour $i, j = 1, 2$, r vecteur de longueur h tel que $r^{(2)} = 0$, H', y', $s' = H'y'^T$, p, h_1, h_2
Sorties : une liste L de d'ensembles I', $|I'| = p$ tels que $\sum_{\ell \in I'} H'_\ell = s'$

1 $r_1 \leftarrow r$
2 $r_2 \leftarrow s' - r$
3 Initialiser et remplir les $D_{I_{i,j}}$ et $D_{f_{i,j}}$, $i, j = 1, 2$
4 **pour** $i = 1, 2$ **faire**
5 $\quad D_{f_i}(r) \leftarrow D_{f_{i,1}} \overset{r_i}{\underset{h_1}{\bowtie}} D_{f_{i,2}}$
6 $\quad D_{I_i}(r) \leftarrow D_{I_{i,1}} \overset{r_i}{\underset{h_1}{\bowtie}} D_{I_{i,2}}$
7 $D \leftarrow D_{f_1}(r) \overset{s'}{\underset{h_2, |I'|=p}{\bowtie}} D_{f_2}(r)$
8 **si** *on peut trouver e de poids t à partir de D* **alors**
9 $\quad$ **retourner** 0
10 **sinon**
11 $\quad$ **retourner** 1

La proportion ε des éléments marqués On définit $M_{f_A} :=$ "$\{I' \mid w_H(e) = t$ où $e = Dépoinconner(I')\}$".

Sous l'hypothèse d'avoir choisi le bon vecteur r, la structure de données D contient des ensembles I' qui s'écrivent $I_1 \square I_2 = (I_{1,1} \sqcup I_{1,2}) \square (I_{2,1} \sqcup I_{2,2})$ avec $\square \in \{\sqcup, \Delta\}$ en fonction de l'algorithme. La probabilité qu'un seul des $T_{i,j}$ contienne l'ensemble $I_{i,j}$ en question est $\frac{T}{|S_{i,j}|}$. Comme les $T_{i,j}$ sont indépendants, la probabilité de succès ε est donc $\left(\frac{T}{|S_{i,j}|}\right)^4$.

Update

THÉORÈME 21. *On choisit les treize sous-structures de données de sorte à avoir un temps d'exécution constant ou au pire en* $\log(T)$ *pour les opérations suivantes :*

1. *Insertion d'un élément dans la structure de données.*
2. *Suppression d'un élément de la structure de données.*
3. *Recherche d'un élément dans la structure de données.*

De plus, on prend $2^{h_1} = 2^{h_2} = T \Leftrightarrow h_1 = h_2 = \log_2(T)$.
Alors la mise à jour de la structure de données s'effectue en temps $O(\log(T)) = \tilde{O}(1)$.

EXEMPLE : Nous pouvons suivre le choix de [4] et utiliser des arbres radix, ce qui permettra de faire toutes ces opérations en $O(\log(T))$.

Démonstration. On remarque que pour un $I_{i,j}$ particulier, il est possible d'écrire $I_i := I_{i,j} \sqcup I_{i,2j \bmod 3}$ de T manières différentes (car il y a T choix de $I_{i,2j \bmod 3}$). Or, les seuls I_i qui sont gardés dans $D_{I_{i,2}}$ sont ceux qui vérifient en plus une condition d'égalité à un vecteur sur les h_1 premières coordonnées. Par conséquent, il y a en moyenne $O(\frac{T}{2^{h_1}})$ ensembles I_i faisant intervenir un $I_{i,j}$ particulier. Par conséquent, en prenant $h_1 = \log_2(T)$, sans perturber la probabilité d'aboutissement de l'algorithme de manière significative, il y aura en moyenne un nombre constant de I_i faisant intervenir un $I_{i,j}$ particulier. De même, en prenant $h_2 = \log_2(T)$, il vaut pour chaque I_i que $I' := I_i \sqcup I_{2i \bmod 3}$ s'écrit de T manières différentes dont en moyenne $O(\frac{T}{2^{h_2}}) = O(1)$ véri fient la condition d'égalité au vecteur s' sur les h_2 dernières coordonnées. Lors d'une étape de la marche aléatoire, un ensemble $I^U_{i,j}$ appartenant à l'un des $T_{i,j}$ est remplacé par un ensemble $J^U_{i,j}$. Pour simplifier la notation et sans perte de généralité, nous allons supposer que c'est $I^U_{1,1} \in T_{1,1}$ qui a été remplacé par $J^U_{1,1}$. Les modifications à apporter dans la structure de données sont alors les suivantes :

Premièrement :

1. Trouver et supprimer $I^U_{1,1}$ de $D_{I_{1,1}}$ (en temps $O(\log(T))$).
2. Calculer $\sum_{\ell \in I^U_{1,1}} H'^{(1)}_\ell$ et supprimer $(\sum_{\ell \in I^U_{1,1}} H'^{(1)}_\ell, I^U_{1,1})$ de $D_{f_{1,1}}$ (en temps $O(\log(T))$).
3. Supprimer tous les I^U_1 s'écrivant $I^U_{1,1} \sqcup I_{1,2}$ de D_{I_1} (d'après ci-dessus, il y en a en moyenne un nombre constant, donc cette étape s'effectue en temps $O(\log(T))$).

4. Calculer $\sum_{\ell \in I_1^U} H_\ell'^{(1)}$ et trouver puis supprimer $(\sum_{\ell \in I_1^U} H_\ell'^{(1)}, I_1^U)$ de D_{f_1} (en temps $O(\log(T))$).
5. Supprimer tous les I' s'écrivant $I_1^U \square I_2$ de D (en temps $O(\log(T))$ pour la même raison).

Par conséquent, la suppression s'effectue en un temps borné par $4\log(T)$.
Ensuite :

1. Ajouter $J_{1,1}^U$ dans $D_{I_{1,1}}$ (en temps $O(\log(T))$).
2. Ajouter $(\sum_{\ell \in J_{1,1}^U} H_\ell'^{(1)}, J_{1,1}^U)$ de $D_{f_{1,1}}$ (en temps $O(\log(T))$).
3. Pour un nombre constant d'ensembles $I_{1,2} \in D_{I_{1,2}}$, poser $J_1^U := J_{1,1}^U \sqcup I_{1,2}$, calculer $\sum_{\ell \in I_1} H_\ell'^{(2)}$ et ajouter $(\sum_{\ell \in I_1} H_\ell'^{(2)}, J_1^U)$ dans D_{f_1}, puis ajouter J_1^U dans D_{I_1} si la condition sur la somme est vérifiée. (en temps $O(\log(T))$).
4. Pour un nombre constant d'ensembles $I_2 \in D_{I_2}$, poser $J' := J_1^U \square I_2$ et ajouter J' dans D si la condition sur la somme est vérifiée (en temps $O(\log(T))$).

Par conséquent, l'ajout s'effectue en un temps borné par $4\log(T)$.

Par conséquent, la mise à jour de la structure de données s'effectue en temps $O(\log(T)) = \tilde{O}(1)$. □

Setup Pour mettre en place l'état initial avec cette structure de données il faut évaluer f_A, on calcule donc son coût en reprenant en partie les calculs qui ont été faits pour $DecodeCollision_{SS}$ et en remplaçant $|S_{i,j}|$ par T.

$$\tilde{O}\left(\max(|D_{f_{i,j}}|, |D_{f_{i,j}}|^2 2^{-h_1}, |D_{f_{i,j}}|^4 2^{-2h_1-h_2})\right) = \tilde{O}\left(\max\left(T, T^2 2^{-h_1}, T^4 2^{-2h_1-h_2}\right)\right)$$

Or, puisque nous avons choisi $h_1 \approx \log_2(T)$ afin de garantir un coût de mise à jour en $\tilde{O}(1)$, cette expression est égale à :
$\tilde{O}\left(\max\left(T, T^2 2^{-h_2}\right)\right) = \tilde{O}\left(T\right)$ en prenant $h_2 = h_1 \approx \log_2(T)$ (en effet, cela ne sert à rien de choisir h_2 plus grand, et en choisissant h_2 plus petit, on aurait un goulot d'étranglement).

En termes de complexité en espace, f_A coûte :

$$\tilde{O}(\max\left(|D_{f_{i,j}}|, |D_{f_{i,j}}|^2 2^{-h_1}\right)) = \tilde{O}(\max\left(T, T^2 2^{-h_1}\right)) = T$$

Check Hormis la première étape de la marche aléatoire, où il faut chercher un élément $I' \in D$ qui donne un vecteur d'erreur e de poids t après application de *Dépoinçonner* (ce qui peut se faire en temps $\tilde{O}(\sqrt{T})$ avec l'algorithme de Grover), l'étape de vérification coûte en moyenne $\tilde{O}(1)$ puisqu'il suffit de vérifier pour un nombre constant de nouveaux ensembles d'indices ajoutés si l'un d'entre eux marche bien.

Complexité Sous l'hypothèse d'avoir le bon vecteur r, la marche aléatoire trouve le bon vecteur d'erreur e en temps :

$$\begin{aligned} S + \frac{1}{\sqrt{\varepsilon}}(C + \frac{1}{\sqrt{\delta}}U) &= O(T) + |S_{i,j}|^2 T^{-2}(\tilde{O}(1) + O(\sqrt{T})O(1)) \\ &= \max\left(T, |S_{i,j}|^2 T^{-3/2}\right) \end{aligned}$$

Il faut ajouter à ce coût le coût de la recherche du bon vecteur r parmi R_A possibilités, ce qui est faisable en temps $O\left(\sqrt{R_A}\right)$ avec l'algorithme de Grover. Ainsi le coût total sera $\tilde{O}\left(\sqrt{R_A}\max\left(T, |S_{i,j}|^2 T^{-3/2}\right)\right)$.

Récapitulation Il reste à résumer ce qui a été dit sous forme d'un squelette pour la fonction $Recherche_A$. On rappelle pour cela la notation $Grover(\varepsilon, f)$ pour la recherche d'un x tel que $f(x) = 1$ avec l'algorithme de Grover, et on rappelle que l'on a défini la marche $QW(J(4|S_{i,j}|, 4T), f_A)$ ci-dessus.

Puis, on définit la fonction $\mathcal{QW}_A : r \mapsto \{0, 1\}$ comme la fonction qui renvoie 1 ssi $QW(J(4|S_{i,j}|, 4T), f_A)$ trouve un bon 4-uplet lorsque f_A prend en argument r.

On peut maintenant donner le squelette de la fonction $Recherche_A$:

Algorithme 14 : Squelette $Recherche_A$ avec Quantum Walk

Entrées : H, y, $s = Hy^T$, t, p
Sorties : e tel que $s = mG + e$ et $w_H(e) = t$, $NULL$ si aucun tel vecteur n'a pu être trouvé

1 Préparer G',H',y' et calculer $s' = H'y'^T$
2 $|\rho\rangle \leftarrow Grover(R_A^{-1}, \mathcal{QW}_A)$
3 $r \leftarrow Mesure(|\rho\rangle)$
4 $I' = (I_{1,1}, I_{1,2}, I_{2,1}, I_{2,2}) \leftarrow QW(J(4|S_{i,j}|, 4T), f_A(..., r, ...))$
5 $e \leftarrow Dépoinçonner(I')$
6 **retourner** e

4.3.2 Algorithme de Shamir-Schroeppel quantique avec Quantum Walk

On notera cet algorithme $SSQW$.

Dans l'algorithme SS, nous avions $|S_{i,j}| \approx \binom{k+h}{p}^{1/4}$. Nous allons donc considérer ici des sous-ensembles $T_{i,j}$ de $S_{i,j}$ de taille $T \leq \binom{k+h}{p}^{1/4}$ et donc une marche sur $J\left(4\binom{k+h}{p}^{1/4}, 4T\right)$.

Le vecteur r cherché comporte h_1 composantes non-nulles, or $h_1 = \log_2(T)$, i.e. $2^{h_1} = T$. Par conséquent $R_A = T$ et donc la marche aléatoire a la complexité suivante :

$$\tilde{O}\left(\sqrt{R_A}\max\left(T, |S_{i,j}|^2 T^{-3/2}\right)\right) = \tilde{O}\left(\max\left(T^{3/2}, \binom{k+h}{p}^{1/2} T^{-1}\right)\right)$$

Les deux termes de la complexité sont égaux si l'on prend $T = \binom{k+h}{p}^{1/5}$ auquel cas on obtient une complexité temporelle T_{SSQW} en $\tilde{O}\left(\binom{k+h}{p}^{3/10}\right)$.

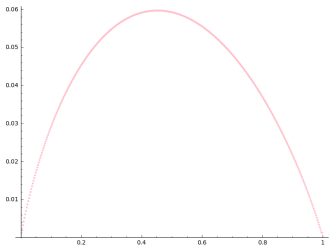
THÉORÈME 22. *Si H est une matrice aléatoire, en notant $R := \frac{k}{n}$, $\tau := \frac{t}{n}$, $\pi := \frac{p}{n}$ et $\eta = \frac{h}{n}$, on a l'expression suivante pour l'exposant α_{SSQW} :*

$$\boxed{\alpha_{SSQW}(R) = \min_{\pi,\eta}\left(\frac{H(\tau) - (1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta}) - \frac{2(R+\eta)}{5}H(\frac{\pi}{R+\eta})}{2}\right)}$$

Soumise aux contraintes :

$$\pi = (R+\eta)H^{-1}(\tfrac{5\eta}{2(R+\eta)})$$
$$0 \leq \pi \leq \min(\tau, R+\eta)$$
$$0 \leq \eta \leq 1 - R - \tau + \pi \leq 1 - R$$

Et $\max_R(\alpha_{SSQW}) = 0.0596979$ *pour* $R = 0.4537$, *valeur obtenue pour* $\pi \approx 0.003675$ *et* $\eta \approx 0.012385$



Courbe de α_{SSQW} en fonction de R

Démonstration. On a $T = \binom{k+h}{p}^{1/5}$ et $h_1 = h_2 = \frac{h}{2} = T$.

- $P_{SSQW} = \frac{\binom{k+h}{p}\binom{n-k-h}{t-p}}{\binom{n}{t}}$
- $T_{SSQW} = \tilde{O}\left(T^{3/2}\right) = \tilde{O}\left(\binom{k+h}{p}^{3/10}\right)$
- $2^h = \binom{k+h}{p}^{2/5}$

On en déduit la complexité totale de l'algorithme, $\sqrt{P_{SSQW}}T_{SSQW}$:

$$\tilde{O}\left(\sqrt{\frac{\binom{n}{t}}{\binom{k+h}{p}\binom{n-k-h}{t-p}}}\binom{k+h}{p}^{3/10}\right)$$
$$=$$
$$\tilde{O}\left(\sqrt{\frac{\binom{n}{t}}{\binom{k+h}{p}\binom{n-k-h}{t-p}}}\binom{k+h}{p}^{6/10}\right)$$
$$=$$
$$\tilde{O}\left(\sqrt{\frac{\binom{n}{t}}{\binom{k+h}{p}^{2/5}\binom{n-k-h}{t-p}}}\right)$$

Par conséquent :

$$\boxed{\alpha_{SSQW}(R) = \min_{\pi,\eta}\left(\frac{H(\tau) - (1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta}) - \frac{2(R+\eta)}{5}H(\frac{\pi}{R+\eta})}{2}\right)}$$

Soumise aux contraintes :

$$\pi = (R+\eta)H^{-1}(\tfrac{5\eta}{2(R+\eta)})$$
$$0 \leq \pi \leq \min(\tau, R+\eta)$$
$$0 \leq \eta \leq 1-R-\tau+\pi \leq 1-R$$

□

4.3.3 Algorithme de May, Meurer et Thomae quantique avec Quantum Walk

Approche directe Une adaptation directe de MMT ne marchera pas. En effet, dans MMT, le choix de h_1 et de h_2 est imposé, ce qui perturbe l'équilibre obtenu précédemment. Plus précisément :
Pour avoir exactement une représentation de chaque ensemble de positions d'erreur, il faut prendre $h_1 = p$. D'autre part, pour garder l'équilibre qui fait fonctionner l'algorithme, il faut prendre $h_1 = \log_2(T)$. Par conséquent, on devrait avoir $p = \log_2(T)$, or on devrait avoir $p < t$, mais dans les cas qui nous intéressent on a $\log_2(T) > t$, par conséquent la totalité des conditions souhaitées sur les paramètres ne peuvent pas être vérifiées.

Pour pallier à ce problème, nous allons opter pour un algorithme hybride entre MMT et SS.

Algorithme hybride entre SS et MMT Cet algorithme, que l'on note $MMTQW$, utilise la technique des représentations. Il s'agit en effet de prendre le cas de figure $h_1 > p$ expliquée dans la partie sur la différence entre la technique des représentations et la recherche de collisions et d'y appliquer la marche aléatoire quantique.

On rappelle que dans ce cas, les conditions d'équilibre nécessaires sont bien remplies, et il y a une garantie d'égalité pour p composantes du vecteur r par la technique des représentations, et il faut chercher exhaustivement les $h_1 - p$ composantes restantes.

Ainsi, on a $|S_{i,j}| \approx \binom{k+h}{\frac{p}{2}}^{1/2}$. Nous allons donc considérer ici des sous-ensembles $T_{i,j}$ de $S_{i,j}$ de taille $T \leq \binom{k+h}{\frac{p}{2}}^{1/2}$ et donc une marche sur $J\left(4\binom{k+h}{\frac{p}{2}}^{1/2}, 4T\right)$.

Comme on cherche exhaustivement $h_1 - p$ composantes du vecteur r et que $2^{h_1} = T$, on a $R_{MMTQW} = \frac{T}{2^p}$ et donc la marche aléatoire a la complexité suivante :

$$\tilde{O}\left(\sqrt{R_{MMTQW}}\max\left(T, |S_{i,j}|^2 T^{-3/2}\right)\right) = \tilde{O}\left(\frac{1}{2^{p/2}}\max\left(T^{3/2}, \binom{k+h}{\frac{p}{2}}T^{-1}\right)\right)$$

Les deux termes de la complexité sont égaux si l'on prend $2^{h/2} = T = \binom{k+h}{\frac{p}{2}}^{2/5}$ auquel cas on obtient une complexité temporelle T_{MMTQW} en $\tilde{O}\left(\frac{\binom{k+h}{\frac{p}{2}}^{3/5}}{2^{p/2}}\right)$.

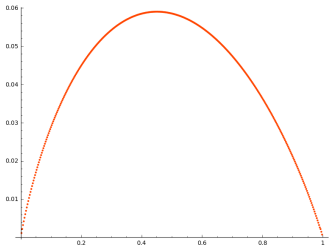
THÉORÈME 23. *Si H est une matrice aléatoire, en notant $R := \frac{k}{n}$, $\tau := \frac{t}{n}$, $\pi := \frac{p}{n}$ et $\eta = \frac{h}{n}$, on a l'expression suivante pour l'exposant α_{MMTQW} :*

$$\boxed{\alpha_{MMTQW}(R) = \min_{\pi,\eta}\left(\frac{H(\tau) - (1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta}) - (R+\eta)H(\frac{\pi}{R+\eta}) + \frac{6(R+\eta)}{5}H(\frac{\pi}{2(R+\eta)}) - \pi)}{2}\right)}$$

Soumise aux contraintes :

$$\pi = 2(R+\eta)H^{-1}(\tfrac{5\eta}{4(R+\eta)})$$
$$0 \leq \pi \leq \min(\tau, R+\eta)$$
$$0 \leq \eta \leq 1 - R - \tau + \pi \leq 1 - R$$

Et $\max_R(\alpha_{MMTQW}) = 0.0590365$ *pour* $R = 0.4514$, *valeur obtenue pour* $\pi \approx 0.0073872451$ *et* $\eta = 0.0249628844$



Courbe de α_{MMTQW} en fonction de R

Démonstration. On a $T = \binom{k+h}{\frac{p}{2}}^{2/5}$ et $h_1 = h_2 = \frac{h}{2} = T$.

- $P_{MMTQW} = \frac{\binom{k+h}{p}\binom{n-k-h}{t-p}}{\binom{n}{t}}$
- $T_{MMTQW} = \tilde{O}\left(T^{3/2}\right) = \tilde{O}\left(\binom{k+h}{\frac{p}{2}}^{3/5} 2^{-p/2}\right)$
- $2^h = \binom{k+h}{\frac{p}{2}}^{4/5}$

On en déduit la complexité totale de l'algorithme, $\sqrt{P_{MMTQW}}T_{MMTQW}$:

$$\tilde{O}\left(\sqrt{\frac{\binom{n}{t}}{\binom{k+h}{p}\binom{n-k-h}{t-p}}}\frac{\binom{k+h}{p/2}^{3/5}}{2^{p/2}}\right)$$
$$=$$
$$\tilde{O}\left(\sqrt{\frac{\binom{n}{t}}{\binom{k+h}{p}\binom{n-k-h}{t-p}}}\frac{\binom{k+h}{p/2}^{6/5}}{2^{p}}\right)$$

Par conséquent :

$$\boxed{\alpha_{MMTQW}(R) = \min_{\pi,\eta}\left(\frac{H(\tau) - (1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta}) - (R+\eta)H(\frac{\pi}{R+\eta}) + \frac{6(R+\eta)}{5}H(\frac{\pi}{2(R+\eta)}) - \pi)}{2}\right)}$$

Soumise aux contraintes :
$$\pi \approx 2(R+\eta)H^{-1}(\tfrac{5\eta}{4(R+\eta)})$$
$$0 \leq \pi \leq \min(\tau, R+\eta)$$
$$0 \leq \eta \leq 1-R-\tau+\pi \leq 1-R$$

□

5 ISD avancé 2 : technique des représentations étendue

5.1 Algorithme MMT^* classique

Cet algorithme ressemble à l'algorithme de MMT, d'où la notation MMT^*, et se trouve dans [1] (chapitre 10, section 2).

Technique des représentations étendue ou 1+1=0 Dans l'algorithme MMT, on représente l'ensemble I' des positions d'erreurs à l'aide de deux sous-ensembles disjoints de taille $\frac{p}{2}$ de $\{1, ..., k+h\}$. Autrement dit, les coordonnées non-nulles du vecteur d'erreur sont écrites comme 1+0 ou 0+1 et les coordonnées nulles comme 0+0. Or, en binaire, 0 s'écrit également comme 1+1. Il est possible d'augmenter le nombre de représentations en exploitant cette double façon d'écrire le 0 binaire.

Par conséquent, on considérera dans cette section des représentations de l'ensemble I' comme $I_1 \Delta I_2$ avec $|I_i| = \frac{p}{2} + \varepsilon$, $i = 1, 2, \varepsilon > 0$, $|I_1 \cap I_2| = \varepsilon$. Cela permet d'avoir $\binom{p}{p/2}\binom{k+h-p}{\varepsilon}$ représentations. Le deuxième facteur est grand lorsque p et par conséquent ε sont petits, ce qui arrive souvent car les vecteurs d'erreur sont souvent creux.

De plus, contrairement à MMT où à la fin il fallait filtrer les sous-ensembles I' de taille p parmi les $I', |I'| \leq p$, ici, en choisissant à bon escient la valeur de ε, par construction on n'aura à la fin que très peu d'ensembles I' de taille différente de p et ce sans avoir besoin de filtrage supplémentaire.

Description de l'algorithme En effectuant les modifications décrites ci-dessus, on se retrouve avec un algorithme semblable à SS et à MMT avec quelques différences dans les définitions des ensembles et listes.
$\begin{cases} J_{1,1} = J_{2,1} = \{1, ..., \frac{k+h}{2}\} \\ J_{1,2} = J_{2,2} = \{\frac{k+h}{2} + 1, ..., \frac{k+h}{2}\} \end{cases}$
Et $S_{i,j} = \{I_{i,j} \subset J_{i,j} \mid |I_{i,j}| = p/4 + \varepsilon/2\}$, $|S_{i,j}| = \binom{\frac{k+h}{2}}{p/4+\varepsilon/2} \approx \binom{k+h}{p/2+\varepsilon}^{1/2}$.
Comme dans MMT, on remarque que I' ne s'écrit pas tout à fait comme $I_1 \Delta I_2$ avec $I_i \subset \{1, ..., k+h\}, |I_i| = p/2 + \varepsilon$, mais un calcul semblable à celui fait dans MMT montre que cette restriction ne fait perdre qu'un facteur polynomial.

Preuve de correction de $DecodeCollision_{MMT^*}$

— La liste L_1 contient les $I_1 = I_{1,1} \sqcup I_{1,2}$ tels que $r^{(1)} = r_1^{(1)} = \sum_{i \in I_{1,1}} H_i'^{(1)} + \sum_{i \in I_{1,2}} H_i'^{(1)} = \sum_{i \in I_1} H_i'^{(1)}$.

— La liste L_2 contient les $I_2 = I_{1,2} \sqcup I_{2,2}$ tels que $s'^{(1)} - r^{(1)} = r_2^{(1)} = \sum_{i \in I_{2,1}} H_i'^{(1)} + \sum_{i \in I_{2,2}} H_i'^{(1)} = \sum_{i \in I_2} H_i'^{(1)}$.

— On a $|I_i| = p/2 + \varepsilon$ pour $i = 1, 2$ par construction car $I_{i,1} \cap I_{i,2} = \emptyset$.

— La liste L contient donc les $I' = I_1 \Delta I_2$ tels que :

$$\begin{cases} I_1 \in L_1, \text{ i.e. } \sum_{i \in I_1} H_i'^{(1)} = r^{(1)} \\ I_2 \in L_2, \text{ i.e. } \sum_{i \in I_2} H_i'^{(1)} = s'^{(1)} - r^{(1)} \\ \sum_{i \in I_1} H_i'^{(2)} + \sum_{i \in I_2} H_i'^{(2)} = s'^{(2)} \end{cases}$$
$$\Rightarrow \begin{cases} \sum_{i \in I_1} H_i'^{(1)} + \sum_{i \in I_2} H_i'^{(1)} = s'^{(1)} \\ \sum_{i \in I_1} H_i'^{(2)} + \sum_{i \in I_2} H_i'^{(2)} = s'^{(2)} \end{cases}$$
$$\Rightarrow \sum_{i \in I'} H_i' = s'.$$

— En choisissant bien la valeur de ϵ, $I' = I_1 \Delta I_2$ vérifiera en moyenne $|I'| = p$.

Par conséquent la liste finale L est bien comme on souhaite.

Représentations Un argument analogue à celui utilisé pour MMT montre que, pour qu'il reste à la fin en moyenne un nombre constant de représentations de chaque ensemble de positions d'erreurs, il faut prendre
$h_1 = \log_2 \left(\left(\binom{p/2}{p/4} \binom{\frac{k+h-p}{2}}{\varepsilon/2} \right)^2 \right) \approx p + (k+h-p) H(\frac{\varepsilon}{k+h-p})$.

Complexité

THÉORÈME 24. *Si H est une matrice aléatoire, l'exposant de complexité temporelle α_{MMT^*} de l'algorithme MMT^* est tel que $|\alpha_{MMT} - \alpha_{MMT^*}|$ est négligeable.*

Démonstration. On cherche à choisir ε afin d'avoir $|I_1 \cap I_2| = \varepsilon$, c'est-à-dire que le poids de la somme des vecteurs d'erreurs dont les coordonnées non-nulles sont indicées par I_1 et I_2 soit p. Or, on peut considérer deux vecteurs de poids $\frac{p}{2} + \varepsilon$ et de longueur $k + h$ comme suivant des lois de Bernoulli de paramètre $q = \frac{\left(\frac{p}{2} + \varepsilon\right)}{k+h}$ et donc, en appliquant théorème 8 (page 6), leur somme suit la loi de Bernoulli de paramètres $2q(1-q)$. On voudrait qu'elle soit en moyenne de poids p, i.e. suive la loi de Bernoulli de paramètres $\frac{p}{k+h}$. On résout donc :

$$2q(1-q) = \frac{p}{k+h}$$
$$(p + 2\varepsilon)\left(1 - \frac{\frac{p}{2}+\varepsilon}{k+h}\right) = p$$
$$p + 2\varepsilon - \frac{2\left(\frac{p}{2}+\varepsilon\right)^2}{k+h} = p$$
$$\varepsilon = \frac{\left(\frac{p}{2}+\varepsilon\right)^2}{k+h}$$

On voudrait donc que $\frac{\left(\frac{p}{2}+\varepsilon\right)^2}{k+h} = \varepsilon$.

On utilise cette égalité pour donner une valeur approximative de ε en fonction

des autres paramètres. Posons $\epsilon := \frac{\varepsilon}{n}$

$$\varepsilon = \frac{\left(\frac{p}{2}+\varepsilon\right)}{k+h}$$
$$n\epsilon = \frac{\left(\frac{n\pi}{2}+n\epsilon\right)^2}{nR+n\eta}$$
$$n\epsilon = \frac{n^2}{n}\frac{\left(\frac{\pi}{2}+\epsilon\right)^2}{R+\eta}$$
$$\epsilon = \frac{\left(\frac{\pi}{2}+\epsilon\right)^2}{R+\eta}$$
$$(R+\eta)\,\epsilon = \frac{\pi^2}{4} + \pi\epsilon + \epsilon^2$$
$$(R+\eta-\pi-\epsilon)\,\epsilon = \frac{\pi^2}{4}$$
$$R\left(1+\frac{\eta-\pi-\epsilon}{R}\right)\epsilon = \frac{\pi^2}{4}$$
$$\epsilon = \frac{\pi^2}{4R}\frac{1}{1+\frac{\eta-\pi-\epsilon}{R}}$$
$$\epsilon = \frac{\pi^2}{4R}\left(1-\frac{\eta-\pi-\epsilon}{R} + o(\eta+\pi+\epsilon)\right)$$

Donc, en pratique (cf. les calculs de développements limités), nous pouvons approximer ϵ par $\frac{\pi^2}{4R}$ pour avoir les meilleurs résultats.

Or, cela entraîne (cf. encore les calculs de développements limités) que $|\alpha_{MMT} - \alpha_{MMT^*}| = O(\pi^2)$. □

5.2 Algorithme de Becker, Joux, May et Meurer classique

Nous avions représenté les algorithmes SS et MMT sous forme d'un arbre à deux niveaux. Les listes du niveau 0 étaient obtenues à partir des listes du niveau 1 par collision (SS) ou par la technique des représentations (MMT), et les listes du niveau 1 étaient obtenues à partir de celles du niveau 2 par collision. Les schémas utilisés ont également rendus évidents ces deux techniques via l'utilisation de deux couleurs différentes.

Dans ce qui suit, nous allons utiliser le vocabulaire suivant pour désigner ces deux technique : résoudre/obtenir par collisions ou résoudre/obtenir par représentations.

L'algorithme de Becker, Joux, May et Meurer, que l'on notera $BJMM$, utilise la technique des représentations étendues sur trois niveaux, où les listes des niveaux 0 et 1 sont obtenues *par représentation* à partir des listes des niveaux respectivement 1 et 2, et celles du niveau 2 sont obtenues à partir de celles du niveau 3 *par collision*.

De plus, contrairement à MMT^* où le paramètre ε de la technique des représentations était choisi de telle sorte à garantir en moyenne le poids p pour la somme des deux vecteurs, il n'y a ici aucune contraine sur les paramètres ε_1 et ε_2 des deux niveaux de représentation. Cela entraîne qu'il y aura une partie importante des vecteurs des deux listes à un niveau dont la somme n'aura pas le poids souhaité au niveau suivant. Par conséquent, il y aura une différence non-négligeable entre d'un côté *le temps de construction* d'une liste et de l'autre côté sa *taille*.

Considérations préliminaires On écrira : $h = h_1 + h_2$, $h_1 = h_{1,1} + h_{1,2}$ et $h_2 = h_{2,1} + h_{2,2}$. On définit les $H'^{(i)}$ comme suit et de manière analogue $v^{(i)}$

pour un vecteur v de longueur h :

$$H' = \begin{bmatrix} H'^{(1)} \\ H'^{(2)} \end{bmatrix} = \begin{bmatrix} H'^{(1,1)} \\ H'^{(1,2)} \\ H'^{(2)} \end{bmatrix}$$

On note S l'ensemble des indices possibles du vecteur d'erreur e', i.e. $S = \{I' \subset \{1, ..., h+k\} \mid |I'| = p\}$.
On définit $0 < \varepsilon_1 \leq \varepsilon_2 < \pi$ où ε_i est le nombre des positions d'erreurs au niveau i sur lesquelles on espère pouvoir utiliser $1 + 1 = 0$.

Puis on définit :
$f_1(p) = p/2 + \varepsilon_1$
$f_2(p) = f_1(p)/2 + \varepsilon_2 = p/4 + \varepsilon_1/2 + \varepsilon_2$
$f_3(p) = f_2(p)/2 = p/8 + \varepsilon_1/4 + \varepsilon_2/2$
On définit aussi deux listes de base :
$S_{3,1} = \{I_{3,j_1} \subset \{1, ..., \frac{k+h}{2}\} \mid |I_{3,j_1}| = f_3(p)\}$
$S_{3,2} = \{I_{3,j_2} \subset \{\frac{k+h}{2} + 1, ..., k+h\} \mid |I_{3,j_2}| = f_3(p)\}$

On suppose que l'on a déjà la valeur optimale de h_1 ainsi qu'un vecteur r qui débouche sur un résultat.

Algorithme $DecodeCollision_{BJMM}$

Algorithme 15 : $DecodeCollision_{BJMM}$

Entrées : H', y', $s' = H'y'^T$, p, $(h_i)_{1 \leq i \leq 4}$, r vecteur de longueur h tel que $r^{(2)} = 0$, $S_{3,1}$ et $S_{3,2}$
Sorties : une liste L de d'ensembles I', $|I'| = p$ tels que $\sum_{i \in I'} H'_i = s'$

$r_1 \leftarrow r$
$r_2 \leftarrow s' - r$
`/* Construction des listes de base (niveau 3) */`
pour $i = 0, 1$ **faire**
  Choisir un vecteur v_{2i+1} de longueur h_1 tels que $v_{2i+1}^{(1,2)} = 0$
  $v_{2(i+1)} \leftarrow r_i^{(1)} + v_{2i+1}$
  Initialiser les listes de base $L_{3,4i+1}, ..., L_{3,4(i+1)}$ à partir de $S_{3,1}$ et $S_{3,2}$
  `/* Construction des listes du niveau 2 par collisions */`
  pour $j = 1, 2$ **faire**
    $L_{2,2i+j}(r) \leftarrow L_{3,4i+2(j-1)+1} \overset{v_i}{\underset{h_{1,1}}{\bowtie}} L_{3,4i+2(j-1)+2}$
`/* Construction des listes du niveau 1 par représentations */`
pour $i = 1, 2$ **faire**
  $L_{1,i}(r) \leftarrow L_{2,2i-1} \overset{r_i}{\underset{h_{1,2}}{\bowtie}} L_{2,2i}$
`/* Construction de la liste finale (niveau 0) par représentations */`
$L \leftarrow L_{1,1}(r) \overset{s'}{\underset{h_2}{\bowtie}} L_{1,2}(r)$
retourner L

Explication plus approfondie et preuve de correction

— Les $L_{3,i}$ contiennent les $I_{3,i}, |I_{3,i}| = f_3(p) = f_2(p)/2 = p/8 + \varepsilon_1/4 + \varepsilon_2/2$
— Les $L_{2,i}(r)$ contiennent les $I_{2,i} \subset \{1, ..., k+h\}, |I_{2,i}| = f_2(p)$ tels que $\sum_{k \in I_{2,i}} H_k'^{(1,1)} = v_i^{(1,1)}$
— $L_{1,i}$ contient les $I_i, I_i := I_{2,2i-1} \Delta I_{2,2i}$ et $|I_1| = f_1(p)$ tels que $\sum_{k \in I_{2,2i-1}} H_k'^{(1,2)} + \sum_{k \in I_{2,2i}} H_k'^{(1,2)} = r_i^{(1,2)}$
Donc :
$\begin{cases} \sum_{k \in I_{2,2i-1}} H_k'^{(1,1)} = v_{2i-1}^{(1,1)} \\ \sum_{k \in I_{2,2i}} H_k'^{(1,1)} = v_{2i}^{(1,1)} = r_i^{(1,1)} + v_{2i-1}^{(1,1)} \\ \sum_{k \in I_{2,2i-1}} H_k'^{(1,2)} + \sum_{k \in I_{2,2i-1}} H_k'^{(1,2)} = r_i^{(1,2)} \end{cases}$
$\begin{cases} \sum_{k \in I_i} H_k'^{(1,1)} = r_i^{(1,1)} \\ \sum_{k \in I_i} H_k'^{(1,2)} = r_i^{(1,2)} \end{cases}$
$\Rightarrow \sum_{k \in I_i} H_k'^{(1)} = r_i^{(1)}$
— L contient les $I', I' := I_1 \Delta I_2$ avec $|I'| = p$ tels que $\sum_{k \in I_1} H_k'^{(2)} + \sum_{k \in I_2} H_k'^{(2)} = s'^{(2)}$
Donc :
$\begin{cases} \sum_{k \in I_1} H_k'^{(1)} = r'^{(1)} \\ \sum_{k \in I_2} H_k'^{(1)} = s'^{(1)} + r'^{(1)} \\ \sum_{k \in I_1} H_k'^{(2)} + \sum_{k \in I_2} H_k'^{(2)} = s'^{(2)} \end{cases}$
$\begin{cases} \sum_{k \in I'} H_k'^{(1)} = s'^{(1)} \\ \sum_{k \in I'} H_k'^{(2)} = s'^{(2)} \end{cases}$
$\Rightarrow \sum_{k \in I'} H_k' = s'$

Donc l'algorithme est correct.

Analyse de complexité Nous savons, d'après les analyses de la représentation à un niveau, qu'en optimisant $h_{1,1}$ et $h_{1,2}$ on peut faire en sorte d'obtenir en moyenne une liste non-vide en $\tilde{O}(1)$ tours de boucle. On se concentre donc d'abord sur ce qui se passe à l'intérieur de la boucle. Par construction, on a :

1. $|L_{3,i}| = |S_{3,i}| = \binom{\frac{k+h}{2}}{f_3(p)} \approx \sqrt{\binom{k+h}{f_2(p)}}$
2. $|L_{2,i}| = O\left(\binom{k+h}{f_2(p)} 2^{-h_{1,1}}\right)$
3. $|L_{1,i}| = O\left(\binom{k+h}{f_1(p)} 2^{-h_1}\right)$

1. Construction des listes $L_{3,i}$
   - Complexité temporelle $T_3 : \tilde{O}\left(|L_{3,i}|\right)$
   - Complexité spatiale $S_3 : O\left(|L_{3,i}|\right)$
2. Construction des listes $L_{2,i}$
   - Complexité temporelle $T_2 : \tilde{O}\left(\max\left(|L_{3,i}|, |L_{3,i}|^2 2^{-h_{1,1}}\right)\right) = \tilde{O}\left(\max\left(S_3, S_3^2 2^{-h_{1,1}}\right)\right)$
   - Complexité spatiale $S_2 : |L_{2,i}|$.
3. Construction des listes $L_{1,i}$
   - Complexité temporelle $T_1 : \tilde{O}\left(\max\left(|L_{2,i}|, |L_{2,i}|^2 2^{-h_{1,2}}\right)\right) = \tilde{O}\left(\max\left(S_2, S_2^2 2^{-h_{1,2}}\right)\right)$
   - Complexité spatiale $S_1 : |L_{1,i}|$.
4. Construction de la liste L
   - Complexité temporelle : $\tilde{O}\left(\max\left(|L_{1,i}|, |L_{1,i}|^2 2^{-h_2}\right)\right) = \tilde{O}\left(\max\left(S_1, S_1^2 2^{-h_2}\right)\right)$
   - Complexité spatiale : $\tilde{O}(1)$

Au final, l'algorithme a une complexité temporelle en :

$$\tilde{O}\left(\max\left(S_3, S_3^2 2^{-h_{1,1}}, S_2, S_2^2 2^{-h_{1,2}}, S_1, S_1^2 2^{-h_2}\right)\right)$$

i.e.

$$\tilde{O}\left(\max\left(\sqrt{\binom{k+h}{f_2(p)}}, \binom{k+h}{f_2(p)}2^{-h_{1,1}}, \binom{k+h}{f_2(p)}^2 2^{-h_1-h_{1,1}}, \binom{k+h}{f_1(p)}2^{-h_1}, \binom{k+h}{f_1(p)}^2 2^{-h-h_1}\right)\right)$$

Et une complexité spatiale en :

$$\tilde{O}\left(\max\left(S_3, S_2, S_1\right)\right)$$

i.e.

$$\tilde{O}\left(\max\left(\sqrt{\binom{k+h}{f_2(p)}}, \binom{k+h}{f_2(p)}2^{-h_{1,1}}, \binom{k+h}{f_1(p)}2^{-h_1}\right)\right)$$

On cherche maintenant à optimiser $h_{1,1}$.

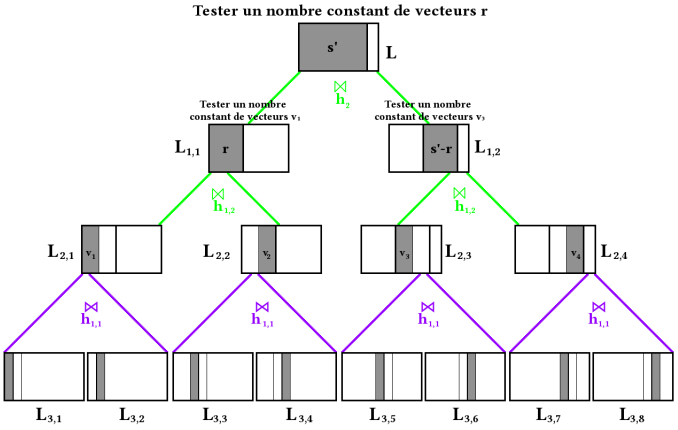
La probabilité qu'un $I_{2,j}$ s'écrive $I_{3,2j-1} \sqcup I_{3,2j}$ est $\binom{\frac{k+h}{2}}{f_3(p)}^2 \binom{k+h}{f_2(p)}^{-1} = \tilde{O}(1)$. Par conséquent, la probabilité qu'ils s'écrivent tous les quatre sous cette forme est aussi $\tilde{O}(1)$. Avec cette hypothèse sur les $I_{2,j}$, le nombre de couples d'ensembles (I_{2,j_1}, I_{2,j_2}) tel que chaque $I_{1,i}$ s'écrit $I_{2,j_1} \Delta I_{2,j_2}$ est :

$$2\binom{\binom{f_1(p)}{f_1(p)/2}\binom{k+h-f_1(p)}{\varepsilon_2}}\tilde{O}(1) \approx \tilde{O}\left(2^{f_1(p)+k+h-f_1(p)H(\frac{\varepsilon_2}{k+h-f_1(p)})}\right)$$

On a deux boucles consécutives pour trouver deux bons vecteurs v_1 et v_3 parmi $2^{h_{1,1}}$ possibilités, afin de les utiliser pour construire les listes du niveau 2. Ainsi, pour qu'en moyenne il y ait une représentation de chaque ensemble de positions d'erreurs qui subsiste dans la liste finale, il faut prendre $h_{1,1} \approx f_1(p) + (k+h-f_1(p))H(\frac{\varepsilon_2}{k+h-f_1(p)})$.

Algorithme $Recherche_{BJMM}$ Dans cet algorithme, il y a une boucle pour trouver r au sein de laquelle on procède comme dans MMT, avec la différence que l'on fait appel à $DecodeCollision_{BJMM}$.

En choisissant $h_1 = \log_2\left(\binom{p}{p/2}\binom{k+h-p}{\varepsilon_1}\right) \approx p + (k+h-p)H(\frac{\varepsilon_1}{k+h-p})$, il y aura en moyenne un nombre constant de représentations de chaque ensemble de positions d'erreur à la fin.



Recherche$_{BJMM}$ - illustration sous forme d'arbre
violet : recherche de collisions classique
vert : technique des représentations
$\bowtie$: Merge-Join

Complexité

THÉORÈME 25 ([1], [2]). *Si H est une matrice aléatoire, en notant $R := \frac{k}{n}$, $\tau := \frac{t}{n}$, $\pi := \frac{p}{n}$,$\eta = \frac{h}{n}$ et $\epsilon_i = \frac{\varepsilon_i}{n}, i = 1, 2$, on a l'expression suivante pour l'exposant α_{BJMM} de l'algorithme de Becker, Joux, May et Meurer :*

$$\alpha_{BJMM}(R) = \min_{\pi,\eta,\epsilon_1,\epsilon_2} (\beta(R,\pi,\eta) + \max_{i=1,2,3,4,5} \gamma_i(R,\eta,\pi,\epsilon_1,\epsilon_2))$$

Avec :

$$\boxed{\beta(R,\pi,\eta) = H(\tau) - (1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta}) - (R+\eta)H(\frac{\pi}{R+\eta})}$$

$$\boxed{\gamma_1(R,\eta,\pi) = \frac{R+\eta}{2}H(\frac{f_2(\pi)}{R+\eta})}$$

$$\boxed{\gamma_2(R,\eta,\pi) = (R+\eta)H(\frac{f_2(\pi)}{R+\eta}) - \eta_{1,1}}$$

$$\boxed{\gamma_3(R,\eta,\pi) = 2(R+\eta)H(\frac{f_2(\pi)}{R+\eta}) - \eta_1 - \eta_{1,1}}$$

$$\boxed{\gamma_4(R,\eta,\pi) = (R+\eta)H(\frac{f_1(\pi)}{R+\eta}) - \eta_1}$$

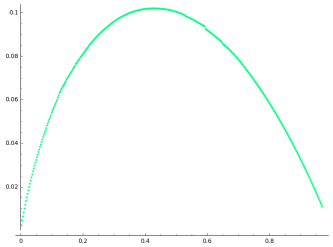
$$\boxed{\gamma_5(R,\eta,\pi) = 2(R+\eta)H(\frac{f_1(\pi)}{R+\eta}) - \eta - \eta_1}$$

Soumise aux contraintes :

$$\eta_1 = \pi + (R+\eta-\pi)H(\tfrac{\epsilon_1}{R+\eta-\pi}) \leq (R+\eta)H(\tfrac{f_1(\pi)}{R+\eta})$$
$$\eta_{1,1} = f_1(\pi) + (R+\eta-f_1(\pi))H(\tfrac{\epsilon_2}{R+\eta-f_1(\pi)}) \leq (R+\eta)H(\tfrac{f_2(\pi)}{R+\eta})$$
$$0 \leq \pi \leq \min(\tau, R+\eta)$$
$$0 \leq \eta \leq 1-R-\tau+\pi \leq 1-R$$
$$0 \leq \epsilon_1 \leq R+\eta-\pi$$
$$0 \leq \epsilon_2 \leq R+\eta-f_1(\pi)$$

$$\epsilon_2 \leq \epsilon_1$$
$$0 \leq \eta_{1,1} \leq \eta_1 \leq \eta$$

Et $\max_R(\alpha_{BJMM}) = 0.1018866$ *pour* $R = 0.4275$*, valeur obtenue pour*
$\pi \approx 0.0560712$
$\eta \approx 0.21226227$
$\epsilon_1 \approx 0.01135971$
$\epsilon_2 \approx 0.002035141$.



Courbe de α_{BJMM} *en fonction de* R

Démonstration. Suit des considérations précédentes. □

5.3 Algorithme de Becker, Joux, May et Meurer quantique avec Quantum Walk

On notera cet algorithme $BJMMQW$.

5.3.1 Prélude : représentations VS collision

On peut trouver des considérations du même type dans [1] (chapitre 10, section 2.4).

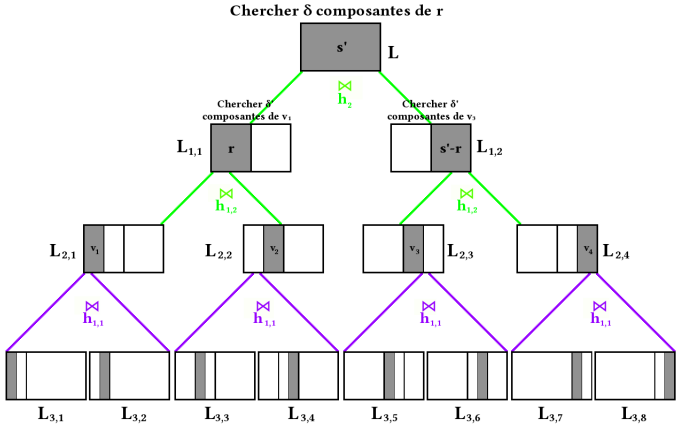
Description Comme on l'a vu, dans l'algorithme $BJMM$ la technique des représentations est utilisée deux fois de suite. Aux niveaux 1 et 2, les $\log_2$ du nombre de représentations de l'ensemble des positions d'erreurs sont comme suit :

1. Niveau 1 : $\pi + (R + \eta - \pi)H(\frac{\epsilon_1}{R+\eta-\pi})$
2. Niveau 2 : $f_1(\pi) + (R + \eta - f_1(\pi))H(\frac{\epsilon_2}{R+\eta-f_1(\pi)})$

On rappelle le constat suivant fait à propos de l'algorithme MMT : il est possible de choisir h_1 supérieur au nombre de représentations, dans ce cas, il y aura certains ensembles de positions d'erreurs qui n'auront pas de représentation, mais celles qui en ont, en one une seule. Alors, quelque soit le vecteur intermédiaire r choisi, il y aura une garantie d'égalité sur un nombre égal au $\log_2$ du nombre de représentations de ses composantes et il faut chercher les autres composantes de manière exhaustive.

Ce raisonnement se transpose facilement aux deux niveaux de l'algorithme $BJMM$. Ainsi, on prend h_1 et $h_{1,1}$ supérieurs au $\log_2$ du nombre de représentations du niveau concerné. Si l'on note δ le nombre de composantes du vecteur r que l'on cherche, et δ' le nombre de composantes de chacun des vecteurs v_1 et v_3 que l'on cherche, on a :

1. $\delta = h_1 - p - (k + h - p)H(\frac{\varepsilon_1}{k+h-p})$
2. $\delta' = h_{1,1} - f_1(p) - (k + h - f_1(p))H(\frac{\varepsilon_2}{k+h-f_1(p)})$



$Recherche_{BJMM}$ dans ce cas - illustration sous forme d'arbre
violet : recherche de collisions classique
vert : technique des représentations
$\bowtie$: Merge-Join

Les calculs suivants prouvent ces affirmations de manière plus rigoureuse.

On se concentre d'abord sur δ'. Le nombre total d'éléments que l'on peut atteindre, à partir des listes de base, dans chacune des listes $L_{1,i}$, et qui vérifient la condition d'égalité sur h_1 bits est au maximum $|L_{3,i}|^4 2^{-h_1}$. Or, en pratique, pour construire les $L_{1,j}$ à partir des $L_{3,i}$, on boucle sur $2^{\delta'}$ vecteurs et on représente les ensembles de positions d'erreurs de $\binom{f_1(p)}{f_1(p)/2}\binom{k+h-f_1(p)}{\varepsilon_2}$ manières : on atteint ainsi $2^{\delta'}\binom{f_1(p)}{f_1(p)/2}\binom{k+h-f_1(p)}{\varepsilon_2}|L_{3,i}|^4 2^{-h_1-h_{1,1}}$ éléments.

On cherche donc à choisir δ' de telle sorte à avoir l'égalite :

$$2^{\delta'}\binom{f_1(p)}{f_1(p)/2}\binom{k+h-f_1(p)}{\varepsilon_2}|L_{3,i}|^4 2^{-h_1-h_{1,1}} = |L_{3,i}|^4 2^{-h_1}$$

$$2^{\delta'} 2^{f_1(p)+(k+h-f_1(p))H(\frac{\varepsilon_2}{k+h-f_1(p)})} = 2^{h_{1,1}}$$

$$\delta' = h_{1,1} - f_1(p) - (k + h - f_1(p))H(\frac{\varepsilon_2}{k+h-f_1(p)})$$

On obtient δ de manière analogue. En effet, le total d'éléments que l'on peut atteindre dans la liste L à partir des listes $L_{1,i}$ et qui vérifient la condition d'égalité sur h bits au syndrome est au maximum $|L_{1,i}|^2 2^{-h}$. En pratique, cela se fait en bouclant sur 2^{δ} vecteurs et avec $\binom{p}{p/2}\binom{k+h-p}{\varepsilon_1}$ représentations des positions d'erreurs. On atteint ainsi au total $2^{\delta}\binom{p}{p/2}\binom{k+h-p}{\varepsilon_1}|L_{1,i}|$ éléments.

Il faut donc choisir δ de sorte à avoir l'égalité :

$$2^{\delta}\binom{p}{p/2}\binom{k+h-p}{\varepsilon_1}|L_{1,i}| = |L_{1,i}|^2 2^{-h}$$

Et un calcule analogue prouve le résultat.

Conséquence sur la forme de l'algorithme et la version quantique Cette modification de l'algorithme aura une conséquence non-négligeable sur la forme de l'algorithme. Quand on utilisait complètement la technique des représentations, on a vu qu'on pouvait construire les listes qui dépendent de v_1 et de v_3 de manière consécutive, puisque l'on avait la garantie d'avoir le bon couple (v_1, v_3) au bout d'un nombre constant d'esssais. Autrement dit, on pouvait considérer v_1 et v_3 comme indépendants. Les choses deviennent plus compliquées lorsque l'on utilise la technique hybride esquissée ci-dessus.

La question est en effet de savoir s'il est possible de trouver v_1 et v_3 à l'aide de deux boucles consécutives ou s'il faut utiliser une boucle imbriquée, en d'autres termes, si c'est le couple (v_1, v_3) qu'il faut chercher. Puisque c'est un algorithme qui utilise cette technique hybride que l'on va adapter au cadre quantique, la réponse à cette question détermine la meilleure complexité que l'on puisse espérer avoir pour cet algorithme.

En théorie, il y a une dépendance entre v_1 et v_3 et il faut donc chercher le couple (v_1, v_3). Mais en pratique, pour pouvoir chercher v_1 et v_3 à l'aide de deux boucles consécutives, il suffit de d'avoir une garantie de leur indépendance. On a vu que pour chaque v_1 et pour chaque v_3, on explore seulement une partie de l'ensemble des éléments que l'on peut atteindre. C'est en réalité pour cette raison qu'il y a une dépendance. Or, on peut chercher v_1 et v_3 de manière consécutive si l'on peut garantir que tout les éléments atteignables sont explorés au niveau 2. Cela revient à considérer l'union des listes construites pour chaque v_1 et pour chaque v_3 pour construire les listes du niveau supérieur (niveau 1). Mais cela augmente la mémoire d'un facteur $2^{\delta'}$.

On a donc le choix entre la recherche consécutive de v_1 et de v_3 à condition d'utiliser $2^{delta'}$ fois plus de mémoire, ou la recherche, plus coûteuse, du couple (v_1, v_3), si l'on ne veut pas garder tous les résultats en mémoire.

Pour la version quantique de cet algorithme, cela nous pousse à privilégier la recherche du couple (v_1, v_3). En effet, on va voir qu'il est possible d'avoir des coûts favorables pour les opérations de la marche aléatoire comme pour $MMTQW$ et $SSQW$ à condition de prendre les listes de tous les niveaux de taille égale. Or, si T est la taille des listes de base, pour pouvoir faire la recherche de v_1 et de v_3 de manière consécutive, il faudrait avoir accès à l'intégralité des $2^{\delta'}\frac{T}{2^{h_{1,1}}}$ éléments atteignables au niveau 2. Or, avec l'égalité $h_{1,1} = \delta' + f_1(p) - (k+h-f_1(p))H(\frac{\varepsilon_2}{k+h-f_1(p)})$, cela entraîne que les listes de base seront de taille $T2^{-f_1(p)+(k+h-f_1(p))H(\frac{\varepsilon_2}{k+h-f_1(p)})}$. Or, comme pour $MMTQW$, cette valeur au dénominateur est trop petit pour avoir ces listes de même taille T que les listes de base.

Le sens de ces considérations deviendra plus clair dans la partie qui suit, où l'on décrit l'algorithme $BJMMQW$.

5.3.2 Description de l'algorithme

Nous allons utiliser ce qui précède pour construire l'algorithme qui utilisera une marche aléatoire. Deux remarques s'imposent avant de rentrer dans les détails.

1. La marche aléatoire va calculer les structures de données des trois niveaux à partir d'un sous-ensemble des ensembles de positions d'erreurs possibles du niveau 3. Or, pour faire ce calcul, il lui faut les bonnes valeurs pour r, mais aussi v_1 et v_3. La valeur de r se calcule en appliquant l'algorithme de Grover à la marche aléatoire comme pour $SSQW$ et $MMTQW$. Pour avoir les bonnes valeurs de v_1 et v_3, il y a plusieurs façons de procéder.
   (a) On peut les calculer lors de SETUP. Mais alors, elles dépendent de l'état courant de la marche aléatoire et on n'a plus aucune garantie qu'elles restent valables après une étape de la marche. Ainsi il faudrait recalculer les structures de données à chaque étape de la marche, ce qui entraîne $U = S$. Or, cela est peu souhaitable.
   (b) Il est également possible, après avoir trouvé la bonne valeur de r avec Grover, de chercher v_1 et v_3 une fois pour toute en utilisant la marche aléatoire de recherche du vecteur d'erreur comme oracle de vérification. Mais alors, cette marche a besoin du *couple* (v_1, v_3) pour s'exécuter. Autrement dit, contrairement à $BJMM$ classique où v_1 et v_3 étaient indépendants et que l'on pouvait les chercher séparément et de manière consécutive, ici il y a une dépendance entre v_1 et v_3 pour les raisons esquissées dans la partie précédente, ce qui fait que l'on ne peut pas les chercher séparément, ni même se servir d'une marche aléatoire de recherche de collision.
2. Lors du calcul de la complexité de $BJMM$, on a différencié entre deux choses : d'un côté le temps de construction des listes du niveau i à partir des listes du niveau $i+1$, ce qui dépend du poids des vecteurs d'erreur dans les listes du niveau $i+1$, et de l'autre côté la taille des listes de niveau i, qui dépend du poids des vecteurs d'erreur que l'on y a gardés. Puisque l'on travaillera dans $BJMMQW$ avec des sous-ensembles des ensembles de tous les vecteurs d'erreurs possibles, il n'est plus possible d'estimer de manière aussi exacte la taille des listes. Seul le calcul du temps de construction reste possible. Or, le temps de construction d'une liste donne une borne supérieure sur sa taille. On va donc se servir du temps de construction à la place de la taille : cela donnera une borne supérieure sur le temps d'exécution réel.

La marche aléatoire et son graphe Le problème que l'on cherche à résoudre est celui de la recherche d'un 8-uplet : on cherche en effet $(I_{3,i})_{i=1,...,8} \in (S_{3,1} \times S_{3,2})^4$ tel que $\sum_{i=1}^{8} \sum_{k \in I_{3,i}} H'_k = s$ avec une condition de poids. On va donc considérer huit sous-ensembles $T_{3,2i-1} \subset S_{3,1}$ et $T_{3,2i} \subset S_{3,2}$, $i = 1, 2, 3, 4$, tous de même taille $T \leq |S_{3,i}| = \binom{\frac{k+h}{2}}{\frac{f_2(p)}{2}}$, $i = 1, 2$.

On peut mettre en bijection chaque sous-ensemble $T_{3,i}$, $i = 1, ..., 8$, avec un sous-ensemble de taille T d'un sous-ensemble de taille $|S_{3,i}|$, $i = 1, 2$. Ainsi, on peut mettre en bijection le 8-uplet $(T_{3,i})_{i=1,...,8}$ et une partie des sous-ensembles de taille $8T$ de $\{1, ..., 8|S_{3,i}|\}$.

On considère donc une marche aléatoire sur le graphe $J(8|S_{i,j}|, 8T)$ pour trouver un 8-uplet convenable pour une fonction f_{BJMMQW} qui sera définie ci-dessous. Il s'agit donc de la marche $QW(J(8|S_{i,j}|, 8T), f_{BJMMQW})$.

On a besoin de définir ou expliciter les éléments suivants :

1. Le trou spectral $\delta = \frac{8|S_{i,j}|}{8T(8(|S_{i,j}|-4T)} = \Omega\left(T^{-1}\right)$.
2. La fonction f_{BJMMQW} à calculer et la structure de données associée.
3. L'ensemble des éléments marqués $M_{f_{BJMMQW}}$, ce qui donnera la proportion ε des éléments marqués.
4. Le coût S de Setup.
5. Le coût C de Check.
6. Le coût U de Update.

La fonction f_{BJMMQW} et la structure de données associée La fonction f_{BJMMQW} est définie en reprenant la fonction $DecodeCollision_{BJMM}$ et en faisant des modifications analogues à celles qui ont été faites dans la fonction f_{MMT} par rapport à la fonction $DecodeCollision_{MMT}$. En particulier, elle prend en argument le vecteur r et le couple de vecteurs (v_1, v_3), il y a quatre niveaux (0 à 4), et à chaque niveau il y a quelques structures de données ordonnées stockant des couples "$(\sum_J, J)$", où J désigné un ensemble d'indices, et un nombre égal de structures de données ordonnées stockant simplement les ensembles d'indices.

La structure de données comporte donc en tout 29 sous-structures de données :

1. Niveau 3 : 8 structures de données $D_{f_{3,i}}$ et 8 structures de données $D_{I_{3,i}}$.
2. Niveau 2 : 4 structures de données $D_{f_{2,i}}$ et 4 structures de données $D_{I_{2,i}}$.
3. Niveau 1 : 2 structures de données $D_{f_{1,i}}$ et 2 structures de données $D_{I_{1,i}}$.
4. Niveau 0 : 1 structure de données D.

La proportion ε des éléments marqués On définit $M_{f_{BJMMQW}} := \{e \mid w_H(e) = t \text{ et } y = mG + e\}$

Sous l'hypothèse d'avoir choisi le bon vecteur r, la structure de données D contient des ensembles I' qui s'écrivent $I_{1,1}\Delta I_{1,2} = (I_{2,1}\Delta I_{2,2})\Delta(I_{2,3}\Delta I_{2,4})$ avec chaque $I_{2,i}$ s'écrivant $I_{3,2i-1} \sqcup I_{3,2i}$. La probabilité qu'un seul des $T_{3,i}$ contienne l'ensemble $I_{3,i}$ en question est $\frac{T}{|S_{3,i}|} \approx T\left(\frac{\frac{k+h}{2}}{\frac{f_2(p)}{2}}\right)$. La probabilité de succès ε est donc majorée par $\left(T\left(\frac{\frac{k+h}{2}}{\frac{f_2(p)}{2}}\right)\right)^8$.

Update

Théorème 26. *On choisit les vingt-neuf sous-structures de données de sorte à avoir un temps d'exécution constant ou au pire en* $\log(T)$ *pour les opérations suivantes :*

1. *Insertion d'un élément dans la structure de données.*

2. *Suppression d'un élément de la structure de données.*
3. *Recherche d'un élément dans la structure de données.*

De plus, on prend $2^{h_{1,1}} = 2^{h_{1,2}} = T \Leftrightarrow h_{1,1} = h_{1,2} = \log_2(T)$ *(par conséquent,* $h_1 = 2\log_2(T)$*).*
Alors la mise à jour de la structure de données s'effectue en temps $O(\log(T)) = \tilde{O}(1)$*.*

Démonstration. Notons que pour un $I_{3,j}$ particulier, il est possible d'écrire $I_{2,i} := I_{3,2i-1} \sqcup I_{3,2i}$ de T manières différentes (car il y a T choix de l'autre ensemble). Or, les seuls $I_{2,i}$ qui sont gardés dans la structure de données correspondante sont ceux qui vérifient en plus une condition de sommation sur les $h_{1,1}$ premières coordonnées. Par conséquent, il y a en moyenne $O(\frac{T}{2^{h_{1,1}}})$ ensembles $I_{2,i}$ faisant intervenir un $I_{3,j}$ particulier. Le choix $h_{1,1} = \log_2(T)$ garantit alors qu'il y en aura un nombre constant. De même, pour chaque $I_{2,i}$ il y a T façons d'écrire $I_{1,\ell} = I_{2,2\ell-1} \Delta I_{2,2\ell}$ et comme on garde seulement les $I_{1,\ell}$ qui vérifient une condition de sommation sur $h_{1,2}$ autres coordonnées, il y a en moyenne $O(\frac{T}{2^{h_{1,1}}})$ ensembles $I_{1,\ell}$ faisant intervenir un $I_{2,i}$ parituclier. Le choix $h_{1,2} = \log_2(T)$ garantit qu'il y en aura également un nombre constant ici. Par conséquent, sans perturber la probabilité d'aboutissement de l'algorithme de manière significative, nous pouvons limiter par un entier constant le nombre de $I_{2,i}$ resp. $I_{1,\ell}$ et donc de $I' = I_{1,1} \sqcup I_{1,2}$) faisant intervenir un $I_{3,j}$ resp. $I_{1,\ell}$ particulier.

La preuve que cela entraîne alors un temps de mise à jour en $O(\log(T))$ est analogue à celle donnée pour le théorème 21 mais en plus long. □

Setup Pour mettre en place l'état initial avec cette structure de données il faut évaluer f_{BJMMQW}, on calcule donc son coût en s'inspirant des calculs faits pour $DecodeCollision_{BJMM}$ à cette différence près que l'on considérera à chaque fois le temps d'exécution de la construction d'une liste comme une borne supérieure sur sa taille.

$$\tilde{O}\left(\max\left(|D_{f_{3,i}}|, |D_{f_{3,i}}|^2 2^{-h_{1,1}}, |D_{f_{3,i}}|^4 2^{-2h_{1,1}-h_{1,2}}, |D_{f_{3,i}}|^8 2^{-4h_{1,1}-2h_{1,2}-h_2}\right)\right)$$

$$\tilde{O}\left(\max\left(T, T^2 2^{-h_{1,1}}, T^4 2^{-h_1-h_{1,1}}, T^8 2^{-h_2-2h_1-2h_{1,1}}\right)\right)$$

Les choix des valeurs de $h_{1,1}$, $h_{1,2}$ pour garantir un coût de mise à jour en $\tilde{O}(1)$ entraînent :

$$\tilde{O}\left(\max\left(T, T^2 2^{-h_2}\right)\right)$$

Puis, en prenant $h_2 = \log_2(T)$ on obtient un coût de mise à jour en $\tilde{O}(T)$.

En termes de complexité en espace, f_{BJMMQW} coûte également $\tilde{O}(T)$.

Notons enfin que ces choix entraînent l'égalité $h = 3\log_2(T) \Leftrightarrow 2^h = T^3$.

Check Hormis la première étape de la marche aléatoire, où il faut chercher un élément $I' \in D$ qui donne un vecteur d'erreur e de poids t après application de *Dépoinçonner* (ce qui peut se faire en temps $\tilde{O}(\sqrt{T})$ avec l'algorithme de Grover), l'étape de vérification coûte en moyenne $\tilde{O}(1)$ puisqu'il suffit de vérifier pour le nombre constant de nouveaux ensembles d'indices ajoutés si l'un d'entre eux marche bien.

Complexité Sous l'hypothèse d'avoir les bons vecteurs r, v_1 et v_3, la marche aléatoire trouve le bon vecteur d'erreur e en temps :

$$S + \frac{1}{\sqrt{\varepsilon}}(C + \frac{1}{\sqrt{\delta}}U) = O(T) + \left(\frac{\frac{k+h}{2}}{\frac{f_2(p)}{2}}\right)^4 T^{-4}(\tilde{O}(\sqrt{T}) + O(\sqrt{T})O(1))$$
$$= \max\left(T, \left(\frac{\frac{k+h}{2}}{\frac{f_2(p)}{2}}\right)^4 T^{-7/2}\right)$$

Ces deux termes sont égaux si l'on prend $T = \left(\frac{\frac{k+h}{2}}{\frac{f_2(p)}{2}}\right)^{8/9} \approx \left(\frac{k+h}{f_2(p)}\right)^{4/9}$.

$Recherche_{BJMMQW}$ *$Recherche_{BJMMQW}$* applique la marche aléatoire quantique à f_{BJMMQW} après avoir trouvé les bons vecteurs r, v_1 et v_3. On va commencer par définir quelques fonctions avant de présenter le déroulement de *$Recherche_{BJMMQW}$*.

- On rappelle que la marche définie ci-dessus est la marche $QW(J(8\left(\frac{\frac{k+h}{2}}{\frac{f_2(p)}{2}}\right), 8T), f_{BJMMQW})$.
- On définit la fonction $\mathcal{O}_{BJMMQW}$ qui prend en argument r, v_1 et v_3 et applique la marche aléatoire définie ci-dessus avec ces arguments et renvoie 0 ssi la marche a trouvé un bon 8-uplet.
- On fait l'hypothèse que l'on connaît le bon vecteur r. On définit la procédure de recherche $Grover(2^{2\delta'}, \mathcal{O}_{BJMMQW}(r, ., .))$, qui renvoie une superposition $|omega\rangle$ tel qu'en mesurant on trouve (v_1, v_3) tel que $\mathcal{O}_{BJMMQW}(r, v_1, v_3) = 0$.
- On définit la fonction $\mathcal{G}_{BJMMQW}$ comme la fonction qui prend en argument r et applique la marche aléatoire de recherche du couple (v_1, v_3) avec ce r en premier argument de la fonction $\mathcal{O}_{BJMMQW}$ et renvoie 1 si cette recherche aboutit.

On peut maintenant donner la suite d'étapes de la fonction *$Recherche_{BJMMQW}$* :

Algorithme 16 : *$Recherche_{BJMMQW}$*

Entrées : H, y, $s = Hy^T$, t, p
Sorties : e tel que $s = mG + e$ et $w_H(e) = t$, $NULL$ si aucun tel vecteur n'a pu être trouvé

1 Préparer G',H',y' et calculer $s' = H'y'^T$
2 $|\rho\rangle \leftarrow Grover(2^{-\delta}, \mathcal{G}_{BJMMQW})$
3 $r \leftarrow Mesure(|\rho\rangle)$
4 $|\omega\rangle \leftarrow Grover(2^{2\delta'}, \mathcal{O}_{BJMMQW}(r, ., .))$
5 $(v_1, v_3) \leftarrow Mesure(|\omega\rangle)$
6 $I' = (I_{1,1}, I_{1,2}, I_{2,1}, I_{2,2}) \leftarrow QW(J(8\left(\frac{\frac{k+h}{2}}{\frac{f_2(p)}{2}}\right), 8T), f_{BJMM}(..., r, v_1, v_3, ...)$
7 $e \leftarrow Dépoinçonner(I')$
8 **retourner** e

On peut maintenant calculer la complexité de tout l'algorithme : pour trouver r, on fait $2^{\frac{\delta}{2}}$ requêtes à une fonction fait appel à une procédure de recherche de Grover qui cherche (v_1, v_3) en faisant $2^{\delta'}$ requêtes à une procédure qui utilise la marche aléatoire principale. Ainsi, le temps de calcul total T_{BJMMQW} est donnée par :

$$\tilde{O}(2^{\frac{\delta}{2}+\delta'}T)$$

Récapitulation

THÉORÈME 27. *Si H est une matrice aléatoire, en notant $R := \frac{k}{n}$, $\tau := \frac{t}{n}$, $\pi := \frac{p}{n}$, $\eta = \frac{h}{n}$ et $\epsilon_i = \frac{\varepsilon_i}{n}, i = 1, 2$, on a l'expression suivante pour l'exposant α_{BJMMQW} :*

$$\alpha_{BJMMQW}(R) = \min_{\pi,\eta,\epsilon_1,\epsilon_2} \left(\frac{H(\tau)-(1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta})-(R+\eta)H(\frac{\pi}{R+\eta})+\frac{8(R+\eta)}{3}H(\frac{f_2(\pi)}{R+\eta})+\nu(R,\pi,\eta,\epsilon_1,\epsilon_2)}{2} \right)$$

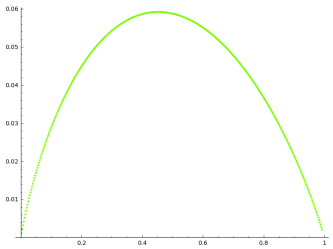
avec :

$$\boxed{\nu(R,\pi,\eta,\epsilon_1,\epsilon_2) = -2\pi - 2\epsilon_1 - (R+\eta-\pi)H(\frac{\epsilon_1}{R+\eta-\pi}) - 2(R+\eta-f_1(\pi))H(\frac{\epsilon_2}{R+\eta-f_1(\pi)})}$$

Soumise aux contraintes :

$$\pi = 4(R+\eta)H^{-1}(\tfrac{3\eta}{4(R+\eta)}) - 2\epsilon_1 - \epsilon_2$$
$$0 \leq \pi \leq \min(\tau, R+\eta)$$
$$0 \leq \eta \leq 1 - R - \tau + \pi \leq 1 - R$$
$$0 \leq \epsilon_1 \leq R + \eta - \pi$$
$$0 \leq \epsilon_2 \leq R + \eta - f_1(\pi)$$
$$\epsilon_2 \leq \epsilon_1$$

À l'aide de l'optimisation numérique, on obtient $\max_R(\alpha_{BJMMQW}) = 0.0591839$ *pour* 0.4537, *valeur obtenue pour* $\pi \approx 0.0103231$, $\eta \approx 0.0358304397$, $\epsilon_1 \approx 0.000649074$ *et* $\epsilon_2 \approx 0.00016391654$.



Courbe de α_{BJMMQW}

Démonstration. On rappelle que :
-$2^{h_{1,1}} = 2^{h_{1,2}} = 2^{h_2} = T$
-$2^{h_1} = T^2$
-$2^h = T^3$
-$T = \binom{k+h}{f_2(p)}^{4/9}$
On rappelle les valeurs de δ et δ' trouvées dans la partie "représentations VS collisions" :

1. $\delta = h_1 - p - (k+h-p)H(\frac{\varepsilon_1}{k+h-p}) = \frac{2}{3}h - p - (k+h-p)H(\frac{\varepsilon_1}{k+h-p})$
2. $\delta' = h_{1,1} - f_1(p) - (k+h-f_1(p))H(\frac{\varepsilon_2}{k+h-f_1(p)}) = \frac{1}{3}h - f_1(p) - (k+h-f_1(p))H(\frac{\varepsilon_2}{k+h-f_1(p)})$

On s'en sert pour calculer l'expression de T_{BJMMQW}, $\tilde{O}(2^{\frac{\delta}{2}+\delta'}T)$.

$$\tfrac{\delta}{2} + \delta' + \log_2(T)$$
$$\tfrac{1}{2}\left(\tfrac{2}{3}h - p - (k+h-p)H(\tfrac{\varepsilon_1}{k+h-p})\right) + \left(\tfrac{1}{3}h - f_1(p) - (k+h-f_1(p))H(\tfrac{\varepsilon_2}{k+h-f_1(p)})\right) + \tfrac{1}{3}h$$
$$h - p - \varepsilon_1 - \tfrac{k+h-p}{2}H(\tfrac{\varepsilon_1}{k+h-p}) - (k+h-f_1(p))H(\tfrac{\varepsilon_2}{k+h-f_1(p)})$$
$$3\log_2(T) + \tfrac{\nu(k,p,h,\varepsilon_1,\varepsilon_2)}{2}$$

Ainsi, $T_{BJMMQW} = T^3 2^{\frac{\nu(k,p,h,\varepsilon_1,\varepsilon_2)}{2}} = \binom{k+h}{f_2(p)}^{4/3} 2^{\frac{\nu(k,p,h,\varepsilon_1,\varepsilon_2)}{2}}$.

On calcule enfin la complexité totale, $\sqrt{P_{BJMMQW}^{-1}} T_{BJMMQW}$:

$$\sqrt{\frac{\binom{n}{t}}{\binom{k+h}{p}\binom{n-k-h}{t-p}}} \binom{k+h}{f_2(p)}^{4/3} 2^{\frac{\nu(k,p,h,\varepsilon_1,\varepsilon_2)}{2}}$$

$$\sqrt{\frac{\binom{n}{t}}{\binom{k+h}{p}\binom{n-k-h}{t-p}}} \binom{k+h}{f_2(p)}^{8/3} 2^{\nu(k,p,h,\varepsilon_1,\varepsilon_2)}$$

□

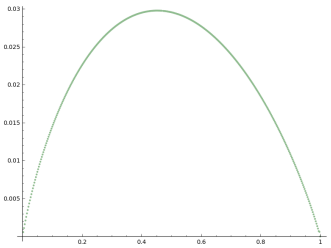
5.3.3 Algorithme $BJMMQW^*$

On va donner ici un algorithme moins efficace mais plus simple que $BJMMQW$, le but étant de réduire le nombre de paramètres pour pouvoir simplifier l'analyse (surtout à l'aide de développements limités). Ainsi, on définit l'algorithme $BJMMQW^*$ obtenu à partir de $BJMMQW$ en supprimant le recours à $1+1=0$, c'est-à-dire où $\epsilon_1 = \epsilon_2 = 0$. Dans ce cas, on obtient l'expression simplifiée suivante :

$$\alpha_{BJMMQW^*}(R) = \min_{\pi,\eta,\epsilon_1,\epsilon_2} \left(\frac{H(\tau) - (1-R-\eta) H(\frac{\tau-\pi}{1-R-\eta}) - (R+\eta) H(\frac{\pi}{R+\eta}) + \frac{8(R+\eta)}{3} H(\frac{\pi}{4(R+\eta)}) - 2\pi}{2} \right) \text{ avec :}$$

$$\begin{gathered} \text{Soumise aux contraintes :} \\ \pi = 4(R+\eta) H^{-1}(\tfrac{3\eta}{4(R+\eta)}) \\ 0 \leq \pi \leq \min(\tau, R+\eta) \\ 0 \leq \eta \leq 1 - R - \tau + \pi \leq 1 - R \end{gathered}$$

Et $\max_R(\alpha_{BJMMQW^*}) = 0.0595337$ pour $R = 0.4537$, valeur obtenue pour $\pi \approx 0.005439695$ et $\eta \approx 0.0179120898$



Courbe de α_{BJMMQW^*} en fonction de R

5.3.4 Explication du résultat

On voit ainsi que $BJMMQW$ est plus lent que $MMTQW$. Les considérations suivantes permettent d'esquisser une explication de ce résultat qui arrête la progression descendante de la complexité à laquelle on s'attendait et qui est donc a priori surprenant.

Dans [1] (chapitre 10, sections 2 et 3), on trouve une analyse et une comparaison

des complexités spatiales et temporelles des algorithmes classiques MMT et de $BJMM$ ainsi que les variantes "hybrides" auxquelles on s'est intéressées dans le but de construire les versions quantiques avec Quantum Walk. Dans [1], l'optimisation des variantes hybrides est faite dans le but d'optimiser l'espace mémoire alors que le souci du présent travail était de réunir les conditions optimales pour le déroulement de la marche aléatoire, par conséquent les résultats obtenus dans [1] ne se transposent pas immédiatement à notre cas.

Il est toutefois possible d'en tirer des conséquences qui restent pertinentes dans une certaine mesure. En effet, la variante hybride de MMT peut être optimisée de sorte à avoir gagner considérablement en mémoire, alors que le gain en mémoire qui peut être obtenu avec la variante hybride de $BJMM$ est comparablement négligeable. De plus, en regardant de plus près les détails, on voit que le gain en mémoire pour la variante hybride de $BJMM$ est obtenue pour $\delta = 0$, choix que l'on ne peut pas se permettre en quantique. Cela s'explique par le raisonnement suivant : lorsque $\delta > 0$ ou $\delta' > 0$, il faut stocker chaque liste du niveau concerné 2^{δ} resp. $2^{\delta'}$ fois pour tous les choix du vecteur intermédiaire, ce qui a pour effet d'augmenter la mémoire. Les listes de base sont très petites. Par conséquent, le fait de les stocker $2^{\delta'}$ fois ne gêne pas trop. Par contre, les listes du niveau supérieur sont déjà trop grandes et on n'a aucun intérêt à les stocker autant de fois.

Cela peut expliquer pourquoi $BJMMQW$ est plus lent que $MMTQW$. Premièrement, au lieu d'avoir un seul vecteur intermédiaire à chercher, on en a trois dont deux qu'il faut chercher sous forme de couple. Deuxièmement, la taille des listes est plus grande.

5.4 Retour aux sources : algorithme $MMT^\sharp$

On va faire un pas en arrière. On reconsidère l'algorithme MMT^*, où le but était d'appliquer la technique des réprésentations étendues ($1+1=0$) à un seul niveau. Dans MMT^*, on avait choisi ε de sorte à avoir le poids du vecteur final égal à p en moyenne et on avait vu que cela n'améliorait pas vraiment le temps d'exécution asymptotique.

On va maintenant enlever cette contrainte sur ε. On notera l'algorithme résultant $MMT^\sharp$: on peut le voir comme une version amortie de $BJMM$ avec $\varepsilon_2 = 0$ et de fait, son temps d'exécution est considérablement meilleur que celui de MMT mais un peu pire que celui de $BJMM$.

Bien évidemment, il s'agit ici comme pour MMT d'utiliser la version "hybride" avec $h_1 > p$.

5.4.1 Version classique

Théorème 28. *Si H est une matrice aléatoire, en notant $R := \frac{k}{n}$, $\tau := \frac{t}{n}$, $\pi := \frac{p}{n}$, $\eta = \frac{h}{n}$, et $\epsilon = \frac{\varepsilon}{n}$, on a l'expression suivante pour l'exposant $\alpha_{MMT^\sharp}$:*

$$\alpha_{MMT^\sharp}(R) = \min_{\pi,\eta,\epsilon} \left(\beta(R,\pi,\eta) + \max_{i=1,2,3} \gamma_i(R,\eta,\pi,\epsilon)\right) \text{ avec :}$$

$$\boxed{\beta(R,\pi,\eta) = H(\tau) - (1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta}) - (R+\eta)H(\frac{\pi}{R+\eta})}$$

$$\boxed{\gamma_1(R,\eta,\pi,\epsilon) = \frac{R+\eta}{2}H(\frac{\pi/2+\epsilon}{R+\eta})}$$

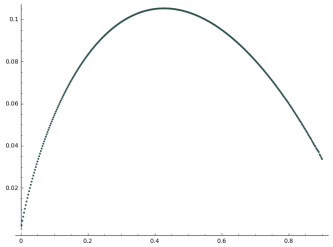
$$\boxed{\gamma_2(R,\eta,\pi,\epsilon) = (R+\eta)H(\frac{\pi/2+\epsilon}{R+\eta}) - \pi - (1-R-\eta)H(\frac{\epsilon}{1-R-\eta})}$$

$$\boxed{\gamma_3(R,\eta,\pi,\epsilon) = 2(R+\eta)H(\frac{\pi/2+\epsilon}{R+\eta}) - (1-R-\eta)H(\frac{\epsilon}{1-R-\eta}) - \pi - \eta}$$

Soumise aux contraintes :

$$0 \leq \epsilon \leq \pi$$
$$0 \leq \pi \leq \min(\tau,\eta) \leq R+\eta$$
$$0 \leq \eta \leq 1-R-\tau+\pi \leq 1-R$$

Et $\max_R(\alpha_{MMT^\sharp}) = 0.1052837$ *pour* $R = 0.4271$, *valeur obtenue pour* $\eta \approx 0.131441089$, $\pi \approx 0.036218826$ *et* $\epsilon \approx 0.003388306$.



Courbe de $\alpha_{MMT^\sharp}$ *en fonction de* R

Démonstration. Il s'agit des mêmes calculs que pour $MMT*$, à cette différence près qu'il n'y a plus de contrainte sur ϵ. □

5.4.2 Version quantique avec Quantum Walk

THÉORÈME 29. *Si* H *est une matrice aléatoire, en notant* $R := \frac{k}{n}$, $\tau := \frac{t}{n}$, $\pi := \frac{p}{n}$, $\eta = \frac{h}{n}$, *et* $\epsilon = \frac{\varepsilon}{n}$, *on a l'expression suivante pour l'exposant* $\alpha_{MMTQW^\sharp}$ *:*

$$\alpha_{MMTQW^\sharp}(R) = \min_{\pi,\eta,\varepsilon}\left(\frac{H(\tau)-(1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta})-(R+\eta)H(\frac{\pi}{R+\eta})+\mu(R,\eta,\pi,\epsilon)}{2}\right)$$

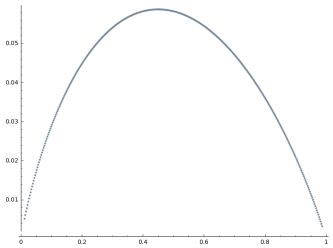
Avec :

$$\boxed{\mu(R,\eta,\pi,\epsilon) = \frac{3(R+\eta)}{5}H(\frac{\pi/2+\epsilon}{R+\eta}) - \pi - (1-R-\eta)H(\frac{\epsilon}{1-R-\eta}))}$$

Soumise aux contraintes :

$$0 \leq \epsilon \leq \pi$$
$$\pi = 2\left((R+\eta)H^{-1}(\tfrac{5\eta}{4(R+\eta)}) - \epsilon\right)$$
$$0 \leq \pi \leq \min(\tau, R+\eta)$$
$$0 \leq \eta \leq 1-R-\tau+\pi \leq 1-R$$

Et $\max_R\left(\alpha_{MMTQW^\sharp}\right) = 0.0586953$ *pour* $R = 0.4514$, *valeur obtenue pour* $\eta \approx 0.0375776$, $\pi \approx 0.0107$ *et* $\epsilon \approx 0.0006935$.



Courbe de $\alpha_{MMTQW^\sharp}$ en fonction de R

Démonstration. Il s'agit des mêmes calculs que pour $MMT*$, à cette différence près qu'il n'y a plus de contrainte sur ϵ. □

6 Algorithme de May et Ozerov

Jusqu'ici, on a procédé de la manière suivante : on considérait un sous-problème de décodage par syndrome et on trouvait une liste de ses solutions à l'aide de la recherche de collisions ou la technique des représentations ou les deux, un nombre variable de fois. Puis, on cherchait parmi les éléments de cette liste, solutions au sous-problème, celui qui, après quelques opérations, permettait de résoudre le problème principal. L'algorithme de May et Ozerov adopte une autre approche : étant donnés deux listes L_1 et L_2, que l'on a généré directement ou par collisions/représentations, trouver directement la solution du problème principal. En termes plus formels, il s'agit de résoudre le problème du voisin le plus proche :

Problème 8 (Nearest Neighbour de paramètres (m, γ, λ)). *Soit $m \in \mathbb{N}$, $0 < \gamma < \frac{1}{2}$ et $0 < \lambda < 1$. Étant donné deux listes L_1, L_2 de taille égale $|L_1| = |L_2| = 2^{\lambda m}$ de vecteurs uniformément choisis et indépendants deux-à-deux de $\mathbb{F}_2^m$, s'il existe $(u^*, v^*) \in L_1 \times L_2$ tel que $w_H(u^* + v^*) = \gamma m$, donner une liste C contenant (u^*, v^*).*

Notons qu'un algorithme naïf pour trouver (u^*, v^*) consisterait à tester, pour tous les couples $(u, v) \in L_1 \times L_2$ s'ils résolvent du problème, i.e. si $w_H(u + v) = \gamma m$. Autrement dit, on est à la recherche d'un élément parmi $2^{2\lambda m}$. Ainsi, l'algorithme naïf de résolution du problème coûte $O(2^{2\lambda m})$ dans le cas classique et $O(2^{\lambda m})$ dans le cas quantique avec l'algorithme de Grover.

6.1 Version classique

6.1.1 Algorithme de résolution du problème Nearest Neighbour

On va donner deux algorithmes : le premier, $NearestNeighbour$, définit quelques paramètres et initialise un environnement et fait un premier appel au second algorithme, $NearestNeighbourRec$, qui, comme son nom l'indique, est un algorithme récursif et fait le travail principal de construction de la liste C, en se basant sur le constat suivant : si on divise longueur m du vecteur en t bandes de longueur $\alpha_1 m, ..., \alpha_t m$, alors si on a $w_H(u^* + v^*) = \gamma m$, on a aussi, éventuellement après avoir appliqué une même permutation aléatoire aux coordonnées des vecteurs, que $w_H(u_i^* + v_i^*) = \gamma \alpha_i m$ si w_i désigne la restriction du vecteur w à la bande de longueur $\alpha_i m$. Autrement dit, la propriété globale

sur le poids vaut aussi localement sur chaque bande, à quelques nuances près.

Algorithme 17 : *NearestNeighbour*

Entrées : L_1, L_2, m, γ, h, λ, ε
Sorties : Liste C contenant la solution $(u^*, v^*) \in L_1 \times L_2$

$y \leftarrow (1-\gamma)\left(1 - H\left(\frac{H^{-1}(1-\lambda)-\frac{\gamma}{2}}{1-\gamma}\right)\right)$
$t \leftarrow \left\lceil \frac{\log_2\left(y-\lambda+\frac{\varepsilon}{2}\right)-\log_2\left(\frac{\varepsilon}{2}\right)}{\log_2(y)-\log_2(\lambda)} \right\rceil$
$\alpha_1 = \frac{1-\frac{\lambda}{y}}{1-\left(\frac{\lambda}{y}\right)^t}$
pour $j = 2$ *à* t **faire**
    $\alpha_j \leftarrow \frac{y}{\lambda}\alpha_{j-1}$
pour *poly*(m) *permutations uniformément aléatoires* π *de* $\{1, ..., m\}$ **faire**
    pour *poly*(m) *vecteurs uniformément aléatoires* r *ayant poids* $\frac{\alpha_j m}{2}$ *sur chaque bande de longueur* $\alpha_j m$ **faire**
        $\overline{L_i} \leftarrow \pi(L_i) + r$ pour $i = 1, 2$
        Enlever des $\overline{L_i}$ les vecteurs de poids différent de $\frac{\alpha_j m}{2}$ sur chaque bande
        retourner $NearestNeighbourRec(\overline{L_1}, \overline{L_2}, m, t, \gamma, \lambda, (\alpha_j)_{j=1,..,t}, y, h, \varepsilon, 1)$

Algorithme 18 : *NearestNeighbourRec*

Entrées : $L_1^{(j-1)}, L_2^{(j-1)}, m, t, \gamma, \lambda, (\alpha_j)_{j=1,..,t}, y, h, \varepsilon, 1$
Sorties : Liste C contenant la solution $(u^*, v^*) \in L_1 \times L_2$

$C \leftarrow \emptyset$
si $j = t + 1$ **alors**
    `/* Cas de base */`
    $C \leftarrow \{(u, v) \in L_1^{(j)} \times L_2^{(j)} \mid w_H(u + v) = \gamma m\}$
sinon
    pour $\tilde{O}\left(2^{y\alpha_j m}\right)$ *tours de boucle* **faire**
        `/* Cas général */`
        $A_j = A_{j,1} \sqcup A_{j,2} \leftarrow$ partition de la bande de longueur $\alpha_j m$ en deux parties égales
        pour $i = 1, 2$ **faire**
            $L_i^{(j)} \leftarrow \{v_{L_i^{(j-1)}} \in L_i^{(j-1)}$ de poids $\frac{\alpha_j h m}{2}$ sur $A_{j,1}\}$
        si $|L_i^{(j)}| \lessapprox \tilde{O}\left(2^{\lambda m(1-\sum_{i=1}^{j}\alpha_i+\frac{\varepsilon}{2})}\right)$ **alors**
            $C \leftarrow C \cup NearestNeighbourRec(L_1^{(j)}, L_2^{(j)}, m, t, \gamma, \lambda, (\alpha_j)_{j=1,..,t}, y, h, \varepsilon, j+1)$

On va prouver la correction de ces algorithmes à l'aide de six lemmes qui interviendront dans la preuve d'un théorème final.

Le premier lemme montre qu'il est fort probable que d'abord la propriété $w_H(u_i^* + v_i^*) = \gamma\alpha_i m$ sur toutes les bandes soit vérifiée, puis que la propriété $w_H(u_j^*) =$

$w_H(v_j^*) = \frac{\alpha_i m}{2}$ est vérifiée avec une probabilité importante aussi, ce qui fait qu'on ne vide pas trop les deux listes dans des boucles de randomisation de l'algorithme $NearestNeighbour$.

LEMME 1. *Soit (L_1, L_2, γ) une instance d'un problème Nearest Neighbour de paramètres (m, γ, λ) dont une solution inconnue est $(u^*, v^*) \in L_1 \times L_2$. On écrit $u^* = (u_1^*, ..., u_t^*)$, $v^* = (v_1^*, ..., v_t^*)$ avec chaque v_j^*, $1 \leq j \leq t$ sur une bande de longueur $\alpha_j m$, $\sum_{j=1}^{t} \alpha_j = 1$. Alors, il est vrai avec une probabilité inversement polynomiale que dans au moins une des listes après la permutation uniforme et l'ajout d'un vecteur r uniformément aléatoire sur chaque bande :*

1. $w_H(u_j^* + v_j^*) = \gamma \alpha_j m$
2. $w_H(u_j^*) = w_H(v_j^*) = \frac{\alpha_j m}{2}$

$\forall\ 1 \leq j \leq t$.

Démonstration. La probabilité d'avoir $w_H(u_j^* + v_j^*) = \gamma \alpha_j m\ \forall 1 \leq j \leq t$ après une permutation aléatoire est :

$$\frac{\binom{\alpha_1 m}{\gamma \alpha_1 m} \cdots \binom{\alpha_t m}{\gamma \alpha_t m}}{\binom{m}{\gamma m}}$$

$$=$$

$$\tilde{O}\left(\frac{2^{m \sum_{j=1}^{t} \alpha_j H(\gamma)}}{2^{mH(\gamma)}}\right)$$

$$=$$

$$\tilde{O}\left(2^{mH(\gamma) - mH(\gamma)}\right)$$

$$=$$

$$\tilde{O}(1)$$

Le vecteur $r = (r_1, ..., r_t)$ de la seconde boucle est choisie uniformément et de sorte à être de poids $\frac{\alpha_j m}{2}$ sur chaque bande. Si, pour un j donné, on subdivise la bande de longueur $\alpha_j m$ en quatre parties $K_{x,y}$ de la manière suivante :

$K_{xy} := \{i \in \{(\alpha_1 + ... + \alpha_{j-1})m + 1, ..., (\alpha_1 + ... + \alpha_j)m\} \mid u_i^* = x, v_i^* = y\}$, et on note $k_{x,y} = \#K_{x,y}$,
on va considérer comme "bons" les vecteurs r qui sont de poids $\frac{k_{x,y}}{2}$ sur chaque partie. Alors, on utilise le fait que $\sum_{x,y} k_{xy} = \alpha_j m$ pour calculer la proportion de bons vecteurs r :

$$\frac{\binom{k_{00}}{\frac{k_{00}}{2}} \binom{k_{01}}{\frac{k_{01}}{2}} \binom{k_{10}}{\frac{k_{10}}{2}} \binom{k_{11}}{\frac{k_{11}}{2}}}{\binom{\alpha_j m}{\frac{\alpha_j m}{2}}}$$

$$\approx$$

$$\frac{2^{\sum_{x,y} k_{x,y} H(\frac{1}{2})}}{2^{\alpha_j m H(\frac{1}{2})}}$$

$$\approx$$

De plus, par définition des $K_{x,y}$, u_j^* est de poids k_{10} sur K_{01}, respectivement k_{11} sur K_{11} et de poids 0 sur K_{01} et K_{00}. Par conséquent, puisque r_j est de poids $\frac{k_{xy}}{2}$ sur K_{xy}, $x, y = 0, 1$, on a $w_H(u_j^* + r_j) = \frac{1}{2}(k_{00}+k_{01}+k_{10}+k_{11}) = \frac{\alpha_j m}{2}$.

Par un argument analogue, on a aussi $w_H(v_j^* + r_j) = \frac{1}{2}(k_{00} + k_{01} + k_{10} + k_{11}) = \frac{\alpha_j m}{2}$. □

DÉFINITION 5. *Soit $(u^*, v^*) \in L_1 \times L_2$ la solution cible. On dit qu'un chemin de la racine jusqu'à une feuille de l'arbre de calcul de NearestNeighbour est bon si toutes les paires de sous-listes à chaque noeud contiennent (u^*, v^*).*

Le lemme suivant porte sur la probabilité importante que les listes construites et gardées à chaque étape par l'algorithme $NearestNeighbourRec$ contiennent la solution cible. On y prouve notamment la nécessité de tester $\tilde{O}\left(2^{y\alpha_j m}\right)$ à chaque étape de la récursion.

LEMME 2. *Soit $t \in \mathbb{N}$ la hauteur de l'arbre de calcul de NearestNeighbour. Alors cet arbre de calcul comporte un bon chemin avec une probabilité pas trop faible (inversement polynomiale).*

Démonstration. Par récurrence.

1) Par définition de L_1 et de L_2, $(u^*, v^*) \in L_1 \times L_2 = L_1^{(j)} \times L_2^{(j)}$.

2) Montrons que si la cible (u^*, v^*) est dans une des paires de listes $L_1^{(j)} \times L_2^{(j)}$ données en entrée à l'algorithme au niveau j, alors elle est dans les listes de sortie (qui sont les listes d'entrée du niveau $j+1$) avec une probabilité inversement polynomiale.

Pour ce faire, on considère le niveau j : au moins une des paires de listes d'entrée contient (u^*, v^*). On écrit $u^* = (u_1^*, ..., u_t^*), v^* = (v_1^*, ..., v_t^*)$ et on se concentre sur u_j^* et v_j^* qui sont de longueur $\alpha_j m$. D'après lemme 1 (page 79), $w_H(u_j^*) = w_H(v_j^*) = \frac{\alpha_j m}{2}$ et $w_H(u_j^* + v_j^*) = \gamma\alpha_j m$.
Par conséquent, si l'on note $k_{xy} := \#\{i \in \{(\alpha_1 + ... + \alpha_{j-1})m + 1, ..., (\alpha_1 + ... + \alpha_j)m\} \mid u_i^* = x, v_i^* = y\}$, on aura :
$k_{01} = k_{10} = \frac{\gamma m}{2}$
$k_{00} = k_{11} = \frac{(1-\gamma)m}{2}$
$\sum_{x,y} k_{x,y} = \alpha_j m$
L'algorithme choisit un sous-ensemble $A \subset \{(\alpha_1 + ... + \alpha_{j-1})m + 1, ..., (\alpha_1 + ... + \alpha_j)m\}, |A| = \frac{\alpha_j m}{2}$ pour partitionner la bande de longueur $\alpha_j m$ en deux parties égales et construit les listes $L_{1,A}^{(j)}$ et $L_{2,A}^{(j)}$ qui contiennent les vecteurs de poids $\frac{hm}{2}$ sur les colonnes indicées par A. Si l'on note $k_{xy,A} := \#\{i \in A \mid u_i^* = x, v_i^* = y\}$, on a alors :

$$\begin{cases} k_{01,A} + k_{11,A} = \frac{hm}{2} \\ k_{10,A} + k_{11,A} = \frac{hm}{2} \\ k_{00,A} + k_{01,A} + k_{10,A} + k_{11,A} = \frac{m}{2} \end{cases}$$

Ce qui équivaut à :

$$\begin{cases} k_{01,A} = k_{10,A} = \frac{cm}{2} \\ k_{11,A} = \frac{(h-c)m}{2} \\ k_{00,A} = \frac{(1-h-c)m}{2} \end{cases} \quad \text{pour un paramètre } c < h$$

Une bonne partition qui fait que (u^*, v^*) est conservé dans les listes de sortie est donc une partition qui débouche sur une distribution de cette forme des valeurs des composantes de u^* et de v^*. Le nombre total des bonnes partitions est donc la somme sur toutes les valeurs possibles que peut prendre c du nombre de bonnes partitions pour une valeur de c fixée :

$$\sum_{c\alpha_j \frac{m}{2}=0}^{h\alpha_j \frac{m}{2}} \binom{\frac{1-\gamma}{2}\alpha_j m}{(1-h-c)\alpha_j \frac{m}{2}} \binom{\frac{\gamma}{2}\alpha_j m}{c\alpha_j \frac{m}{2}}^2 \binom{\frac{1-\gamma}{2}\alpha_j m}{(h-c)\alpha_j \frac{m}{2}}$$

Or, l'ordre de grandeur de cette expression est :

$$poly\left(\max_c\left(\binom{\frac{1-\gamma}{2}\alpha_j m}{(1-h-c)\alpha_j \frac{m}{2}} \binom{\frac{\gamma}{2}\alpha_j m}{c\alpha_j \frac{m}{2}}^2 \binom{\frac{1-\gamma}{2}\alpha_j m}{(h-c)\alpha_j \frac{m}{2}}\right)\right)$$

On cherche donc la valeur de c qui maximise le terme ainsi que la valeur maximale prise par le terme. Notons que, pour que les termes binomiaux soient bien définis, il faut $c \leq \min(h, 1-h, \gamma)$.

$$\binom{\frac{1-\gamma}{2}\alpha_j m}{(1-h-c)\alpha_j \frac{m}{2}} \binom{\frac{\gamma}{2}\alpha_j m}{c\alpha_j \frac{m}{2}}^2 \binom{\frac{1-\gamma}{2}\alpha_j m}{(h-c)\alpha_j \frac{m}{2}} \approx 2^{\alpha_j m\left(\frac{1-\gamma}{2}H\left(\frac{1-h-c}{1-\gamma}\right)+\gamma H\left(\frac{c}{\gamma}\right)+\frac{1-\gamma}{2}H\left(\frac{h-c}{1-\gamma}\right)\right)}$$

Posons $f(c) := \frac{1-\gamma}{2}H\left(\frac{1-h-c}{1-\gamma}\right) + \gamma H\left(\frac{c}{\gamma}\right) + \frac{1-\gamma}{2}H\left(\frac{h-c}{1-\gamma}\right)$.
On pose :

1. $f_1(c) = \frac{1-h-c}{1-\gamma} \Rightarrow f_1'(c) = -\frac{1}{1-\gamma}$
2. $f_2(c) = \frac{c}{\gamma} \Rightarrow f_2'(c) = \frac{1}{\gamma}$
3. $f_3(c) = \frac{h-c}{1-\gamma} \Rightarrow f_3'(c) = -\frac{1}{1-\gamma}$

On a alors $f(c) = \frac{1-\gamma}{2}H(f_1(c)) + \gamma H(f_2(c)) + \frac{1-\gamma}{2}H(f_3(c))$.

On rappelle que $H'(x) = \log_2\left(\frac{1-x}{x}\right)$ (et donc, pour une fonction quelconque F, $H(F(x)) = F'(x)\log_2\left(\frac{1-F(x)}{F(x)}\right)$).

Par conséquent :

$$\begin{aligned} f'(c) &= \frac{1-\gamma}{2}\left(-\frac{1}{1-\gamma}\right)\log_2\left(\frac{1-\frac{1-h-c}{1-\gamma}}{\frac{1-h-c}{1-\gamma}}\right) + \gamma\left(\frac{1}{\gamma}\right)\log_2\left(\frac{1-\frac{c}{\gamma}}{\frac{c}{\gamma}}\right) + \frac{1-\gamma}{2}\left(-\frac{1}{1-\gamma}\right)\log_2\left(\frac{1-\frac{h-c}{1-\gamma}}{\frac{h-c}{1-\gamma}}\right) \\ &= \log_2\left(\frac{\gamma-c}{c}\right) - \frac{1}{2}\left(\log_2\left(\frac{1-\gamma-1+h+c}{1-h-c}\right) + \log_2\left(\frac{1-\gamma-h+c}{h-c}\right)\right) \\ &= \log_2\left(\frac{\gamma-c}{c}\right) - \frac{1}{2}\left(\log_2\left(\frac{c-\gamma+h}{1-h-c}\right) + \log_2\left(\frac{c-\gamma+1-h}{h-c}\right)\right) \end{aligned}$$

Alors, $f'(c) = 0$ équivaut à :

$$2\log_2\left(\frac{\gamma-c}{c}\right) = \log_2\left(\frac{c-\gamma+h}{1-h-c}\right) + \log_2\left(\frac{c-\gamma+1-h}{h-c}\right)$$

$$\left(\frac{\gamma-c}{c}\right)^2 = \frac{(c-\gamma)+h}{1-h-c}\frac{(c-\gamma)+1-h}{h-c}$$

$$\frac{(1-h-c)(h-c)}{c^2} = \frac{(c-\gamma)^2+(1-h)(c-\gamma)+h(c-\gamma)+h(1-h)}{(\gamma-c)^2}$$

$$\frac{h-c-h^2+hc-hc+c^2}{c^2} = 1-\frac{1}{\gamma-c}+\frac{h(1-h)}{(\gamma-c)^2}$$

$$h-c-h^2+c^2 = c^2-\frac{c^2}{\gamma-c}+c^2\frac{h(1-h)}{(\gamma-c)^2}$$

$$c^2-\frac{c^2}{\gamma-c}+c^2\frac{h(1-h)}{(\gamma-c)^2}-h+c+h^2-c^2 = 0$$

$$(\gamma-c)^2\left(-h+c+h^2\right)-(\gamma-c)c^2+c^2h(1-h) = 0$$

$$(\gamma^2-2\gamma c+c^2)\left(-h+c+h^2\right)-\gamma c^2+c^3+c^2h-c^2h^2 = 0$$

$$(-\gamma^2h+\gamma^2c+\gamma^2h^2+2\gamma hc-2\gamma c^2-2\gamma h^2c-c^2h+c^3+c^2h^2)-\gamma c^2+c^3+c^2h-c^2h^2 = 0$$

$$-\gamma^2h+\gamma^2c+\gamma^2h^2+2\gamma hc-3\gamma c^2-2\gamma h^2c+2c^3 = 0$$

$$c^3(2)+c^2(-3\gamma)+c\left(\gamma^2+2\gamma h-2\gamma h^2\right)+(-\gamma^2h+\gamma^2h^2) = 0$$

On pose $P(X) := X^3(2)+X^2(-3\gamma)+X\left(\gamma^2+2\gamma h-2\gamma h^2\right)+(-\gamma^2h+\gamma^2h^2)$
Est-ce que $\frac{\gamma}{2}$ est racine de ce polynome ?

$c \leftarrow \frac{\gamma}{2}$
$\Rightarrow$

$$\begin{aligned}
P(c) &= \left(\frac{\gamma}{2}\right)^3(2)+\left(\frac{\gamma}{2}\right)^2(-3\gamma)+\left(\frac{\gamma}{2}\right)\left(\gamma^2+2\gamma h-2\gamma h^2\right)+(-\gamma^2h+\gamma^2h^2)\\
&= \frac{\gamma^3}{8}(2)+\frac{\gamma^2}{4}(-3\gamma)+\frac{\gamma}{2}\left(\gamma^2+2\gamma h-2\gamma h^2\right)+(-\gamma^2h+\gamma^2h^2)\\
&= \frac{\gamma^3}{4}-\frac{3\gamma^3}{4}+\frac{\gamma^3}{2}+\gamma^2h-\gamma^2h^2-\gamma^2h+\gamma^2h^2\\
&= -\frac{2\gamma^3}{4}+\frac{\gamma^3}{2}\\
&= 0
\end{aligned}$$

Factorisation de $P(X) = 2(X-\frac{\gamma}{2})(X^2-\gamma X+\gamma(h(1-h)) = 2(X-\frac{\gamma}{2})(X-g_+)(X-g_-)$
Avec $g_\pm = \frac{1}{2}\left(\gamma \pm \sqrt{4\gamma h^2+\gamma^2-4\gamma h}\right)$

On a alors deux cas de figures :

1. Si $g_\pm \in \mathbb{R}$, puisque $g_- < \frac{\gamma}{2} < g_+$, c'est au point $c = \frac{\gamma}{2}$ que la fonction f atteint un maximum local.
2. Si $g_\pm \in \mathbb{C}\setminus\mathbb{R}$, puisque la fonction $x \mapsto x^2-\gamma x+\gamma(h(1-h))$ est positif, f atteint un point selle au point $c = \frac{\gamma}{2}$ (!)

On en déduit que le nombre de bonnes partitions est :

$$poly\left(\binom{\frac{1-\gamma}{2}\alpha_j m}{(1-h-\frac{\gamma}{2})\alpha_j\frac{m}{2}}\binom{\frac{\gamma}{2}\alpha_j m}{\frac{\gamma}{4}\alpha_j m}^2\binom{\frac{1-\gamma}{2}\alpha_j m}{(h-\frac{\gamma}{2})\alpha_j\frac{m}{2}}\right)$$

Par conséquent, la proportion des bonnes partitions parmi toutes les partitions possibles est de l'ordre de :

$$poly\left(\frac{\binom{\frac{1-\gamma}{2}\alpha_j m}{(1-h-\frac{\gamma}{2})\alpha_j\frac{m}{2}}\binom{\frac{\gamma}{2}\alpha_j m}{\frac{\gamma}{4}\alpha_j m}^2\binom{\frac{1-\gamma}{2}\alpha_j m}{(h-\frac{\gamma}{2})\alpha_j\frac{m}{2}}}{\binom{\alpha_j m}{\frac{\alpha_j m}{2}}}\right)$$

i.e.

$$\tilde{O}\left(2^{\alpha_j m\left(\frac{1-\gamma}{2}H\left(\frac{1-h-\frac{\gamma}{2}}{1-\gamma}\right)+\gamma H\left(\frac{1}{2}\right)+\frac{1-\gamma}{2}H\left(\frac{h-\frac{\gamma}{2}}{1-\gamma}\right)\right)-\alpha_j m H\left(\frac{1}{2}\right)}\right)$$

$$\tilde{O}\left(2^{\alpha_j m\left(\frac{1-\gamma}{2}H\left(\frac{1-\gamma-h+\frac{\gamma}{2}}{1-\gamma}\right)+\gamma+\frac{1-\gamma}{2}H\left(\frac{h-\frac{\gamma}{2}}{1-\gamma}\right)-1\right)}\right)$$

$$\tilde{O}\left(2^{\alpha_j m\left(\frac{1-\gamma}{2}H\left(1-\frac{h-\frac{\gamma}{2}}{1-\gamma}\right)+\frac{1-\gamma}{2}H\left(\frac{h-\frac{\gamma}{2}}{1-\gamma}\right)+\gamma-1\right)}\right)$$

$$\tilde{O}\left(2^{\alpha_j m\left((1-\gamma)\left(H\left(\frac{h-\frac{\gamma}{2}}{1-\gamma}\right)-1\right)\right)}\right)$$

En posant $y = (1-\gamma)\left(1-H\left(\frac{h-\frac{\gamma}{2}}{1-\gamma}\right)\right)$, on a alors une proportion $\tilde{O}\left(2^{-y\alpha_j m}\right)$ de bonnes partitions. Par conséquent, au bout de $\tilde{O}\left(2^{y\alpha_j m}\right)$ répétitions de choix aléatoire de la partition A, on tombe sur la bonne partition avec une très forte probabilité. □

Le lemme suivant prouve que le choix d'imposer une limite de taille sur les listes gardées est pertinente.

LEMME 3. *Soit $t \in \mathbb{N}$ la hauteur de l'arbre de calcul de NearestNeighbour et $\varepsilon > 0$. Alors, parmi les $2t$ listes aux noeuds d'un bon chemin, chacune au niveau j est de taille $\tilde{O}\left(2^{\lambda m(1-\sum_{i=1}^{j}\alpha_i)+\frac{\varepsilon}{2}}\right)$ avec une probabilité inversement polynomiale.*

Démonstration. On fera l'analyse pour L_1, l'analyse pour L_2 est analogue.

On a $|L_1| = 2^{\lambda m}$. On fixe un niveau j et on définit la variable aléatoire suivante pour toute $v_k \in L_1$:

$$X_k = \begin{cases} 1 \text{ si } v_k \in L_1^{(i)} & \forall\ 1 \leq i \leq j \\ 0 \text{ sinon.} \end{cases}$$

Alors, $X := \sum_{k=1}^{2^{\lambda m}} X_k$ donne le nombre d'éléments contenus dans la liste $L_1^{(j)}$.

Comme par hypothèse le chemin de calcul est bon, il existe $v_{k^*} \in L_1$, $\mathbb{P}(X_{k^*} = 1) = 1$. Tous les autres éléments dans $L_1^{(j)}$ sont choisis indépendamment de v_{k^*} et uniformément parmi les vecteurs de poids total $\frac{\alpha_i m}{2}$ sur chaque bande de

taille $\alpha_i m$, $1 \leq i \leq j$, et de poids $\frac{\alpha_i h m}{2}$ sur une moitié de chacune de ces bandes. On en déduit :

$$\begin{aligned}
\mathbb{P}(X_k = 1) &= \prod_{i=1}^{j} \frac{\binom{\frac{\alpha_i m}{2}}{\frac{\alpha_i h m}{2}} \binom{\frac{\alpha_i m}{2}}{\frac{\alpha_i (1-h) m}{2}}}{\binom{\alpha_i m}{\frac{\alpha_i m}{2}}} \\
&\approx \prod_{i=1}^{j} \frac{2^{\frac{\alpha_i m}{2} H(h)} 2^{\frac{\alpha_i m}{2} H(1-h)}}{2^{\alpha_i m H(\frac{1}{2})}} \\
&\approx \prod_{i=1}^{j} 2^{\alpha_i m H(h) - \alpha_i m} \\
&\approx 2^{m \sum_{i=1}^{j} \alpha_i (H(h)-1)}
\end{aligned}$$

On calcule maintenant la taille moyenne de la liste $L_1^{(j)}$:

$$\begin{aligned}
\mathbb{E}(X) &= \sum_{k=1}^{2^{\lambda m}} \mathbb{P}(X_k = 1) \\
&= \mathbb{P}(X_{k^*} = 1) + \sum_{k=1, k \neq k^*}^{2^{\lambda m}} \mathbb{P}(X_k = 1) \\
&\approx 1 + (2^{\lambda m} - 1) 2^{m \sum_{i=1}^{j} \alpha_i (H(h)-1)}
\end{aligned}$$

On rappelle l'inégalité de Tchebychev :

$$\mathbb{P}\left(|X - \mathbb{E}(X)| \geq \sqrt{\mathbb{V}(X)} k \right) \leq \frac{1}{k^2} \ \forall \ k$$

On remarque aussi que $\mathbb{V}(X) \leq \mathbb{E}(X)$. En effet :

$$\begin{aligned}
\mathbb{V}(X) &= \mathbb{V}\left(\sum_{k=1}^{2^{\lambda m}} X_k \right) \\
&= \sum_{k=1}^{2^{\lambda m}} \mathbb{V}(X_k) \text{ par indépendance 2-à-2 des } X_k \\
&= \sum_{k=1}^{2^{\lambda m}} \left(\mathbb{E}(X_k^2) - \mathbb{E}(X_k)^2 \right) \\
&\leq \sum_{k=1}^{2^{\lambda m}} \mathbb{E}(X_k) \text{ pourquoi ?} \\
&= \mathbb{E}(X)
\end{aligned}$$

Alors, en utilisant l'inégalité de Tchebychev avec $k = \frac{2^{\frac{\varepsilon}{2}m}\mathbb{E}(X)}{\sqrt{\mathbb{V}(x)}}$, on obtient :

$$\begin{aligned}
\mathbb{P}\left(|X - \mathbb{E}(X)| \geq \sqrt{\mathbb{V}(X)}\frac{2^{\frac{\varepsilon}{2}m}\mathbb{E}(X)}{\sqrt{\mathbb{V}(X)}}\right) &\leq \frac{\mathbb{V}(X)}{2^{\varepsilon m}\mathbb{E}(X)^2} \\
&\leq \frac{\mathbb{E}(X)}{2^{\varepsilon m}\mathbb{E}(X)^2} \\
&\leq \frac{2^{-\varepsilon m}}{\mathbb{E}(X)} \\
&\leq 2^{-\varepsilon m}
\end{aligned}$$

Par conséquent, la valeur de X est proche de celle de $\mathbb{E}(X)$ avec une très forte probabilité. On voudrait qu'il y ait un nombre au plus polynomial d'éléments dans la liste finale, i.e. $\mathbb{E}(X) = \tilde{O}(1)$
$\Leftrightarrow 1 + (2^{\lambda m} - 1)2^{m\sum_{i=1}^{j}\alpha_i(H(h)-1)} = \tilde{O}(1)$
$\Leftrightarrow 1 + 2^{m\left(\lambda+\sum_{i=1}^{j}\alpha_i(H(h)-1)\right)} - 2^{m\sum_{i=1}^{j}\alpha_i(H(h)-1)} = \tilde{O}(1)$
$\Leftrightarrow 2^{m\left(\lambda+\sum_{i=1}^{j}\alpha_i(H(h)-1)\right)} = \tilde{O}(1)$
$\Leftrightarrow \lambda + \sum_{i=1}^{j}\alpha_i\left(H(h) - 1\right) = 0$
$\Leftrightarrow \lambda = \sum_{i=1}^{j}\alpha_i\left(1 - H(h)\right) \leq 1 - H(h)$
$\Leftrightarrow H(h) \leq 1 - \lambda$
$\Leftrightarrow h \leq H^{-1}(1 - \lambda)$

Par conséquent, puisque (voir lemme 2 (page 80)) $\frac{\gamma}{2} \leq h \leq H^{-1}(1-\lambda)$, $H(\frac{\gamma}{2}) \leq 1 - \lambda$ et donc $\lambda \leq 1 - H(\frac{\gamma}{2})$. □

On aura besoin du lemme suivant dans la preuve du théorème final.

LEMME 4. *Soient* $0 < \gamma < \frac{1}{2}$ *et* $0 < \lambda < 1 - H(\frac{\gamma}{2})$. *Soit* $y = (1-\gamma)\left(1 - H\left(\frac{h-\frac{\gamma}{2}}{1-\gamma}\right)\right)$. *Alors* $y > \lambda$.

Démonstration. On pose $y_\lambda = (1-\gamma)\left(1 - H\left(\frac{H^{-1}(1-\lambda)-\frac{\gamma}{2}}{1-\gamma}\right)\right)$ et $f(\lambda) = y_\lambda - \lambda = (1-\gamma)\left(1 - H\left(\frac{H^{-1}(1-\lambda)-\frac{\gamma}{2}}{1-\gamma}\right)\right) - \lambda$ et on cherche à étudier les variations et le signe de f.

$$f'(\lambda) = -(1-\gamma)\log_2\left(\frac{1-\gamma}{H^{-1}(1-\lambda)-\frac{\gamma}{2}} - 1\right)\left(-\frac{1}{(1-\gamma)\log_2\left(\frac{1}{H^{-1}(1-\lambda)}-1\right)}\right) - 1$$

$$f'(\lambda) = \frac{\log_2\left(\frac{1-\gamma}{H^{-1}(1-\lambda)-\frac{\gamma}{2}}-1\right)}{\log_2\left(\frac{1}{H^{-1}(1-\lambda)}-1\right)} - 1$$

$$f'(\lambda) \leq 0$$
$$\Leftrightarrow$$
$$\frac{\log_2\left(\frac{1-\gamma}{H^{-1}(1-\lambda)-\frac{\gamma}{2}} - 1\right)}{\log_2\left(\frac{1}{H^{-1}(1-\lambda)} - 1\right)} \leq 1$$
$$\Leftrightarrow$$

$$\log_2\left(\frac{1-\gamma}{H^{-1}(1-\lambda)-\frac{\gamma}{2}}-1\right) \leq \log_2\left(\frac{1}{H^{-1}(1-\lambda)}-1\right)$$
$$\Leftrightarrow$$
$$\frac{1-\gamma}{H^{-1}(1-\lambda)-\frac{\gamma}{2}} \leq \frac{1}{H^{-1}(1-\lambda)}$$
$$\Leftrightarrow$$
$$(1-\gamma)H^{-1}(1-\lambda) \leq H^{-1}(1-\lambda)-\frac{\gamma}{2}$$
$$\Leftrightarrow$$
$$(1-\gamma-1)H^{-1}(1-\lambda) \leq -\frac{\gamma}{2}$$
$$\Leftrightarrow$$
$$-\gamma H^{-1}(1-\lambda) \leq -\frac{\gamma}{2}$$
$$\Leftrightarrow$$
$$H^{-1}(1-\lambda) \geq \frac{1}{2}$$
$$\Leftrightarrow$$
$$1-\lambda \geq H(\frac{1}{2}) = 1$$
$$\Leftrightarrow$$
$$\lambda \leq 0$$

Ce qui n'est jamais le cas, donc f est strictement croissante.

On calcule la valeur de f aux bornes de son intervalle de définition :

1.
$$\begin{aligned} f(0) &= (1-\gamma)\left(1-H\left(\frac{H^{-1}(1)-\frac{\gamma}{2}}{1-\gamma}\right)\right)-0 \\ &= (1-\gamma)\left(1-H(\frac{1}{2})\right) \\ &= 0 \end{aligned}$$

2.
$$\begin{aligned} f(1-H(\frac{\gamma}{2})) &= (1-\gamma)\left(1-H\left(\frac{H^{-1}(1-1+H(\frac{\gamma}{2}))-\frac{\gamma}{2}}{1-\gamma}\right)\right)-(1-H(\frac{\gamma}{2})) \\ &= (1-\gamma)\left(1-H\left(\frac{\frac{\gamma}{2}-\frac{\gamma}{2}}{1-\gamma}\right)\right)-1+H(\frac{\gamma}{2}) \\ &= (1-\gamma)-1+H(\frac{\gamma}{2}) \\ &= H(\frac{\gamma}{2})-\gamma \\ &> 0 \text{ pour } 0<\gamma<\frac{1}{2} \end{aligned}$$

On en déduit que $y_\lambda > \lambda$ pour $0 < \gamma < \frac{1}{2}$ et $0 < \lambda < 1 - H(\frac{\gamma}{2})$.

Il reste à montrer que $y \geq y_\lambda$. En effet :

$$h \leq H^{-1}(1-\lambda) \leq \frac{1}{2}$$
$$\frac{h-\frac{\gamma}{2}}{1-\gamma} \leq \frac{H^{-1}(1-\lambda)-\frac{\gamma}{2}}{1-\gamma} \leq \frac{\frac{1}{2}-\frac{\gamma}{2}}{1-\gamma}$$
$$\frac{h-\frac{\gamma}{2}}{1-\gamma} \leq \frac{H^{-1}(1-\lambda)-\frac{\gamma}{2}}{1-\gamma} \leq \frac{1}{2}$$
$$H\left(\frac{h-\frac{\gamma}{2}}{1-\gamma}\right) \leq H\left(\frac{H^{-1}(1-\lambda)-\frac{\gamma}{2}}{1-\gamma}\right)$$
$$1 - H\left(\frac{h-\frac{\gamma}{2}}{1-\gamma}\right) \geq 1 - H\left(\frac{H^{-1}(1-\lambda)-\frac{\gamma}{2}}{1-\gamma}\right)$$
$$(1-\gamma)\left(1 - H\left(\frac{h-\frac{\gamma}{2}}{1-\gamma}\right)\right) \geq (1-\gamma)\left(1 - H\left(\frac{H^{-1}(1-\lambda)-\frac{\gamma}{2}}{1-\gamma}\right)\right)$$
$$y \geq y_\lambda$$

□

LEMME 5. *Soient $\alpha_1, ...\alpha_t$ une suite de nombres telle que $\alpha_{j+1} = r\alpha_j$ et $\sum_{i=1}^{t} \alpha_i = s$. Alors $\alpha_1 = \frac{s(1-r)}{1-r^t}$ et $\alpha_t = \frac{s(1-\frac{1}{r})}{1-\left(\frac{1}{r}\right)^t}$.*

Démonstration. Premièrement :

$$\begin{aligned} s &= \sum_{i=1}^{t} \alpha_i \\ &= \sum_{i=1}^{t} r^i \alpha_1 \\ &= \frac{1-r^t}{1-r}\alpha_1 \end{aligned}$$

On en déduit $\alpha_1 = \frac{s(1-r)}{1-r^t}$.

Deuxièmement, si $\alpha_{j+1} = r\alpha_j$, alors $\frac{\alpha_{j+1}}{r} = \alpha_j$.

$$\begin{aligned} s &= \sum_{i=1}^{t} \alpha_i \\ &= \sum_{i=1}^{t} \left(\frac{1}{r}\right)^i \alpha_t \\ &= \frac{1-\left(\frac{1}{r}\right)^t}{1-\frac{1}{r}}\alpha_t \end{aligned}$$

Donc $\alpha_t = \frac{s(1-\frac{1}{r})}{1-\left(\frac{1}{r}\right)^t}$ □

LEMME 6. *1. Si* $\lambda + y\alpha_1 + \frac{\varepsilon}{2} = y + \varepsilon$, *alors* $\alpha_1 = \frac{y-\lambda+\frac{\varepsilon}{2}}{y}$.

2. Si $\lambda\alpha_t + y + \frac{\varepsilon}{2} = y + \varepsilon$, *alors* $\alpha_t = \frac{\varepsilon}{2\lambda}$.

THÉORÈME 30. *Pour tout* $\varepsilon > 0$ *et* $\lambda < 1 - H(\frac{\gamma}{2})$, *l'algorithme NearestNeighbour résout le problème Nearest Neighbour de paramètres* (m, γ, λ) *avec une probabilité inversement polynomiale en temps* $\tilde{O}\left(2^{(y+\varepsilon)m}\right)$ *où* $y = (1-\gamma)\left(1 - H\left(\frac{h-\frac{\gamma}{2}}{1-\gamma}\right)\right)$.

Démonstration. D'après le lemme 2 (page 80), l'arbre de calcul de l'algorithme comporte un bon chemin avec une probabilité inversement polynomiale. On considère ce chemin. Alors, d'après le lemme 3, la taille de chaque liste d'entrée au niveau j, $1 \leq j \leq t$ est $\tilde{O}\left(2^{m\left(\lambda\left(1-\sum_{i=1}^{j-1}\alpha_i\right)+\frac{\varepsilon}{2}\right)}\right)$. On construit $\tilde{O}\left(2^{y\alpha_j m}\right)$ nouvelles sous-listes à ce niveau, ce qui, par récurrence, donne un total de $\tilde{O}\left(2^{y\sum_{i=1}^{j}\alpha_i m}\right)$ sous-listes à ce niveau, construites à l'aide d'une procédure de recherche dans les listes d'entrées. La complexité de calcul au niveau j est donc :

$$\tilde{O}\left(2^{m\left(\lambda\left(1-\sum_{i=1}^{j-1}\alpha_i\right)+\frac{\varepsilon}{2}\right)}2^{y\sum_{i=1}^{j}\alpha_i m}\right)$$

$$\tilde{O}\left(2^{m\left(\lambda\left(1-\sum_{i=1}^{j-1}\alpha_i\right)+\frac{\varepsilon}{2}+y\sum_{i=1}^{j}\alpha_i\right)}\right)$$

Au dernier niveau ($j = t+1$), il y a $\tilde{O}\left(2^{ym}\right)$ listes d'entrée de taille $\tilde{O}\left(2^{m\left(\lambda\left(1-\sum_{i=1}^{t}\alpha_i\right)+\frac{\varepsilon}{2}\right)}\right) = \tilde{O}\left(2^{\frac{\varepsilon}{2}}\right)$ et on construit la liste des solutions de manière naïve (i.e. complexité quadratique en la taille des listes) à partir de ces listes, donc la complexité totale à ce niveau est $\tilde{O}\left(2^{m(y+\varepsilon)}\right)$. La complexité totale sera donc :

$$\max_{j=1,..t}\left(2^{m\left(\lambda\left(1-\sum_{i=1}^{j-1}\alpha_i\right)+y\sum_{i=1}^{j}\alpha_i+\frac{\varepsilon}{2}\right)}, 2^{m(y+\varepsilon)}\right)$$

Nous allons faire deux calculs séparément : d'abord, faire en sorte que les complexités de toutes les étapes sauf la dernière sont égales, et ensuite obtenir l'égalité entre la complexité de la dernière étape et celle des autres.

1. On voudrait que $\lambda\left(1-\sum_{i=1}^{j_0-1}\alpha_i\right)+y\sum_{i=1}^{j_0}\alpha_i+\frac{\varepsilon}{2} = \lambda\left(1-\sum_{i=1}^{j_1-1}\alpha_i\right)+y\sum_{i=1}^{j_1}\alpha_i+\frac{\varepsilon}{2}\ \forall\ 1 \leq j_0, j_1 \leq t$, en particulier pour $j_1 = j_0 + 1$:

$$\lambda\left(1-\sum_{i=1}^{j_0-1}\alpha_i\right)+y\sum_{i=1}^{j_0}\alpha_i+\frac{\varepsilon}{2} = \lambda\left(1-\sum_{i=1}^{j_0}\alpha_i\right)+y\sum_{i=1}^{j_0+1}\alpha_i+\frac{\varepsilon}{2}$$
$$\Leftrightarrow$$
$$-\lambda\alpha_{j_0} + y\alpha_{j_0+1} = 0$$
$$\Leftrightarrow$$
$$\alpha_{j_0+1} = \frac{\lambda}{y}\alpha_{j_0}$$

Cela signifie que les α_j constituent une suite décroissante de raison $\frac{\lambda}{y}$ (puisque $y > \lambda$). On se sert de 5 (page 87) afin de calculer l'expression

de α_1 : $\alpha_1 = \frac{1-\frac{\lambda}{y}}{1-\left(\frac{\lambda}{y}\right)^t}$.

Avec ce choix de α_1, toutes les complexités dépendantes de j seront égales et l'expression la plus simple de leur valeur sera :

$$\tilde{O}\left(2^{m\left(\lambda+y\alpha_1+\frac{\varepsilon}{2}\right)}\right)$$

2. Pour avoir égalité entre toutes les complexités, on cherche à avoir $\lambda + y\alpha_1 + \frac{\varepsilon}{2} = y + \varepsilon$. Or, d'après 6 (page 88), on aura alors $\alpha_1 = \frac{y-\lambda+\frac{\varepsilon}{2}}{y}$.

   Nous pouvons maintenant, en nous servant de l'égalité $\alpha_1 = \frac{1-\frac{\lambda}{y}}{1-\left(\frac{\lambda}{y}\right)^t}$, calculer la valeur de t en fonction des autres paramètres.

$$\alpha_1 = \frac{y-\lambda+\frac{\varepsilon}{2}}{y}$$

$$\frac{1-\frac{\lambda}{y}}{1-\left(\frac{\lambda}{y}\right)^t} = \frac{y-\lambda+\frac{\varepsilon}{2}}{y}$$

$$y\frac{1-\frac{\lambda}{y}}{y-\lambda+\frac{\varepsilon}{2}} = 1-\left(\frac{\lambda}{y}\right)^t$$

$$1-\frac{y-\lambda}{y-\lambda+\frac{\varepsilon}{2}} = \left(\frac{\lambda}{y}\right)^t$$

$$\frac{y-\lambda+\frac{\varepsilon}{2}-y+\lambda}{y-\lambda+\frac{\varepsilon}{2}} = \left(\frac{\lambda}{y}\right)^t$$

$$\log_2\left(\frac{\frac{\varepsilon}{2}}{y-\lambda+\frac{\varepsilon}{2}}\right) = t\log_2\left(\frac{\lambda}{y}\right)$$

$$\log_2\left(\frac{\varepsilon}{2}\right) - \log_2\left(y-\lambda+\frac{\varepsilon}{2}\right) = t\left(\log_2(\lambda) - \log_2(y)\right)$$

$$t = \frac{\log_2\left(y-\lambda+\frac{\varepsilon}{2}\right) - \log_2\left(\frac{\varepsilon}{2}\right)}{\log_2(y) - \log_2(\lambda)}$$

□

6.1.2 Application au décodage : $BJMM + MO$

Dans leur article, May et Ozerov intègrent cet algorithme de résolution du problème Nearest Neighbour à l'algorithme de Stern (que l'on n'a pas vu) et à l'algorithme $BJMM$. À chaque fois, il s'agit de calculer les listes L_1 et L_2 comme dans l'algorithme habituel, c'est-à-dire directement pour Stern, et en utilisant une recherche de collisions puis la technique des représentations pour $BJMM$, puis donner ces deux listes en entrée à $NearestNeighbour$.

En fait, cela n'est pas tout à fait vrai pour $BJMM$. Rappelons que dans $BJMM$ on peut écrire H sous la forme :
$\begin{bmatrix} H'|0_{h,n-k-h} \\ B|I_{n-k-h} \end{bmatrix}$ avec $H' \in \mathcal{M}_{h,k+h}$, $B \in \mathcal{M}_{n-k-h,k+h}$.

D'après la spécification du problème Nearest Neighbour, l'algorithme doit donner en sortie une liste C contenant $(u^*, v^*) \in L_1 \times L_2$ tel que $w_H(u^* + v^*) = \gamma(n-k-h)$, où en pratique $\gamma = \frac{t-p}{n-k-h}$. On est ensuite censé trouver $e_1 \in L_1$ et $e_2 \in L_2$ tels que $u^* = He_1^T$ et $v^* = He_2^T + s$ sur les $n-k-h$ dernières coordonnées seulement, c'est-à-dire, plus précisément, $u^* = [B|I_{n-k-h}]e_1^T$ et $v^* = [B|I_{n-k-h}]e_2^T + s^*$ où s^* signifie s restreint à ses $n-k-h$ dernières coordonnées. Le vecteur d'erreur cherché est alors $e_1 + e_2 + (0^{k+h}|u^* + v^*)$. En effet :

$$\begin{aligned} H(e_1 + e_2 + (0^{k+h}|u^* + v^*))^T &= He_1^T + He_2^T + H(0^{k+h}|u^* + v^*)^T \\ &= H(e_1 + e_2)^T + (0^h|u^* + v^*)^T \\ &= H(e_1 + e_2)^T + (0^h|[B|I_{n-k-h}]e_1 + [B|I_{n-k-h}]e_2 + s^*)^T \\ &= H(e_1 + e_2)^T + H(e_1 + e_2)^T + s \qquad (*) \\ &= s \end{aligned}$$

(*) n'est vrai que si la façon de construire L_1 et de L_2 garantit l'égalité $H'(e_1 + e_2)^T = s'$ sur toute la longueur du syndrome. Or, dans l'algorithme $BJMM$ habituel, par définition, les éléments de L_1 sont égaux au syndrome sur $h_1 < h$ coordonnées seulement. Par conséquent, quand on applique l'algorithme de May et Ozerov pour le problème Nearest Neighbour à $BJMM$, il faut prendre $h_1 = h$ (et donc on prend pour $h_{1,1}$ la valeur qu'on aurait pris pour h_1 dans la version habituelle de $BJMM$, à savoir $h_{1,1} = f_1(p) + (k+h-f_1(p))H(\frac{\varepsilon_2}{k+h(f_1(p)}$) . Cela entraîne :

$H(e_1+e_2)^T+s = (H'(e_1+e_2) \mid [B|I_{n-k-h}](e_1+e_2))+(s'|s^*) = (0^h|[B|I_{n-k-h}]e_1+ [B|I_{n-k-h}]e_2 + s^*)^T$.

THÉORÈME 31 ([13]). *Si H est une matrice aléatoire, en notant $R := \frac{k}{n}$, $\tau := \frac{t}{n}$, $\pi := \frac{p}{n}$,$\eta = \frac{h}{n}$ et $\epsilon_i = \frac{\varepsilon_i}{n}, i = 1, 2$ et $\gamma = \frac{\tau-\pi}{1-R-\eta}$, on a l'expression suivante pour l'exposant $\alpha_{BJMM+MO}$ de cet algorithme :*

$$\alpha_{BJMM+MO}(R) = \min_{\pi,\eta,\epsilon_1,\epsilon_2} (\beta(R,\pi,\eta) + \max_{i=1,2,3,4,5} \gamma_i(R,\eta,\pi,\epsilon_1,\epsilon_2))$$

Avec :

$$\boxed{\beta(R,\pi,\eta) = H(\tau) - (1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta}) - (R+\eta)H(\frac{\pi}{R+\eta})}$$

$$\boxed{\gamma_1(R,\eta,\pi) = \frac{R+\eta}{2}H(\frac{f_2(\pi)}{R+\eta})}$$

$$\boxed{\gamma_2(R,\eta,\pi) = (R+\eta)H(\frac{f_2(\pi)}{R+\eta}) - \eta_1}$$

$$\boxed{\gamma_3(R,\eta,\pi) = 2(R+\eta)H(\frac{f_2(\pi)}{R+\eta}) - \eta - \eta_1}$$

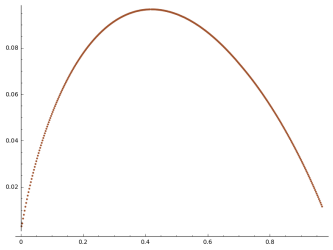
$$\boxed{\gamma_4(R,\eta,\pi) = (R+\eta)H(\frac{f_1(\pi)}{R+\eta}) - \eta}$$

$$\boxed{\gamma_5(R,\eta,\pi) = (1-R-\eta)(1-\gamma)\left(1 - H\left(\frac{H^{-1}(1-\frac{\gamma_4(R,\eta,\pi)}{1-R-\eta}) - \frac{\gamma}{2}}{1-\gamma}\right)\right)}$$

Soumise aux contraintes :

$$\eta = \pi + (R+\eta-\pi)H(\tfrac{\epsilon_1}{R+\eta-\pi}) \leq (R+\eta)H(\tfrac{f_1(\pi)}{R+\eta})$$
$$\Leftrightarrow \epsilon_1 = (R+\eta-\pi)H(\tfrac{\eta-\pi}{R+\eta-\pi})$$
$$\eta_1 = f_1(\pi) + (R+\eta-f_1(\pi))H(\tfrac{\epsilon_2}{R+\eta-f_1(\pi)}) \leq (R+\eta)H(\tfrac{f_2(\pi)}{R+\eta})$$
$$0 \leq \pi \leq \min(\tau, R+\eta)$$
$$0 \leq \eta \leq 1-R-\tau+\pi \leq 1-R$$
$$0 \leq \epsilon_1 \leq R+\eta-\pi$$
$$0 \leq \epsilon_2 \leq R+\eta-f_1(\pi)$$
$$\epsilon_2 \leq \epsilon_1$$
$$0 \leq \eta_1 \leq \eta$$

Et $\max_R(\alpha_{BJMM+MO}) = 0.0966448$ *pour* $R = 0.4174$, *valeur obtenue pour*
$\pi \approx 0.065453458$
$\eta \approx 0.19080535358$
$\epsilon_1 \approx 0.02038174$
$\epsilon_2 \approx 0.00401212098$.



Graphe pour $BJMM + MO$

Démonstration. Les γ_i pour $i = 1, 2, 3, 4$ sont les mêmes que ceux de l'algorithme de $BJMM$ habituel, à la substitution de η_1 et $\eta_{1,1}$ par η resp. η_1 près.

Le terme γ_5 vient de l'algorithme de résolution du problème Nearest Neighbour : en effet, les listes L_1 et L_2 sont de taille $2^{\gamma_4(R,\eta,\pi)} = 2^{\lambda(1-R-\eta)} \Leftrightarrow \lambda = \frac{\gamma_4(R,\eta,\pi)}{1-R-\eta}$. □

6.2 Version quantique

6.2.1 Avec l'algorithme de Grover

Il s'agit d'utiliser l'algorithme de Grover pour effectuer les opérations de recherche. Pour la recherche de la permutation et du vecteur aléatoire, cela se fait de manière assez intuitive. Nous allons donc nous concentrer sur la recherche de bonnes partitions A_j, écrivant ainsi une version quantique de la fonction $NearestNeighbourRec$, $NearestNeighbourGrover$.

On considère le niveau j avec les listes $L_1^{(j-1)}$ et $L_2^{(j-1)}$ contenant les vecteurs de poids $\frac{\alpha_k hm}{2}$ sur la partie $A_{k,1}$ de chacune des bandes de longueur $\alpha_k m$, pour $1 \leq k \leq j-1$. À quelle condition une partition A_j est-elle bonne à ce niveau ? À

condition qu'il existe un chemin du sous-arbre du calcul jusqu'aux noeuds, c'est-à-dire, des partitions A_{j+1}, ... , A_t telles qu'une solution (u^*, v^*) du problème se trouve dans toutes les listes du chemin de calcul.

Par conséquent, il y a la procédure récursive suivante pour vérifier qu'une partition A_j est bonne :

- $1 \leq j \leq t-1$: il faut vérifier qu'il existe une bonne partition A_{j+1}.
- $j = t$: il faut vérifier qu'il existe une bonne partition A_t, puis qu'il y a bien un couple $(u, v) \in L_1^{(t)} \times L_1^{(t)}$ tel que $w_H(u+v) = \gamma m$.

Pour faire cela en quantique, nous allons définir t fonctions f_j qui serviront d'oracle de vérification à t procédures de recherche de Grover G_j. Les fonctions f_j se servent de procédures K_j afin de construire des sous-listes. Nous en donnons d'abord les spécifications :

— $f_j : A_j = A_{j,1} \sqcup A_{j,2}, L_1^{(j-1)}, L_2^{(j-1)}, m, \gamma, h, \lambda, \varepsilon \to \{0,1\}$
Soit $(u^*, v^*) \in L_1^{(j-1)} \times L_2^{(j-1)}$ une solution du problème Nearest Neighbour.
$f_j(A_j, L_1^{(j-1)}, L_2^{(j-1)}, ...) = \begin{cases} 1 & \text{si } u^* \text{ et } v^* \text{ sont de poids } \frac{\alpha_k hm}{2} \text{ sur } A_{k,1} \forall j \leq k \leq t, \\ 0 & \text{sinon.} \end{cases}$

— $G_j = Grover(2^{\frac{y\alpha_j m}{2}}, f_j : *, L_1^{(j-1)}, L_2^{(j-1)}, m, \gamma, h, \lambda, \varepsilon\})$

Autrement dit :

$G_j : \{f_j : *, L_1^{(j-1)}, L_2^{(j-1)}, m, \gamma, h, \lambda, \varepsilon\} \to \{$partitions de la bande de longueur $\alpha_j m$ en deux parties égales$\}$
G_j renvoie un A_j tel que $f_j(A_j, L_1^{(j-1)}, L_2^{(j-1)}, m, \gamma, h, \lambda, \varepsilon) = 1$.

— $K_j : L_i^{(j-1)}, h, A_j \mapsto L_i^{(j)}$ où $L_i^{(j)} \subset \{v_{L_i^{(j-1)}} \in L_i^{(j-1)}$ de poids $\frac{\alpha_j hm}{2}$ sur $A_{j,1}\}$.

À chaque fois, f_j est donc l'oracle d'inversion de la phase de G_j. Nous allons maintenant donner le corps de ces algorithmes. Il s'agit d'une imbrication sur $t-1$ niveaux puisque pour tout $1 \leq j \leq t-1$, il y a un appel à la procédure G_{j+1} dans le corps de la fonction f_j, c'est-à-dire qu'elle intervient dans l'étape d'inversion de la phase de G_j.

Algorithme 19 : $f_j, 1 \leq j \leq t-1$

Entrées : A_j, $L_1^{(j-1)}$, $L_2^{(j-1)}$, m, γ, h, λ, ε
Sorties : 0 ou 1

pour $i = 1, 2$ **faire**
  $L_i^{(j)} \leftarrow K_j(L_i^{(j-1)}, h, A_j)$
si $|L_i^{(j)}| \lessapprox \tilde{O}\left(2^{\lambda m(1-\sum_{i=1}^{j}\alpha_i + \frac{\varepsilon}{2})}\right)$ **alors**
  $A_{j+1} \leftarrow G_{j+1}\left(f_{j+1}(*, L_1^{(j)}, L_2^{(j)}, m, \gamma, h, \lambda, \varepsilon)\right)$
  retourner $f_{j+1}(A_{j+1}, L_1^{(j)}, L_2^{(j)}, m, \gamma, h, \lambda, \varepsilon)$
sinon
  retourner 0

Algorithme 20 : f_t

Entrées : A_t, $L_1^{(t-1)}$, $L_2^{(t-1)}$, m, γ, h, λ, ε
Sorties : 0 ou 1

pour $i = 1, 2$ **faire**
  $L_i^{(t)} \leftarrow K_t(L_i^{(t-1)}, h, A_t)$
si $|L_i^{(t)}| \lessapprox \tilde{O}\left(2^{m\frac{\varepsilon}{2}}\right)$ **alors**
  $C \leftarrow \{(u, v) \in L_1^{(t)} \times L_2^{(t)} \mid w_H(u + v) = \gamma m\}$
  si $C \neq \emptyset$ **alors**
    retourner 1
  sinon
    retourner 0
sinon
  retourner 1

À l'aide de ces fonctions, nous définissions la fonction $NearestNeighbourGrover$.

Algorithme 21 : $NearestNeighbourGrover$

Entrées : $\overline{L_1} = L_1^{(0)}, \overline{L_2} = L_2^{(0)}, m, t, \gamma, \lambda, (\alpha_j)_{j=1,..,t}, y, h, \varepsilon, 1$
Sorties : 0 ou 1

pour $j = 1$ *à* t **faire**
  $A_j \leftarrow G_j\left(f_j(*, L_1^{(j-1)}, L_2^{(j-1)}, m, \gamma, h, \lambda, \varepsilon)\right)$
  pour $i = 1, 2$ **faire**
    $L_i^{(j)} \leftarrow K_j(L_i^{(j-1)}, h, A_j)$

Dans un premier temps, nous supposerons que K_j construit toute la liste voulue. Ainsi, si l'on dénote la complexité d'une fonction f par $\mathcal{C}(f)$, on a $\mathcal{C}(K_j) = 2^{\lambda m(1-\sum_{i=1}^{j-1}\alpha_i)+\frac{\varepsilon}{2}}$.

On s'en sert pour calculer les complexités des autres procédures.

- $\mathcal{C}(G_j) = 2^{\frac{y\alpha_j m}{2}}\mathcal{C}(f_j)$.
- $\mathcal{C}(f_j) = \max\left(\mathcal{C}(K_j), \mathcal{C}(G_{j+1}), \mathcal{C}(f_{j+1})\right) = \max\left(\mathcal{C}(K_j), \mathcal{C}(G_{j+1})\right)$.
- $\mathcal{C}(f_t) = \max\left(\mathcal{C}(K_t), 2^{\varepsilon m}\right)$.

Par conséquent :

$$\begin{aligned}
\mathcal{C}(f_j) &= \max\left(\mathcal{C}(K_j), 2^{\frac{y\alpha_{j+1}m}{2}}\mathcal{C}(f_{j+1})\right)\\
&= \max\left(\mathcal{C}(K_j), 2^{\frac{y\alpha_{j+1}m}{2}}\max\left(\mathcal{C}(K_{j+1}), \mathcal{C}(G_{j+1})\right)\right)\\
&= \max\left(\mathcal{C}(K_j), 2^{\frac{y\alpha_{j+1}m}{2}}\max\left(\mathcal{C}(K_{j+1}), 2^{\frac{y\alpha_{j+2}m}{2}}\mathcal{C}(f_{j+2})\right)\right)\\
&= \max_{0\leq k\leq t-j-1}\left(2^{\sum_{i=j+1}^{j+k}\frac{y\alpha_i m}{2}}\mathcal{C}(K_{j+k}), 2^{\sum_{i=j+1}^{t}\frac{y\alpha_i m}{2}}\mathcal{C}(f_t)\right)\\
&= \max_{0\leq k\leq t-j-1}\left(2^{\sum_{i=j+1}^{j+k}\frac{y\alpha_i m}{2}}\mathcal{C}(K_{j+k}), 2^{\sum_{i=j+1}^{t}\frac{y\alpha_i m}{2}}\max\left(\mathcal{C}(K_t), 2^{\varepsilon m}\right)\right)\\
&= \max_{0\leq k\leq t-j}\left(2^{\sum_{i=j+1}^{j+k}\frac{y\alpha_i m}{2}}\mathcal{C}(K_{j+k}), 2^{\sum_{i=j+1}^{t}\frac{y\alpha_i m}{2}+\varepsilon m}\right)
\end{aligned}$$

et

$$\begin{aligned}
\mathcal{C}(G_j) &= 2^{\frac{y\alpha_j m}{2}}\mathcal{C}(f_j)\\
&= \max_{0\leq k\leq t-j}\left(2^{\sum_{i=j}^{j+k}\frac{y\alpha_i m}{2}}\mathcal{C}(K_{j+k}), 2^{\sum_{i=j}^{t}\frac{y\alpha_i m}{2}+\varepsilon m}\right)
\end{aligned}$$

Par conséquent, dans le cas présent, nous avons :

$$\mathcal{C}(G_j) = \max_{0\leq k\leq t-j}\left(2^{\sum_{i=j}^{j+k}\frac{y\alpha_i m}{2}}2^{\lambda m(1-\sum_{i=1}^{j+k-1}\alpha_i)+\frac{\varepsilon}{2}}, 2^{\sum_{i=j}^{t}\frac{y\alpha_i m}{2}+\varepsilon m}\right)$$

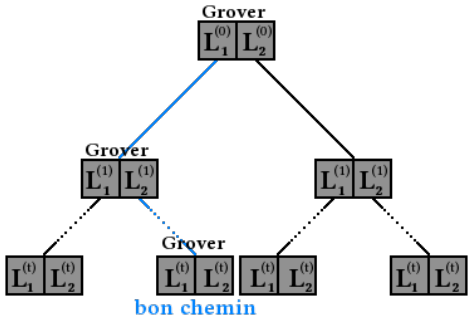
En particulier,

$$\begin{aligned}
\max_{1\leq j\leq t}\mathcal{C}(G_j) &= \mathcal{C}(G_1)\\
&= \max_{0\leq k\leq t-1}\left(2^{\sum_{i=1}^{k+1}\frac{y\alpha_i m}{2}}2^{\lambda m(1-\sum_{i=1}^{k}\alpha_i)+\frac{\varepsilon}{2}}, 2^{(\frac{y}{2}+\varepsilon)m}\right)\\
&\geq 2^{\frac{y\alpha_1 m}{2}+\lambda m+\frac{\varepsilon}{2}}\\
&\geq 2^{\lambda m}
\end{aligned}$$

Or, cette borne inférieure est trop grande car c'est la complexité de l'algorithme quantique naïf de résolution du problème. Par conséquent, l'algorithme esquissé ci-dessus s'avère inefficace.

6.2.2 Avec Quantum Walk ?

Le problème de l'algorithme esquissé dans la partie précédente vient de la taille des listes construites. Une solution naturelle pour réduire la taille de ces listes serait d'utiliser une marche aléatoire sur un graphe de Johnson.
On a dit que l'on pouvait considérer l'algorithme de May et Ozerov sous forme d'un arbre de calcul à $t+1$ niveaux numérotés de 0 à t, où à chaque noeud il y a deux listes $L_j^{(1)}$ et $L_j^{(2)}$ et chaque noeud de niveau j a en moyenne $2^{y\alpha_j m}$ fils. Le but était de trouver un bon chemin, vérifiant certaines propriétés, de la racine à une feuille.
Ce qu'on a fait avec l'algorithme de Grover consistait à améliorer le temps de recherche du bons fils à chaque niveau de l'arbre. Illustration :



Le problème de décodage par collision que l'on avait considérait avant cette section, et qui s'est apprêté avec succès à l'utilisation d'une marche aléatoire quantique s'illustrait également sous forme d'arbre.

Dans les deux problèmes, ce sont les listes au niveau des feuilles qui sont (a priori) les plus petites et celles au niveau du noeud qui sont (a priori) les plus grandes. Les problèmes sont néanmoins différents dans la mesure où dans le problème du décodage par collision, on commençait par les listes situées aux feuilles, alors que dans le cas présent, on commence par les listes à la racine.

7 Développements limités pour les formules de complexité

Nous donnons ici des résultats dont les détails de calcul se trouvent dans l'appendice B.2. L'intégralité des résultats donnés dans cette section est valable dans le cas $t = \frac{d_{GV}}{2}$. Dans le cas $t = d_{GV}$, les résultats ne valent plus pour les algorithmes suivants : DG, SSG et MMT. Sans doute faudrait-il aller plus loin dans les développements limités pour obtenir des résultats qui seraient également valables dans ce cas.

Pour tous les algorithmes sans technique de représentations (D/SS, DG, SSG et $SSQW$), leur complexité temporelle s'écrit sous la forme $F(\pi,\eta) = H(\tau) - (1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta}) - c(R+\eta)H(\frac{\pi}{R+\eta})$ avec $0 \leq c \leq 1$ (nous ferons abstraction ici du facteur $\frac{1}{2}$ des algorithmes quantiques).

De plus, il y a une relation de dépendance entre les paramètres de la forme $2^h = \binom{k+h}{p}^s$, c'est-à-dire $\eta = s(R+\eta)H(\frac{\pi}{R+\eta})$ avec $0 \leq s \leq 1$. Cette relation de dépendance est vérifiée ssi la fonction suivante est nulle : $G(\pi,\eta) := \eta - s(R+\eta)H(\frac{\pi}{R+\eta})$.

Alors, si l'on note $\textsc{Prange} := \alpha_P(R)$ (l'exposant de Prange), et que l'on définit :

- $a := a(c,s) := s\frac{\frac{c}{s} + \frac{\alpha\tau}{(1-R)} - H(\frac{\tau}{1-R})}{\ln(2)}$
- $\alpha := \log_2\left(\frac{1-R-\tau}{\tau}\right)$

En utilisant les développements limités, nous trouvons :

$$\boxed{F(\pi,\eta) = F(\pi) = \textsc{Prange} + a\pi \ln\left(\frac{\pi}{R}\right) + (\alpha - a)\,\pi + o(\pi)}$$

Cela nous permet de calculer la valeur approximative du point $\pi_{\min}$ où la complexité est minimale, ainsi que la valeur de la fonction donnant la complexité en ce point :

- $\pi_{\min} = Re^{-\frac{\alpha}{a}}$
- $\min_\pi F(\pi) = \textsc{Prange} - aRe^{-\frac{\alpha}{a}} + o(\pi)$

De manière plus générale, en testant plusieurs valeurs numériquement, on fait le constat suivant : à c constant, plus s est petit, mieux c'est. A s constant, plus c est grand, mieux c'est.

Les résultats obtenus de cette manière sont très proches de ceux obtenus avec l'optimisation numérique (cf. appendice C pour les codes et courbes).

En ce qui concerne les algorithmes utilisant la technique des représentations, a priori :

- La complexité ne s'écrit plus sous la forme F.
- Dans MMT, il n'y a pas de dépendance entre π et η.
- Dans l'algorithme $MMTQW$, la dépendance entre π et η n'a pas la forme G.

En faisant les développements limités, nous faisons toutefois les constats suivants pour MMT :

- La taille des listes de base $L_{i,j}$ est inférieure à celle des autres listes L_i et L.
- Si l'on part de l'hypothèse qu'il y a une dépendance entre η et π, nous trouvons que c'est effectivement le cas pour le point où la complexité est minimale ! Ce point est en effet atteint pour $0,54 < s < 0,59$ (valeur exacte dépendant de R). À ce point, les listes L_i et L sont de même taille.
- La complexité s'écrit bien à l'aide de la fonction F, avec $c = s$.

De même, les développements limités nous permettent d'affirmer à propos de $MMTQW$ que :

- La dépendance entre η et π s'écrit *approximativement* à l'aide de G, i.e. $s = 2/5 + \epsilon$ avec ϵ un terme d'erreur.
- Ici aussi complexité s'écrit bien à l'aide de F, avec $c = s$.

Le tableau suivant résume les valeurs de c et s pour chaque algorithme :

Algorithme	s	c
D/SS	$1/2$	$= s$
DG	1	$1/3$
SSG	$1/2$	$1/4$
$SSQW$	$2/5$	$= s$
MMT	$0{,}54 < . < 0{,}59$	$= s$
$MMTQW$	$2/5 + \epsilon$	$= s$

8 Conclusion

En guise de conclusion, on va donner une liste de questions levées dans ce mémoire auxquelles on n'a pas su répondre, et énumérer quelques pistes pour affiner les analyses qui ont été effectuées.

1. Dans l'algorithme $BJMM$ classique, on a distingué entre le temps de construction des listes et la taille des listes. Dans l'algorithme $BJMMQW$, on a fait les calculs, faute de mieux, avec le temps de construction des listes comme une borne supérieure sur leur taille. Cela signifie que la complexité réelle est très probablement inférieure à la valeur que l'on a trouvé. Il faut donc trouver une façon de faire rentrer la vraie taille des listes dans le calcul de complexité pour avoir une valeur plus exacte de la complexité réelle de l'algorithme.
2. On a analysé quelques conditions d'application de la marche aléatoire quantique à l'algorithme de May et Ozerov, mais on n'a pas réussi à l'appliquer en pratique.
3. Là où le problème du décodage était pertinent pour les cryptosystèmes basés sur les codes correcteurs, un autre problème est pertinent pour les schemas de signatures. Il s'agit du problème de décodage multiple (Decoding One out of Many). Il faudrait trouver des algorithmes quantiques pour ce problème aussi.
4. Il y aurait éventuellement un gain en poinçonnant le code avant d'appliquer un algorithme ISD : ce n'est pas le cas pour l'algorithme de Prange, mais cela l'est peut-être pour les autres algorithmes. Il faut également étudier cela car en ce faisant, on ajoute un problème de recherche qui se prête très bien à une application de l'algorithme de Grover.

Je tiens également à remercier les personnes suivantes :

— Jean-Pierre Tillich pour sa patience, ses remarques subtiles et précises, de m'avoir suggéré des idées fructueuses et son extrême gentillesse.
— Nicolas Sendrier pour sa gentillesse, ses explications détaillées et ses idées fructueuses.
— Rodolfo Canto Torres pour m'avoir envoyé son logiciel d'optimisation des paramètres.
— Thomas Debris pour m'avoir aidé à préparer mes transparents et ses conseils pour l'optimisation numérique.
— Équipe-projet SECRET pour avoir créé un environnement de travail fort sympathique.

Références

[1] Becker, A. *The representation technique, Applications to hard problems in cryptography.* PhD thesis, Université Versailles Saint-Quentin en Yvelines, Oct. 2012.

[2] Becker, A., Joux, A., May, A., and Meurer, A. Decoding Random Binary Linear Codes in $2^{n/20}$: How $1 + 1 = 0$ Improves Information Set Decoding. In *Advances in Cryptology - EUROCRYPT 2012* (2012), Lecture Notes in Comput. Sci., Springer.

[3] Bernstein, D. J. Grover vs. McEliece. In *Post-Quantum Cryptography 2010* (2010), N. Sendrier, Ed., vol. 6061 of *Lecture Notes in Comput. Sci.*, Springer, pp. 73–80.

[4] Bernstein, D. J., Jeffery, S., Lange, T., and Meurer, A. Quantum Algorithms for the Subset-Sum Problem. In *Post-Quantum Cryptography 2011* (Limoges, France, June 2013), vol. 7932 of *Lecture Notes in Comput. Sci.*, pp. 16–33.

[5] Cerf, N. J., Grover, L. K., and Williams, C. P. Nested Quantum Search and NP-Hard Problems. *Applicable Algebra in Engineering, Communication and Computing 10*, 4 (2000), 311–338.

[6] de Wolf, R. Quantum Computing : Lecture Notes (consulté le 20/03/2015), 2016. homepages.cwi.nl/~rdewolf/qcnotes.pdf.

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

[8] Jeffery, S. *Frameworks for Quantum Algorithms.* PhD thesis, University of Waterloo, 2014.

[9] Jeffery, S., Kothari, R., and Magniez, F. Nested quantum walks with quantum data structures. In *the 24th ACM-SIAM Symposium on Discrete Algorithms (SODA 2013)* (2013), pp. 1474–1485. arXiv :1210.1199.

[10] Lee, P. J., and Brickell, E. F. An Observation on the Security of McEliece's Public-Key Cryptosystem. In *Advances in Cryptology - EUROCRYPT'88* (1988), vol. 330 of *Lecture Notes in Comput. Sci.*, Springer, pp. 275–280.

[11] Loeliger, H.-A. On the Basic Averaging Arguments for Linear Codes. In *Communications and Cryptography : Two Sides of One Tapestry*, R. E. Blahut, D. J. Costello, U. Maurer, and T. Mittelholzer, Eds. Springer US, 1994, pp. 251–261.

[12] May, A., Meurer, A., and Thomae, E. Decoding random linear codes in $O(2^{0.054n})$. In *Advances in Cryptology - ASIACRYPT 2011* (2011), D. H. Lee and X. Wang, Eds., vol. 7073 of *Lecture Notes in Comput. Sci.*, Springer, pp. 107–124.

[13] May, A., and Ozerov, I. On Computing Nearest Neighbors with Applications to Decoding of Binary Linear Codes. In *Advances in Cryptology - EUROCRYPT 2015* (2015), E. Oswald and M. Fischlin, Eds., vol. 9056 of *Lecture Notes in Comput. Sci.*, Springer, pp. 203–228.

[14] NIST. Post-Quantum Cryptography Project (consulté le 24/07/2016), 2016. csrc.nist.gov/groups/ST/post-quantum-crypto/index.html.

[15] Prange, E. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory 8*, 5 (1962), 5–9.

[16] Preskill, J. Lecture Notes for Ph219/CS219 : Quantum Information (consulté le 03/04/2015), 2015. www.theory.caltech.edu/~preskill/ph219/chap2_15.pdf.

[17] Santha, M. Quantum walk based search algorithms. In *5th TAMC* (2008), pp. 31–46. arXiv/0808.0059.

[18] Sendrier, N. Decoding One Out of Many. In *Post-Quantum Cryptography 2011* (2011), vol. 7071 of *Lecture Notes in Comput. Sci.*, pp. 51–67.

[19] Sendrier, N. Code-Based Cryptography MOOC - Message Attacks (ISD), 2014. www.fun-mooc.fr/courses/inria/41006S02/session02/about.

A Le formalisme quantique

Nous allons présenter ici le formalisme quantique d'une manière pertinente pour l'algorithmique quantique, c'est-à-dire, par exemple, en nous restreignant au cas d'espaces de dimension finie, et en considérant les aspects informationnels des objets introduits à l'aide d'une terminologie et d'une notation adéquates.

A.1 Environnement mathématique

Espaces de Hilbert On considère l'espace de Hilbert complexe $\mathcal{H}$ de dimension n. Une base canonique de cet espace est donnée par $(e_i)_{i=1,...,n}$ où e_i est le vecteur dont la $i^{\text{ème}}$ coordonnées est 1 et toutes les autres sont nulles. Alors :

1. Pour tout $v \in \mathcal{H}$, nous pouvons écrire $v = \sum_{i=1^n} v_i e_i$ avec $v_i \in \mathbb{C}$.
2. Toute fonction (opérateur) linéaire $A : \mathcal{H} \to \mathcal{H}$ peut se donner comme une matrice de $\mathcal{M}_{n,n}(\mathbb{C})$.
3. L'espace $\mathcal{H}$ est doté d'un produit scalaire $\langle .|. \rangle : (u, v) \mapsto \sum_{i=1^n} \overline{u_i} v_i$ sesquilinéaire.
4. À tout opérateur A est associé son opérateur adjoint A^* définie par l'équation $\langle Au|v\rangle = \langle u|A^*v\rangle$. La matrice adjointe A^* d'une matrice A est la matrice transposée de la matrice conjuguée, $\overline{A}^T$.
5. Un opérateur A est **auto-adjoint** si $A = A^*$. Ainsi une matrice carrée auto-adjointe sera réelle et symétrique.
6. Un opérateur A est **unitaire** s'il est inversible et $A^{-1} = A^*$. Ainsi, une matrice unitaire sera inversible et préservera la norme.

Notation de Dirac Pour un vecteur colonne v, au lieu d'écrire simplement v, nous utiliserons l'écriture $|v\rangle$, appelé **ket**. Nous définissons aussi l'écriture suivante pour le vecteur dual, $\langle v| := \overline{v}^T$, appelé **bra**. Cela est inspiré de la notation pour le produit scalaire : en effet, nous avons alors de façon naturelle l'égalité $\langle u|v\rangle = \overline{u}^T v$. De même, $|v\rangle\langle u|$ est une matrice carrée $n \times n$, donc un opérateur.

Produit tensoriel Le produit tensoriel de deux espaces $\mathcal{H}_A$ et $\mathcal{H}_B$, noté $\mathcal{H}_A \otimes \mathcal{H}_B$, est un espace produit qui vérifie aussi peu de contraintes que possibles. Plus précisément, on exige seulement les identifications suivantes, pour $u_1, u_2 \in \mathcal{H}_A$ et $v_1, v_2 \in \mathcal{H}_B$, $z \in \mathbb{C}$:

1. $(u_1, v_1) + (u_2, v_1) \sim (u_1 + u_2, v_1)$
2. $(u_1, v_1) + (u_1, v_2) \sim (u_1, v_1 + v_2)$
3. $z(u_1, v_1) \sim (zu_1, v_1)$
4. $z(u_1, v_1) \sim (u_1, zv_1)$

Le produit tensoriel se définit aussi pour les vecteurs et les applications linéaires :

1. $u = (u_1, ..., u_n) \in \mathcal{H}_A$ et $v = (v_1, ..., v_m) \in \mathcal{H}_B$
   $u \otimes v := (u_1 v, ..., u_n v)$ de longueur mn

2. $A \in \mathcal{L}(\mathcal{H}_A) \simeq \mathcal{M}_{n,n}(\mathbb{C})$ et $B \in \mathcal{L}(\mathcal{H}_B) \simeq \mathcal{M}_{m,m}(\mathbb{C})$

$$A \otimes B = \begin{bmatrix} a_{1,1}B & ... & a_{1,n}B \\ ... & ... & ... \\ a_{n,1}B & ... & a_{n,n}B \end{bmatrix}$$

$A \otimes B \in \mathcal{L}(\mathcal{H}_A \otimes \mathcal{H}_B)$

On a aussi :

1. Si $(e_i)_{i=1,..,n}$ est une base de $\mathcal{H}_A$ et $(f_i)_{i=1,..,m}$ est une base de $\mathcal{H}_B$, $(e_i f_j)_{i=1,..,n;j=1,...,m}$ est une base de $\mathcal{H}_A \otimes \mathcal{H}_B$.
2. $(A \otimes B)|u\rangle \otimes |v\rangle = A|u\rangle \otimes B|v\rangle$

A.2 Mécanique et information quantique : états, observables, mesure, évolution, systèmes composés

États quantiques Le bit est l'unité de base de l'information classique. Il peut être dans deux états différents : 0 ou 1. L'unité de base de l'information quantique s'appelle le **qubit**. Le propre des états quantiques est que toute combinaison linéaire de deux états quantiques est encore un état quantique, ce que l'on appelle **principe de superposition**. Ainsi, un qubit peut être dans l'état 0, 1 ou une combinaison des deux.
Cela se formalise de la manière suivante :

On se place dans l'espace de Hilbert complexe $\mathcal{H}$ de dimension 2 et on note $|0\rangle$ et $|1\rangle$ les vecteurs de la base canonique. Tout élément de cet espace s'écrit alors $\alpha|0\rangle+\beta|1\rangle$ avec $\alpha, \beta \in \mathbb{C}$. α et β s'appellent respectivement les **amplitudes** ou **phases** de $|0\rangle$ et de $|1\rangle$. On définit la relation d'équivalence suivante sur les éléments de $\mathcal{H}$: $v \sim u \Leftrightarrow \exists a \in \mathbb{C} \setminus \{0\}, v = au$.

> Un état quantique est une classe d'équivalence de vecteurs de $\mathcal{H}$. Un représentant canonique de cette classe sera un vecteur unitaire.

Ainsi, l'état d'un qubit peut être $|0\rangle$, $|1\rangle$ ou bien **l'état superposé** $\alpha|0\rangle + \beta|1\rangle$ avec $|\alpha|^2 + |\beta|^2 = 1$.

Remarque : En réalité, la définition que nous avons donnée est celle d'un état pur, et il existe d'autres types d'états quantiques (états mixtes). Mais cela est suffisant pour ce qui suit.

Évolution L'évolution dans le temps d'un état quantique est donné par l'équation de Schrödinger. Sans rentrer dans les détails, cela entraîne que cette évolution est décrite par un opérateur unitaire. En particulier, cette évolution est **déterministe** et **réversible**.

Observables et mesure Un observable est une propriété d'un système physique qui peut être mesurée, du moins en principe. Dans le formalisme quantique, un observable est un opérateur auto-adjoint A, c'est-à-dire, dans le cas fini, une matrice réelle symétrique $n \times n$ ($n = 2$ dans le cas d'un qubit). Nous savons que toute matrice réelle symétrique est diagonalisable et ses vecteurs propres sont 2-à-2 orthogonaux.

Plus précisément, si l'on note $|p_i\rangle$, $i = 1, ..., n$ ses vecteurs propres orthonormés, a_i la valeur propre associée, et que l'on définit $P_i := |p_i\rangle\langle p_i|$, les P_i sont des projeteurs orthonormaux sur l'espace propre correspondant au vecteur p_i. En effet :

- $P_i|p_i\rangle = |p_i\rangle\langle p_i|p_i\rangle = |p_i\rangle$
- $P_i|p_j\rangle = |p_i\rangle\langle p_i|p_j\rangle = 0$ pour $j \neq i$

Alors, le théorème spectral dit que A se décompose comme :

$$A = \sum_{i=1}^{n} a_i P_i$$

Mesurer un observable revient alors à projeter sur un espace, par exemple sur un ou plusieurs espaces propres, ou bien tout sous-espace d'une décomposition de l'espace de Hilbert en sous-espaces. Dans ce qui suit, nous allons considérer les projections sur les espaces propres.

Ainsi, dans le formalisme, la mesure d'un état quantique correspond à une opération $|v\rangle \mapsto P_i|v\rangle$.
Par conséquent, la mesure **modifie** l'état quantique. Si l'état avant la mesure est $|\phi\rangle$, l'état après la mesure est le résultat de la projection (normalisé, pour obtenir un représentant canonique) :
$\frac{P_i|\phi\rangle}{||P_i|\phi\rangle||} = \frac{P_i|\phi\rangle}{\sqrt{\langle\phi|P_i|\phi\rangle}}$

En revanche, la mesure est une opération **irréversible** et **probabiliste** car, si la mesure est bien une projection, le résultat de la projection est aléatoire. Tout ce que l'on peut calculer est la probabilité avec laquelle l'état sera projeté sur un sous-espace donné, avec la formule suivante. Cette probabilité est donnée par la formule $||P_i|\phi\rangle||^2 = \langle\phi|P_i|\phi\rangle$.

Systèmes composés Étant donné deux qubits A et B dont les états sont représentés par les vecteurs $|\phi_A\rangle \in \mathcal{H}_A$ et $|\phi_B\rangle \in \mathcal{H}_B$, l'état du système composé AB est donné par le produit tensoriel $|\phi_A\rangle \otimes |\phi_B\rangle \in \mathcal{H}_A \otimes \mathcal{H}_B$. Cela se généralise facilement à un nombre quelconque de qubits.

Quelques remarques et conventions de notation :

1. Lorsque l'on considère des états quantiques de N qubits, on omettra souvent le symbole du produit tensoriel. On écrira ainsi $|\phi_1\rangle...|\phi_N\rangle$, voire $|\phi_1...\phi_N\rangle$ au lieu de $|\phi_1\rangle \otimes ... \otimes |\phi_n\rangle$. On peut également mélanger ces deux abbréviations de la notation dans le but de distinguer entre des **registres** de qubits. Un registre correspond à un groupement de qubits qui a une signification particulière à un niveau d'abstraction plus élevé. Par exemple, si les K premiers qubits d'un système de N qubits correspondent à l'espace mémoire principal et les $N-K$ qubits restants correspondent à l'espace mémoire utilisé lors des calculs intermédiaires, on peut souligner les rôles distincts de ces deux registres en écrivant $|\phi_1...\phi_K\rangle|\phi_{K+1}...\phi_N\rangle$.
2. Il y a une autre écriture dont on se servira pour les systèmes à N qubits. Les éléments de la base canonique de $\mathcal{H}_1 \otimes ... \otimes \mathcal{H}_N$ sont en effet les

vecteurs $|0...0\rangle$, $|0...1\rangle$, ... , $|1...1\rangle$. On peut les voir comme l'écriture en binaire d'un nombre $0 \leq j \leq 2^N - 1$, c'est pourquoi nous écrirons parfois $|j\rangle$ pour le vecteur dont les coordonnées sont l'écriture en binaire de j.

3. Tout ce que nous avons dit au sujet d'un seul qubit reste valable pour plusieurs qubits. Ainsi, le principe de superposition reste valable, l'évolution est donnée par le produit tensoriel de deux opérations unitaires, la mesure par celui de deux opérations auto-adjointes.

A.3 Algorithmique quantique

A.3.1 Circuits quantiques

Il existe plusieurs modèles de calcul classique. Un modèle qui s'adapte bien au cadre quantique est le modèle des circuits.

Un algorithme est une suite finie d'étapes qui transforme une donnée, appelée entrée, en une autre donnée censée être la solution à un problème, appelée sortie. Un circuit modélise un algorithme en représentant la transformation effectuée sur l'entrée à l'aide d'opérations logiques élémentaires, appelées des portes logiques. On représente un circuit graphiquement en utilisant la convention suivante : les bits en entrée sont écrits à gauche, puis, dans l'ordre, on indique à l'aide de portes logiques ou d'autres circuits les opérations effectuées sur chaque bit ou groupement de bits.

Il y a divers résultats sur le modèle des circuits classique, par exemple l'ordre de grandeur des portes élémentaires nécessaires pour calculer tel ou tel algorithme. Un résultat important est celui de l'universalité des portes NAND ($(a, b) \mapsto 1 \oplus (a \wedge b)$) : cela signifie que tout algorithme peut se calculer à l'aide d'un circuit construit uniquement à partir de portes NANDs.

Un circuit quantique est comme un circuit classique, à ceci près que les portes logiques sont définies autrement. On rappelle qu'un état quantique peut subir deux types de transformations : l'évolution dans le temps (opérateur unitaire) et la mesure (projection, donc opérateur auto-adjoint). C'est le premier type de transformation qui nous intéressera principalement pour la construction de circuits quantiques, le deuxième type de transformation intervenant seulement au moment de la récupération du résultat de calcul.

Une porte quantique sur n qubits est ainsi une matrice $2^n \times 2^n$ unitaire.

Cela entraîne que toute porte quantique et donc tout circuit quantique est réversible.

A.3.2 Quelques portes quantiques élémentaires

Les portes de Pauli Ces portes agissent sur un seul qubit.

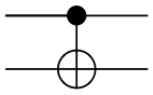
$X : \begin{cases} \|0\rangle \mapsto \|1\rangle \\ \|1\rangle \mapsto \|0\rangle \end{cases}$	$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
$Y : \begin{cases} \|0\rangle \mapsto i\|1\rangle \\ \|1\rangle \mapsto -i\|0\rangle \end{cases}$	$Z = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
$Z : \begin{cases} \|0\rangle \mapsto \|0\rangle \\ \|1\rangle \mapsto -\|1\rangle \end{cases}$	$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

La porte de Hadamard Cette porte agit sur un seul qubit.

$H : \begin{cases} \|0\rangle \mapsto \frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle) \\ \|1\rangle \mapsto \frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle) \end{cases}$	$H = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$

Les portes controllées
La porte $C - NOT$ agit sur deux qubits.
$C - NOT : |x\rangle|y\rangle \mapsto |x\rangle|y \oplus x\rangle$

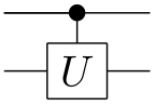
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$



Soit U une transformation unitaire sur n qubit. On définit la porte $C - U$ sur $n + 1$ qubits de la manière suivante :

$$C - U : \begin{cases} |0\rangle|x\rangle \mapsto |0\rangle|x\rangle \\ |1\rangle|x\rangle \mapsto |1\rangle|Ux\rangle \end{cases}$$

$$\begin{bmatrix} Id_2 & 0_2 \\ 0_2 & U \end{bmatrix}$$



La porte de Toffoli La porte de Toffoli est un exemple de porte $C - U$ avec $U = C - NOT$. Elle agit donc sur 3 qubits de la manière suivante :

$$C - U : \begin{cases} |0\rangle|y, z\rangle \mapsto |0\rangle|y, z\rangle \\ |1\rangle|y, z\rangle \mapsto |1\rangle|C - NOT(y, z)\rangle \end{cases}$$

$\Leftrightarrow$

$$C-U : \begin{cases} |0\rangle|y,z\rangle \mapsto |0\rangle|y,z\rangle \\ |1\rangle|0,z\rangle \mapsto |1\rangle|0,z\rangle \\ |1\rangle|1,z\rangle \mapsto |1\rangle|1,z\oplus 1\rangle \end{cases}$$
$\Leftrightarrow$
$C-U : |x,y,z\rangle \mapsto |x,y,z\oplus xy\rangle$
Cette porte est importante dans la mesure où, en fixant $z=1$, elle permet de calculer le NAND de x et y dans le troisième registre et ce de manière réversible. Cela permet de dire que tout algorithme calculable avec un ordinateur classique l'est avec un ordinateur quantique, car il suffit de remplacer les portes NAND par des portes Toffoli.

A.4 Quelques algorithmes quantiques

Transformée de Fourier quantique (QFT) La transformée de Fourier classique est l'application linéaire $v \mapsto \hat{v} := F_N v$ pour un vecteur $v \in \mathbb{R}^N$, où F_N est la matrice ayant les entrées suivantes :
$F_{N_{j,k}} = \frac{1}{\sqrt{N}} e^{2\pi i \frac{jk}{N}}$
C'est une matrice unitaire, donc son inverse F_N^{-1} est sa transposée conjuguée, i.e. la matrice ayant les entrées suivantes :
$F^{-1}_{N_{j,k}} = \frac{1}{\sqrt{N}} e^{-2\pi i \frac{jk}{N}}$
On a alors $\hat{v} = \sum_{k=0}^{N-1} \frac{1}{\sqrt{N}} e^{2\pi i \frac{jk}{N}} v_k$

La transformée de Fourier quantique consiste à appliquer cette transformation à un état quantique $|\phi\rangle = \sum_{k=0}^{N-1} \alpha_k |k\rangle$ (où $N = 2^n$). Il s'agit de l'opération suivante sur chaque élément $|k\rangle$ de la base :
$|k\rangle \mapsto F_N|k\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{2\pi i \frac{jk}{N}} |j\rangle$.

Il est possible d'implémenter QFT à l'aide de $O(n\log(n))$ circuits (Hadamard, et R_1, R_2, R_3 avec $R_s = \left[\begin{smallmatrix} 1 & 0 \\ 0 & e^{\frac{2i\pi}{2^s}} \end{smallmatrix}\right]$) avec une petite erreur [6].

La transformée de Fourier quantique est au coeur de la majorité des algorithmes quantiques où il y a une amélioration considérable par rapport aux algorithmes classiques. Par exemple, elle joue un rôle dans l'algorithme quantique d'estimation de phase qui va nous intéresser puisque les marches aléatoires quantiques comportent une étape d'estimation de phase.

Estimation de phase Il s'agit de résoudre le problème suivant :

Problème 9 (Estimation de phase). *Étant donné U unitaire (donc telle que $||U|v\rangle|| = |||v\rangle|| \ \forall\ v$ et $|\psi\rangle$ un vecteur propre de U avec la valeur propre $\lambda = e^{2i\pi\phi}$, estimer ϕ avec n bits de précision (cela permet alors de calculer λ).*

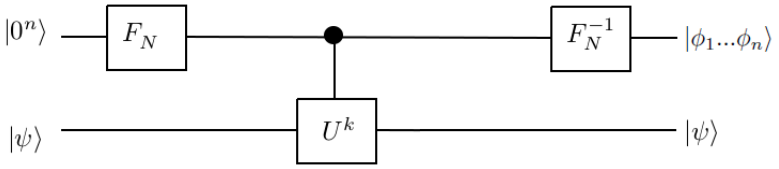
Note : On suppose que $\phi \in [0,1[$, $\phi = 0,\phi_1...\phi_n...$ en binaire.

Cet algorithme effectue alors les étapes suivantes :

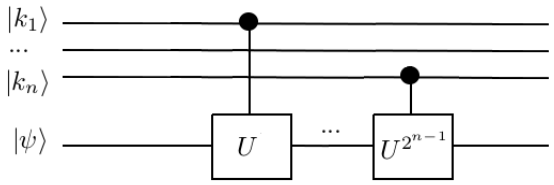
$$|0^n\rangle|\psi\rangle \xrightarrow{F_N\otimes Id} \frac{1}{\sqrt{N}}\sum_{k=0}^{N-1}|k\rangle|\psi\rangle$$
$$\xrightarrow{Id\otimes U^k} \frac{1}{\sqrt{N}}\sum_{k=0}^{N-1}e^{2i\pi k\phi}|k\rangle|\psi\rangle \approx (F_N\otimes Id)(|\phi_1...\phi_n\rangle|\psi\rangle)(*)$$
$$\xrightarrow{F_N^{-1}\otimes Id} |\phi_1...\phi_n\rangle|\psi\rangle$$

(*) U^k désigne ici la porte qui applique k fois la porte U au second registre, où k est le nombre binaire dans le premier registre. Par conséquent, $U^k|\phi\rangle = \lambda^k|\phi\rangle = e^{2i\pi k\phi}|\phi\rangle$.
Puis, on a $F_N|\phi_1...\phi_n\rangle = \frac{1}{\sqrt{N}}\sum_{j=0}^{N-1}e^{2\pi ij\frac{(\phi_1...\phi_n)_2}{2^n}}|j\rangle = \frac{1}{\sqrt{N}}\sum_{j=0}^{N-1}e^{2\pi ij0,\phi_1...\phi_n}|j\rangle$, d'où l'égalité approximative obtenue à cette étape.

Il suffit donc de mesurer les n premiers qubits pour obtenir la phase avec n bits de précision. Ainsi le circuit de cet algorithme est le suivant :



Si l'on écrit $k = (k_n...k_1)_2$ en binaire, le circuit suivant donne les détails de l'implémentation de U^k :



B Preuves des développements limités

B.0.1 Préliminaires et résultats communs

Fonction d'entropie H

$$\begin{aligned}
H(x) &= -x\log_2(x) - (1-x)\log_2(1-x) \\
&= \frac{1}{\ln(2)}\left(-x\ln(x) - (1-x)\ln(1-x)\right) \\
&= \frac{1}{\ln(2)}\left(-x\ln(x) - (1-x)(-x - \frac{x^2}{2} + o(x^2))\right) \\
&= \frac{1}{\ln(2)}\left(-x\ln(x) + x + \frac{x^2}{2} - x^2 - \frac{x^3}{2} + o(x^2)\right) \\
&= \frac{1}{\ln(2)}\left(-x\ln(x) + x - \frac{x^2}{2} + o(x^2)\right)
\end{aligned}$$

Dérivées de la fonction d'entropie H

$$\begin{aligned}
H'(x) &= \log_2(\frac{1-x}{x}) \\
H''(x) &= \frac{1}{\ln(2)}\frac{1}{x(x-1)}
\end{aligned}$$

Développement limité de $(R+\eta)H(\frac{\pi}{R+\eta})$
$(R+\eta)H(\frac{\pi}{R+\eta}) =$
$\frac{1}{\ln(2)}\left((R+\eta)(-\frac{\pi}{R+\eta}\ln(\frac{\pi}{R+\eta}) + \frac{\pi}{R+\eta} + o(\pi+\eta)\right)$
$\frac{1}{\ln(2)}\left(-\pi\ln(\frac{\pi}{R+\eta}) + \pi + o(\pi+\eta)\right)$
$\frac{1}{\ln(2)}\left(-\pi\ln(\pi) + \pi\ln(R+\eta) + \pi + o(\pi+\eta)\right)$
$\frac{1}{\ln(2)}\left(-\pi\ln(\pi) + \pi\left(\ln(R) + \ln(1+\frac{\eta}{R}) + 1\right) + o(\pi+\eta)\right)$
$\frac{1}{\ln(2)}\left(-\pi\ln(\pi) + \pi\left(\ln(R) + \frac{\eta}{R} + o(\eta) + 1\right) + o(\pi+\eta)\right)$
$\frac{1}{\ln(2)}\left(-\pi\ln(\frac{\pi}{R}) + \pi + \frac{\pi\eta}{R} + o(\pi+\eta)\right)$

Développement limité de $(1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta})$
Première étape : $\frac{\tau-\pi}{1-R-\eta} = \frac{\tau-\pi}{1-R}\frac{1}{1-\frac{\eta}{1-R}}$
$\frac{\tau-\pi}{1-R}(1+\frac{\eta}{1-R}) + o(\pi+\eta)$
$\frac{\tau-\pi}{1-R} + \frac{(\tau-\pi)\eta}{(1-R)^2} + o(\pi+\eta)$
$\frac{\tau}{1-R} - \frac{\pi}{1-R} + \frac{\tau\eta}{(1-R)^2} - \frac{\pi\eta}{(1-R)^2} + o(\pi+\eta)$
$\frac{\tau}{1-R} - \frac{\pi}{1-R} + \frac{\tau\eta}{(1-R)^2} + o(\pi+\eta)$
$\frac{\tau}{1-R} + \epsilon + o(\pi)$
Avec $\epsilon = -\frac{\pi}{1-R} + \frac{\tau\eta}{(1-R)^2}$

Deuxième étape : série de Taylor de $H(\frac{\tau-\pi}{1-R-\eta})$:
$H(\frac{\tau-\pi}{1-R-\eta}) = H(\frac{\tau}{1-R} + \epsilon) =$
$H(\frac{\tau}{1-R}) + \epsilon H'(\frac{\tau}{1-R}) + o(\epsilon)$
$H(\frac{\tau}{1-R}) + \epsilon\alpha + o(\epsilon)$ avec $\alpha = \log_2\left(\frac{1-R-\tau}{\tau}\right)$

Développement limité de $H(\tau) - (1 - R - \eta)H(\frac{\tau-\pi}{1-R-\eta})$
$H(\tau) - (1 - R - \eta)H(\frac{\tau-\pi}{1-R-\eta}) =$
$H(\tau) - (1 - R - \eta)\left(H(\frac{\tau}{1-R}) + \epsilon\alpha + o(\epsilon)\right)$
$H(\tau) - (1 - R)H(\frac{\tau}{1-R}) - (1 - R)\epsilon\alpha + \eta H(\frac{\tau}{1-R}) + \epsilon\alpha\eta + o(\epsilon)$
$H(\tau) - (1 - R)H(\frac{\tau}{1-R}) - (1 - R)\epsilon\alpha + \eta H(\frac{\tau}{1-R}) + o(\pi + \eta)$

B.0.2 Algorithmes sans technique des représentations

Il s'agit des algorithmes suivants : Dumer/Shamir-Schroeppel classique (D, SS) qui ont la même complexité temporelle, Dumer quantique avec Grover (DG), Shamir-Schroeppel quantique avec Grover (SSG) et Shamir-Schroeppel quantique avec Quantum Walk ($SSQW$).

La complexité de ces algorithmes s'écrit $F(\pi, \eta) = H(\tau) - (1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta}) - c(R+\eta)H(\frac{\pi}{R+\eta})$ avec $0 \leq c \leq 1$.

De plus, il y a une relation de dépendance entre les paramètres de la forme $2^h = \binom{k+h}{p}^s$ avec $0 \leq s \leq 1$. Cette relation de dépendance est vérifiée ssi la fonction G définie ci-dessous est nulle.

$G(\pi, \eta) := \eta - s(R + \eta)H(\frac{\pi}{R+\eta}) \Leftrightarrow \eta = s(R + \eta)H(\frac{\pi}{R+\eta})$

On peut en déduire la valeur de η en fonction de π. En effet, nous pouvons nous servir du développement limité de $(R + \eta)H(\frac{\pi}{R+\eta})$ que nous avons déjà calculé. Nous obtenons ainsi :

$$\begin{aligned} G(\pi, \eta) &= \eta - s(R + \eta)H(\frac{\pi}{R + \eta}) \\ &= \eta - \frac{s}{\ln(2)}\left(-\pi \ln(\frac{\pi}{R}) + \pi + \frac{\pi\eta}{R} + o(\pi + \eta)\right) \\ &= \eta(1 - \frac{s}{\ln(2)}\frac{\pi}{R}) - \frac{s}{\ln(2)}\left(-\pi \ln(\frac{\pi}{R}) + \pi\right) + o(\pi + \eta) \end{aligned}$$

D'où :

$\eta = s\frac{-\pi \ln\left(\frac{\pi}{R}\right)+\pi+o(\pi+\eta)}{(\ln(2)-s\frac{\pi}{R})}$
On en déduit :

$$\begin{aligned} \eta &= \frac{s}{\ln(2)}\frac{-\pi \ln\left(\frac{\pi}{R}\right) + \pi + o(\pi + \eta)}{(1 - \frac{s\pi}{\ln(2)R})} \\ &= \frac{s}{\ln(2)}\left(-\pi \ln\left(\frac{\pi}{R}\right) + \pi + o(\pi + \eta)\right)(1 + \frac{s\pi}{\ln(2)R} + o(\pi)) \\ &= \frac{s}{\ln(2)}\left(-\pi \ln\left(\frac{\pi}{R}\right) + \pi\right)o(\pi + \eta) \end{aligned}$$

Conclusion :

$$\boxed{\eta = \frac{s}{\ln(2)}\left(-\pi \ln\left(\frac{\pi}{R}\right) + \pi\right) + o(\pi)}$$

Valeurs de c et s

Algorithme	c	s
D/SS	$1/2$	$1/2$
DG	$1/3$	1
SSG	$1/4$	$1/2$
$SSQW$	$2/5$	$2/5$

Développement limité de $F(\pi,\eta)$ et analyse

$F(\pi,\eta) =$
$H(\tau) - (1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta}) - c(R+\eta)H(\frac{\pi}{R+\eta}) =$
$H(\tau) - (1-R)H(\frac{\tau}{1-R}) - (1-R)\epsilon\alpha + \eta H(\frac{\tau}{1-R}) - \frac{c}{s}\eta + o(\pi)$
$H(\tau) - (1-R)H(\frac{\tau}{1-R}) - (1-R)\left(-\frac{\pi}{1-R} + \frac{\tau\eta}{(1-R)^2}\right)\alpha + \eta\left(H(\frac{\tau}{1-R}) - \frac{c}{s}\right) + o(\pi)$
$H(\tau) - (1-R)H(\frac{\tau}{1-R}) + \pi\alpha + \eta\left(H(\frac{\tau}{1-R}) - \frac{c}{s} - \frac{\alpha\tau}{(1-R)}\right) + o(\pi)$
$H(\tau) - (1-R)H(\frac{\tau}{1-R}) + \pi\alpha + \frac{1}{\ln(2)}\left(s\pi\ln\left(\frac{\pi}{R}\right) - s\pi\right)\left(H(\frac{\tau}{1-R}) - \frac{c}{s} - \frac{\alpha\tau}{(1-R)}\right) + o(\pi)$
$H(\tau) - (1-R)H(\frac{\tau}{1-R}) + \pi\ln\left(\frac{\pi}{R}\right)\left(s\frac{\frac{c}{s} + \frac{\alpha\tau}{(1-R)} - H(\frac{\tau}{1-R})}{\ln(2)}\right) + \pi\left(\alpha + s\frac{H(\frac{\tau}{1-R}) - \frac{c}{s} - \frac{\alpha\tau}{(1-R)}}{\ln(2)}\right) + o(\pi)$

$$\boxed{F(\pi,\eta) = \textsc{Prange} + a\pi\ln\left(\frac{\pi}{R}\right) + (\alpha - a)\,\pi + o(\pi)}$$
$$\text{Avec } a = a(c,s) = s\frac{\frac{c}{s} + \frac{\alpha\tau}{(1-R)} - H(\frac{\tau}{1-R})}{\ln(2)}$$

Nous donnons les valeurs concrètes de a pour chacun des quatre algorithmes dans le tableau qui suit.

Algorithme	c	s	$a(c,s)$
D/SS	$1/2$	$1/2$	$\frac{1 + \frac{\alpha\tau}{(1-R)} - H(\frac{\tau}{1-R})}{2\ln(2)}$
DG	$1/3$	1	$\frac{\frac{1}{3} + \frac{\alpha\tau}{(1-R)} - H(\frac{\tau}{1-R})}{\ln(2)}$
SSG	$1/4$	$1/2$	$\frac{\frac{1}{2} + \frac{\alpha\tau}{(1-R)} - H(\frac{\tau}{1-R})}{2\ln(2)}$
$SSQW$	$2/5$	$2/5$	$2\frac{1 + \frac{\alpha\tau}{(1-R)} - H(\frac{\tau}{1-R})}{5\ln(2)}$

Si l'on pose $f(\pi) := a\pi\ln\left(\frac{\pi}{R}\right) + (\alpha - a)\,\pi$, on a :

$$\begin{aligned} f'(\pi) &= a\left(1 + \ln\left(\frac{\pi}{R}\right)\right) + \alpha - a \\ &= a\ln\left(\frac{\pi}{R}\right) + \alpha \end{aligned}$$

On cherche le minimum.

$f'(\pi) = 0$
$\Leftrightarrow \ln\left(\frac{\pi}{R}\right) = -\frac{\alpha}{a}$
$\Leftrightarrow \pi = Re^{-\frac{\alpha}{a}}$

On réinjecte cette valeur de π dans la formule pour avoir le minimum.

$\textsc{Prange} + a\pi \ln\left(\frac{\pi}{R}\right) + (\alpha - a)\,\pi + o(\pi)$
$\textsc{Prange} + aRe^{-\frac{\alpha}{a}} \ln\left(\frac{Re^{-\frac{\alpha}{a}}}{R}\right) + (\alpha - a)\,Re^{-\frac{\alpha}{a}} + o(\pi)$
$\textsc{Prange} - \alpha Re^{-\frac{\alpha}{a}} + (\alpha - a)\,Re^{-\frac{\alpha}{a}} + o(\pi)$
$\textsc{Prange} - aRe^{-\frac{\alpha}{a}} + o(\pi)$
En testant plusieurs valeurs numériquement, on fait le constat suivant : à c constant, plus s est petit, mieux c'est. A s constant, plus c est grand, mieux c'est.

B.0.3 Algorithme MMT

Pour MMT classique il n'y a pas de dépendance entre les paramètres η et π. Néanmoins, on peut artificiellement supposer qu'il y en a une de la même forme qu'avant, i.e. $2^h = \binom{k+h}{p}^s$ avec $0 \leq s$ (s n'est plus borné par-dessus par 1 et s'il a une borne supérieure, elle est a priori inconnue). On obtient ainsi une relation de dépendance de la forme $G(\pi, \eta) = 0$ avec $G(\pi, \eta) = \eta - s(R + \eta)H(\frac{\pi}{R+\eta})$. C'est-à-dire, en reprenant les calculs faits ci-dessus :

$$\boxed{\eta = \frac{s}{\ln(2)} \left(-\pi \ln\left(\frac{\pi}{R}\right) + \pi \right) + o(\pi)}$$

Ensuite, comme nous l'avons vu, le minimum est obtenu en calculant le maximum de trois fonctions. Nous pouvons donc calculer les développement limités des trois fonctions un par un et comparer ensuite. En d'autres termes, si on reprend la même fonction F que ci-dessus avec $c = 1$, i.e. $F(\pi, \eta) = H(\tau) - (1 - R - \eta)H(\frac{\tau-\pi}{1-R-\eta}) - (R+\eta)H(\frac{\pi}{R+\eta})$, alors il faut regarder les trois fonctions suivantes :

- $F_1(\pi, \eta) = F(\pi, \eta) + \frac{R+\eta}{2} H(\frac{\pi}{2(R+\eta)})$
- $F_2(\pi, \eta) = F(\pi, \eta) + (R + \eta)H(\frac{\pi}{2(R+\eta)}) - \pi$
- $F_3(\pi, \eta) = F(\pi, \eta) + 2(R + \eta)H(\frac{\pi}{2(R+\eta)}) - \pi - \eta$

Développement limité de $(R+\eta)H(\frac{\pi}{2(R+\eta)})$ Nous allons le calculer car c'est un terme qui intervient dans les trois fonctions F_i. Pour ce faire, nous allons reprendre le calcul du DL de $(R + \eta)H(\frac{\pi}{R+\eta})$. $(R + \eta)H(\frac{\pi}{2(R+\eta)}) =$

$\frac{1}{\ln(2)} \left((R + \eta)(-\frac{\pi}{2(R+\eta)} \ln(\frac{\pi}{2(R+\eta)}) + \frac{\pi}{2(R+\eta)} + o(\pi + \eta) \right)$

$\frac{1}{\ln(2)} \left(-\frac{\pi}{2} \ln(\frac{\pi}{2(R+\eta)}) + \frac{\pi}{2} + o(\pi + \eta) \right)$

$\frac{1}{\ln(2)} \left(-\frac{\pi}{2} \ln(\pi) + \frac{\pi}{2} \ln(2) + \frac{\pi}{2} \ln(R + \eta) + \frac{\pi}{2} + o(\pi + \eta) \right)$

$\frac{1}{2\ln(2)} \left(-\pi \ln(\pi) + \pi \left(\ln(2) + \ln(R) + \ln(1 + \frac{\eta}{R}) + 1 \right) + o(\pi + \eta) \right)$

$\frac{1}{2\ln(2)} \left(-\pi \ln(\pi) + \pi \left(\ln(2) + \ln(R) + \frac{\eta}{R} + o(\eta) + 1 \right) + o(\pi + \eta) \right)$

$\frac{1}{2\ln(2)} \left(-\pi \ln(\frac{\pi}{R}) + \pi \left(1 + \ln(2) \right) + \frac{\pi\eta}{R} + o(\pi + \eta) \right)$

$\frac{1}{2\ln(2)} \left(-\pi \ln(\frac{\pi}{R}) + \pi \left(1 + \ln(2) \right) \right) + o(\pi)$

Développement limité de $F(\pi, \eta)$ avec $c = 1$ On reprend les calculs faits ci-dessus en remplaçant c par sa valeur.

$$\boxed{F(\pi, \eta) = \textsc{Prange} + a\pi \ln\left(\frac{\pi}{R}\right) + (\alpha - a)\,\pi + o(\pi)}$$

Avec $a = a(s) = s\frac{\frac{1}{s} + \frac{\alpha\tau}{(1-R)} - H(\frac{\tau}{1-R})}{\ln(2)} = \frac{1 + s\left(\frac{\alpha\tau}{(1-R)} - H(\frac{\tau}{1-R})\right)}{\ln(2)}$

Développement limité de $F_1(\pi, \eta)$

$$\begin{aligned}
F_1(\pi,\eta) &= F(\pi,\eta) + \frac{R+\eta}{2} H(\frac{\pi}{2(R+\eta)}) \\
&= \textsc{Prange} + a\pi \ln\left(\frac{\pi}{R}\right) + (\alpha - a)\,\pi + \frac{1}{4\ln(2)}\left(-\pi \ln(\frac{\pi}{R}) + \pi\,(1+\ln(2))\right) + o(\pi) \\
&= \textsc{Prange} + \left(a - \frac{1}{4\ln(2)}\right)\pi \ln\left(\frac{\pi}{R}\right) + \left(\alpha + \frac{\ln(2)}{4\ln(2)} - a + \frac{1}{4\ln(2)}\right)\pi + o(\pi) \\
&= \textsc{Prange} + \left(a - \frac{1}{4\ln(2)}\right)\pi \ln\left(\frac{\pi}{R}\right) + \left(\left(\alpha + \frac{1}{4}\right) - \left(a - \frac{1}{4\ln(2)}\right)\right)\pi + o(\pi) \\
&= \textsc{Prange} + b_1 \pi \ln\left(\frac{\pi}{R}\right) + (\beta_1 - b_1)\,\pi + o(\pi)
\end{aligned}$$

Avec :

$\beta_1 = \alpha + \frac{1}{4} = \log_2\left(\frac{1-R-\tau}{\tau}\right) + \frac{1}{4}$
$b_1 = b_1(s) = a(s) - \frac{1}{4\ln(2)} = \frac{1+s\left(\frac{\alpha\tau}{(1-R)} - H(\frac{\tau}{1-R})\right)}{\ln(2)} - \frac{1}{4\ln(2)} = \frac{\frac{3}{4}+s\left(\frac{\alpha\tau}{(1-R)} - H(\frac{\tau}{1-R})\right)}{\ln(2)}$

Développement limité de $F_2(\pi, \eta)$

$$\begin{aligned}
F_2(\pi,\eta) &= F(\pi,\eta) + (R+\eta) H(\frac{\pi}{2(R+\eta)}) - \pi \\
&= \textsc{Prange} + a\pi \ln\left(\frac{\pi}{R}\right) + (\alpha - a)\,\pi + \frac{1}{2\ln(2)}\left(-\pi \ln(\frac{\pi}{R}) + \pi\,(1+\ln(2))\right) - \pi + o(\pi) \\
&= \textsc{Prange} + a\pi \ln\left(\frac{\pi}{R}\right) + (\alpha - a)\,\pi + \frac{1}{2\ln(2)}\left(-\pi \ln(\frac{\pi}{R}) + \pi\right) + \frac{\pi}{2} - \pi + o(\pi) \\
&= \textsc{Prange} + \left(a - \frac{1}{2\ln(2)}\right)\pi \ln\left(\frac{\pi}{R}\right) + \left(\alpha - \frac{1}{2} - a + \frac{1}{2\ln(2)}\right)\pi + o(\pi) \\
&= \textsc{Prange} + \left(a - \frac{1}{2\ln(2)}\right)\pi \ln\left(\frac{\pi}{R}\right) + \left(\left(\alpha - \frac{1}{2}\right) - \left(a - \frac{1}{2\ln(2)}\right)\right)\pi + o(\pi) \\
&= \textsc{Prange} + b_2 \pi \ln\left(\frac{\pi}{R}\right) + (\beta_2 - b_2)\,\pi + o(\pi)
\end{aligned}$$

Avec :

$\beta_2 = \alpha - \frac{1}{2} = \log_2\left(\frac{1-R-\tau}{\tau}\right) - \frac{1}{2}$
$b_2 = b_2(s) = a(s) - \frac{1}{2\ln(2)} = \frac{1+s\left(\frac{\alpha\tau}{(1-R)} - H(\frac{\tau}{1-R})\right)}{\ln(2)} - \frac{1}{2\ln(2)} = \frac{\frac{1}{2}+s\left(\frac{\alpha\tau}{(1-R)} - H(\frac{\tau}{1-R})\right)}{\ln(2)}$

Développement limité de $F_3(\pi, \eta)$

$$
\begin{aligned}
F_3(\pi,\eta) &= F(\pi,\eta) + 2(R+\eta)H(\frac{\pi}{2(R+\eta)}) - \pi - \eta \\
&= \textsc{Prange} + a\pi \ln\left(\frac{\pi}{R}\right) + (\alpha - a)\,\pi + \frac{1}{\ln(2)}\left(-\pi\ln(\frac{\pi}{R}) + \pi\,(1+\ln(2))\right) - \pi - \eta + o(\pi) \\
&= \textsc{Prange} + a\pi \ln\left(\frac{\pi}{R}\right) + (\alpha - a)\,\pi + \frac{1}{\ln(2)}\left(-\pi\ln(\frac{\pi}{R}) + \pi\right) + \pi - \pi - \eta + o(\pi) \\
&= \textsc{Prange} + \left(a - \frac{1}{\ln(2)}\right)\pi\ln\left(\frac{\pi}{R}\right) + \left(\alpha - a + \frac{1}{\ln(2)}\right)\pi - \eta + o(\pi) \\
&= \textsc{Prange} + \left(a - \frac{1}{\ln(2)}\right)\pi\ln\left(\frac{\pi}{R}\right) + \left(\alpha - a + \frac{1}{\ln(2)}\right)\pi - \left(\frac{s}{\ln(2)}\left(-\pi\ln\left(\frac{\pi}{R}\right) + \pi\right)\right) + o(\pi) \\
&= \textsc{Prange} + \left(a - \frac{1}{\ln(2)} + \frac{s}{\ln(2)}\right)\pi\ln\left(\frac{\pi}{R}\right) + \left(\alpha - \left(a - \frac{1}{\ln(2)} + \frac{s}{\ln(2)}\right)\right)\pi + o(\pi) \\
&= \textsc{Prange} + b_3\pi\ln\left(\frac{\pi}{R}\right) + (\beta_3 - b_3)\,\pi + o(\pi)
\end{aligned}
$$

Avec :

$\beta_3 = \alpha = \log_2\left(\frac{1-R-\tau}{\tau}\right)$

$b_3 = b_3(s) = a(s) - \frac{1}{2\ln(2)} + \frac{s}{\ln(2)} = \frac{1 + s\left(\frac{\alpha\tau}{(1-R)} - H(\frac{\tau}{1-R})\right)}{\ln(2)} + \frac{s-1}{\ln(2)} = \frac{s\left(\frac{\alpha\tau}{(1-R)} - H(\frac{\tau}{1-R}) + 1\right)}{\ln(2)}$

Calcul final Pour un s fixé, en posant $f_i(\pi) := b_i\pi\ln\left(\frac{\pi}{R}\right) + (\beta_i - b_i)\,\pi$, par un calcul analogue à celui qui a été fait avant, on montre que le minimum est atteint au point $\pi_i = Re^{-\frac{\beta_i}{b_i}}$.

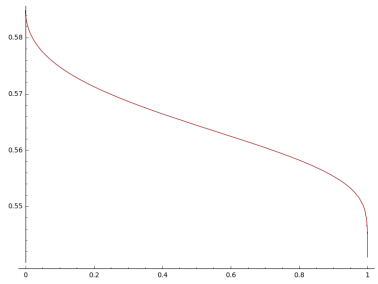
Ces valeurs de π_i sont en fait fonction de s que l'on doit optimiser plus tard. Mais pour l'instant, on les injecte dans les formules afin d'avoir le point minimum.

$$
\begin{aligned}
\min_\pi f_i &= b_i\pi_i\ln\left(\frac{\pi_i}{R}\right) + (\beta_i - b_i)\,\pi_i + o(\pi) \\
&= b_i R e^{-\frac{\beta_i}{b_i}}\ln\left(\frac{Re^{-\frac{\beta_i}{b_i}}}{R}\right) + (\beta_i - b_i)\,Re^{-\frac{\beta_i}{b_i}} + o(\pi) \\
&= -b_i R e^{-\frac{\beta_i}{b_i}} + o(\pi)
\end{aligned}
$$

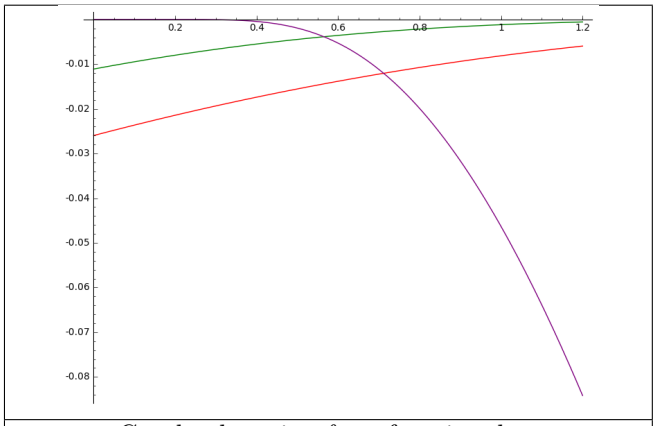
On peut enfin optimiser s. En effet, la complexité de la fonction est donnée par le maximum des $\min_\pi f_i$, $i = 1, 2, 3$. Il faut donc choisir s afin de minimiser ce maximum. Autrement dit, s doit minimiser $\max_{i=1,2,3}\left(-b_i R e^{-\frac{\beta_i}{b_i}}\right)$, c'est-à-dire :

s doit maximiser $\min_{i=1,2,3}\left(b_i R e^{-\frac{\beta_i}{b_i}}\right)$.

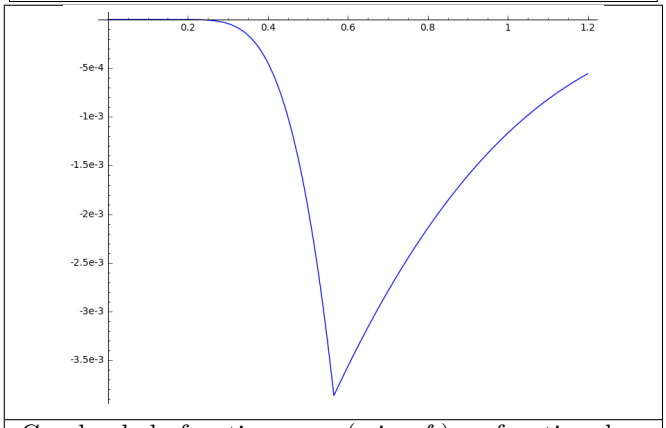
En faisant ce calcul numériquement nous voyons que ce minimum du maximum est atteint pour $0,54 < s < 0,59$. La courbe ci-dessous trace la valeur exacte de s en fonction de R.



Avec cette valeur de s, le maximum est donné par les valeurs des fonctions F_2 et F_3 qui dans ce cas sont égales. À titre d'exemple, nous avons donné ci-dessous les courbes des $\min_\pi f_i$ ainsi que de leur maximum pour $R = 0,4675$.



Courbe des $\min_\pi f_i$ en fonction de s
$\min_\pi f_1$ en rouge, $\min_\pi f_2$ en vert, $\min_\pi f_3$ en mauve



Courbe de la fonction $\max_i(\min_\pi f_i)$ en fonction de s

Autre constat : si l'on reprend les fonctions définies dans la dernière partie $F(\pi,\eta) = H(\tau) - (1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta}) - c(R+\eta)H(\frac{\pi}{R+\eta})$

et
$G(\pi,\eta) := \eta - s(R+\eta)H(\frac{\pi}{R+\eta})$
la complexité de MMT est donnée, à première vue, par les choix de paramètres $c = 1$ (le choix optimal) et $0,54 < s < 0,59$ (choix moyen) auquel on ajoute un autre terme qui est, pour le choix optimal de π, indifféremment $(R+\eta)H(\frac{\pi}{2(R+\eta)})-\pi$ ou $2(R+\eta)H(\frac{\pi}{2(R+\eta)})-\pi-\eta$.

En realité, il est possible de choisir les paramètres de MMT de cette façon, mais $c \neq 1$. Pour voir cela, nous allons considérer la seconde expression $(2(R+\eta)H(\frac{\pi}{2(R+\eta)})-\pi-\eta)$ et nous allons reprendre son développement limité.

Tout d'abord, pour pouvoir comparer plus facilement avec les autres algorithmes, notons que l'on a l'égalité suivante entre les développements limités de $(R+\eta)H(\frac{\pi}{2(R+\eta)})$ et de $(R+\eta)H(\frac{\pi}{R+\eta})$:

$$\begin{aligned}(R+\eta)H(\frac{\pi}{2(R+\eta)}) &= \frac{1}{2\ln(2)}\left(-\pi\ln(\frac{\pi}{R})+\pi\left(1+\ln(2)\right)\right)+o(\pi)\\ &= \frac{1}{2}\frac{1}{\ln(2)}\left(-\pi\ln(\frac{\pi}{R})+\pi\right)+\frac{\pi}{2}+o(\pi)\\ &= \frac{1}{2}(R+\eta)H(\frac{\pi}{R+\eta})+\frac{\pi}{2}+o(\pi)\end{aligned}$$

Par conséquent : $2(R+\eta)H(\frac{\pi}{2(R+\eta)})-\pi-\eta =$
$(R+\eta)H(\frac{\pi}{R+\eta})+\pi-\pi-\eta =$
$(R+\eta)H(\frac{\pi}{R+\eta})-\eta$

Et donc on peut injecter cela dans l'expression de F afin d'obtenir :

$F^*(\pi,\eta) =$
$F(\pi,\eta)+2(R+\eta)H(\frac{\pi}{2(R+\eta)})-\pi-\eta =$
$H(\tau)-(1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta})-(R+\eta)H(\frac{\pi}{R+\eta})+(R+\eta)H(\frac{\pi}{R+\eta})-\eta$
$H(\tau)-(1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta})-\eta$
$H(\tau)-(1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta})-s(R+\eta)H(\frac{\pi}{R+\eta})$
Donc on peut en réalité considérer que les paramètres de MMT sont $c = s$.

B.0.4 Algorithme MMTQW

L'exposant est donnée par la fonction $F_0(\pi,\eta) = F(\pi,\eta)+\frac{6(R+\eta)}{5}H(\frac{\pi}{2(R+\eta)})-\pi$, où $F(\pi,\eta) = H(\tau)-(1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta})-(R+\eta)H(\frac{\pi}{R+\eta})$.

La dépendance entre les paramètres η et π change de forme : elle devient $2^h = \binom{k+h}{\frac{p}{2}}^{4/5}$, i.e. on a une une relation de dépendance de la forme $G(\pi,\eta) = 0$ avec $G(\pi,\eta) = \eta - s(R+\eta)H(\frac{\pi}{2(R+\eta)})$ où $s = 4/5$. On obtient ainsi un développement limité différent pour η. Pour le calculer, on reprend le calcul du développement limité de $(R+\eta)H(\frac{\pi}{2(R+\eta)})$:

$(R+\eta)H(\frac{\pi}{2(R+\eta)}) = \frac{1}{2\ln(2)}\left(-\pi\ln(\frac{\pi}{R}) + \pi\left(1+\ln(2)\right) + \frac{\pi\eta}{R} + o(\pi+\eta)\right)$

$$\begin{aligned} G(\pi,\eta) &= \eta - s(R+\eta)H(\frac{\pi}{2(R+\eta)}) \\ &= \eta - \frac{s}{2\ln(2)}\left(-\pi\ln(\frac{\pi}{R}) + \pi\left(1+\ln(2)\right) + \frac{\pi\eta}{R} + o(\pi+\eta)\right) \\ &= \eta(1 - \frac{s}{2\ln(2)}\frac{\pi}{R}) - \frac{s}{2\ln(2)}\left(-\pi\ln(\frac{\pi}{R}) + \pi\left(1+\ln(2)\right)\right) + o(\pi+\eta) \end{aligned}$$

D'où $\eta = \frac{s}{2\ln(2)}\frac{-\pi\ln(\frac{\pi}{R})+\pi(1+\ln(2))+o(\pi+\eta)}{1-\frac{s}{2\ln(2)}\frac{\pi}{R}}$

On en déduit :

$$\begin{aligned} \eta &= \frac{s}{2\ln(2)}\left(-\pi\ln\left(\frac{\pi}{R}\right) + \pi\left(1+\ln(2)\right) + o(\pi+\eta)\right)\left(1 + \frac{s\pi}{2\ln(2)R} + o(\pi+\eta)\right) \\ &= \frac{s}{2\ln(2)}\left(-\pi\ln\left(\frac{\pi}{R}\right) + \pi\left(1+\ln(2)\right)\right) + o(\pi) \end{aligned}$$

Conclusion (en injectant $s = \frac{4}{5}$) :

$$\boxed{\eta = \frac{2}{5\ln(2)}\left(-\pi\ln\left(\frac{\pi}{R}\right) + \pi\left(1+\ln(2)\right)\right) + o(\pi)}$$

Pour pouvoir comparer plus facilement avec les autres algorithmes, rappelons l'égalité suivante entre les développements limités de $(R+\eta)H(\frac{\pi}{2(R+\eta)})$ et de $(R+\eta)H(\frac{\pi}{R+\eta})$:

$$(R+\eta)H(\frac{\pi}{2(R+\eta)}) = \frac{1}{2}(R+\eta)H(\frac{\pi}{R+\eta}) + \frac{\pi}{2} + o(\pi)$$

Alors, $\eta = s(R+\eta)H(\frac{\pi}{2(R+\eta)}) \Leftrightarrow \eta = \frac{s}{2}(R+\eta)H(\frac{\pi}{R+\eta}) + s\frac{\pi}{2} + o(\pi)$
C'est-à-dire que pour la même valeur de s et de π, la valeur de η pour $MMTQW$ est un peu plus que la moitié de sa valeur pour les autres algorithmes. Pour le dire plus exactement, on peut écrire $\eta = s_0(R+\eta)H(\frac{\pi}{R+\eta})$ avec $s_0 = \frac{s}{2} + s\frac{\pi}{2(R+\eta)H(\frac{\pi}{R+\eta})}$.

Puisque la relation de dépendance a changé, on doit recalculer le développement limité de $F(\pi,\eta) = H(\tau) - (1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta}) - (R+\eta)H(\frac{\pi}{R+\eta})$.

Développement limité de $F_0(\pi,\eta)$
Tout d'abord, on a : $\eta = \frac{s}{2}(R+\eta)H(\frac{\pi}{R+\eta}) + s\frac{\pi}{2} + o(\pi)$ avec $s = \frac{4}{5}$
$\eta = \frac{2}{5}(R+\eta)H(\frac{\pi}{R+\eta}) + \frac{2}{5}\pi + o(\pi)$
$\eta - \frac{2}{5}\pi = \frac{2}{5}(R+\eta)H(\frac{\pi}{R+\eta}) + o(\pi)$
$\frac{5}{2}\eta - \pi = (R+\eta)H(\frac{\pi}{R+\eta}) + o(\pi)$

De plus : $\eta = \frac{4}{5}(R+\eta)H(\frac{\pi}{2(R+\eta)}) \Leftrightarrow \frac{6}{5}(R+\eta)H(\frac{\pi}{2(R+\eta)}) = \frac{3}{2}\eta$.
On utilise cela pour calculer le DL de $F_0(\pi,\eta)$.

$F_0(\pi,\eta) =$
$H(\tau) - (1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta}) - (R+\eta)H(\frac{\pi}{R+\eta}) + \frac{6(R+\eta)}{5}H(\frac{\pi}{2(R+\eta)}) - \pi =$
$H(\tau) - (1-R)H(\frac{\tau}{1-R}) - (1-R)\epsilon\alpha + \eta H(\frac{\tau}{1-R}) - \frac{5}{2}\eta + \pi + \frac{3}{2}\eta - \pi + o(\pi)$
$H(\tau) - (1-R)H(\frac{\tau}{1-R}) - (1-R)\left(-\frac{\pi}{1-R} + \frac{\tau\eta}{(1-R)^2}\right)\alpha + \eta\left(H(\frac{\tau}{1-R}) - 1\right) + o(\pi)$
$H(\tau) - (1-R)H(\frac{\tau}{1-R}) + \pi\alpha + \eta\left(H(\frac{\tau}{1-R}) - 1 - \frac{\alpha\tau}{(1-R)}\right) + o(\pi)$
$H(\tau) - (1-R)H(\frac{\tau}{1-R}) + \pi\alpha + \frac{2}{5\ln(2)}\left(-\pi\ln\left(\frac{\pi}{R}\right) + \pi\left(1+\ln(2)\right)\right)\left(H(\frac{\tau}{1-R}) - 1 - \frac{\alpha\tau}{(1-R)}\right) + o(\pi)$
On pose $\upsilon := H(\frac{\tau}{1-R}) - 1 - \frac{\alpha\tau}{(1-R)}$
$H(\tau) - (1-R)H(\frac{\tau}{1-R}) + \pi\alpha + \frac{2}{5\ln(2)}\left(-\pi\ln\left(\frac{\pi}{R}\right) + \pi\left(1+\ln(2)\right)\right)\upsilon + o(\pi)$
$H(\tau) - (1-R)H(\frac{\tau}{1-R}) + \left(\alpha + \frac{2\upsilon}{5} + \frac{2\upsilon}{5\ln(2)}\right)\pi + \left(-\frac{2\upsilon}{5\ln(2)}\right)\pi\ln\left(\frac{\pi}{R}\right) + o(\pi)$
$H(\tau) - (1-R)H(\frac{\tau}{1-R}) + (\delta - d)\,\pi + d\pi\ln\left(\frac{\pi}{R}\right) + o(\pi)$

$$\boxed{F(\pi,\eta) = \textsc{Prange} + d\pi\ln(\frac{\pi}{R}) + (\delta-d)\,\pi + o(\pi)}$$

$$\text{Avec } d = -\frac{2\upsilon}{5\ln(2)} = \frac{2}{5}\frac{\frac{\alpha\tau}{(1-R)} - 1 - H(\frac{\tau}{1-R})}{\ln(2)}$$

$$\text{Et } \delta = \alpha + \frac{2\upsilon}{5} = \log_2\left(\frac{1-R-\tau}{\tau}\right) + \frac{2}{5}\left(\frac{\alpha\tau}{(1-R)} - 1 - H(\frac{\tau}{1-R})\right)$$

Essayons maintenant de trouver c et s tels que la complexité de $MMTQW$ s'écrit
avec la condition sur η :
$G(\pi,\eta) := \eta - s(R+\eta)H(\frac{\pi}{R+\eta})$

Nous connaissons déjà $s = s_0 = \frac{2}{5} + \epsilon$ avec $\epsilon = \frac{\pi}{5(R+\eta)H(\frac{\pi}{R+\eta})}$. Nous rappelons également que cela vient de l'égalité $\eta = \frac{4}{5}(R+\eta)H(\frac{\pi}{2(R+\eta)}) = \frac{2}{5}(R+\eta)H(\frac{\pi}{R+\eta}) + \frac{2}{5}\pi + o(\pi)$.
Il s'agit de trouver c. Pour ce faire, nous reprenons le développement limité de F : $F^*(\pi,\eta) =$
$F(\pi,\eta) + \frac{6(R+\eta)}{5}H(\frac{\pi}{2(R+\eta)}) - \pi =$
$H(\tau) - (1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta}) - (R+\eta)H(\frac{\pi}{R+\eta}) + \frac{6(R+\eta)}{5}H(\frac{\pi}{2(R+\eta)}) - \pi =$
$H(\tau) - (1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta}) - \left(\frac{5}{2}\eta - \pi\right) + \frac{3}{2}\eta - \pi =$
$H(\tau) - (1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta}) - \eta =$
$H(\tau) - (1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta}) - s_0(R+\eta)H(\frac{\pi}{R+\eta})$

Par conséquent $c = s_0 = \frac{2}{5} + \epsilon$

B.0.5 Algorithme MMT^*

On reprend les mêmes hypothèses que pour MMT, i.e. on suppose que : $\eta = \frac{s}{\ln(2)}\left(-\pi\ln\left(\frac{\pi}{R}\right) + \pi\right) + o(\pi)$ avec $0 \leq s$ un paramètre à optimiser.

Par rapport à MMT, il y a un paramètre supplémentaire ϵ, mais comme nous l'avons vu, afin d'exploiter au mieux cette technique, il est judicieux de prendre $\epsilon \approx \frac{\pi^2}{4R}$.

On considère encore $F(\pi,\eta) = H(\tau) - (1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta}) - (R+\eta)H(\frac{\pi}{R+\eta})$.
La stratégie à suivre est identique que pour MMT, avec pour différence que les trois fonctions sont cette fois-ci les suivantes :

— $F_1(\pi,\eta) = F(\pi,\eta) + \frac{R+\eta}{2} H\left(\frac{\frac{\pi}{2}+\epsilon}{R+\eta}\right)$

— $F_2(\pi,\eta) = F(\pi,\eta) + (R+\eta)H\left(\frac{\frac{\pi}{2}+\epsilon}{R+\eta}\right) - (R+\eta-\pi)\,H\left(\frac{\epsilon}{R+\eta-\pi}\right) - \pi$

— $F_3(\pi,\eta) = F(\pi,\eta) + 2(R+\eta)H\left(\frac{\frac{\pi}{2}+\epsilon}{R+\eta}\right) - (R+\eta-\pi)\,H\left(\frac{\epsilon}{R+\eta-\pi}\right) - \pi - \eta$

Pour analyser ces fonctions, remarquons tout d'abord que $\frac{\pi}{2}+\epsilon = \frac{\pi}{2} + o(\pi)$. On se sert de ce constat afin de calculer le développement limité de $\frac{R+\eta}{2} H\left(\frac{\frac{\pi}{2}+\epsilon}{R+\eta}\right)$.

$\frac{R+\eta}{2}\left(H\left(\frac{\frac{\pi}{2}+\epsilon}{R+\eta}\right)\right) =$
$\frac{R+\eta}{2}\left(H\left(\frac{\pi+o(\pi)}{2(R+\eta)}\right)\right)$
$\frac{R+\eta}{2\ln(2)}\left(-\frac{\pi+o(\pi)}{2(R+\eta)}\ln(\frac{\pi+o(\pi)}{2(R+\eta)}) + \frac{\pi+o(\pi)}{2(R+\eta)} + o(\pi)\right)$
$\frac{1}{2\ln(2)}\left(-\frac{\pi}{2}\ln(\frac{\pi}{2(R+\eta)}) + \frac{\pi}{2} + o(\pi)\right)$
$H\left(\frac{\pi}{2(R+\eta)}\right) + o(\pi)$

Par conséquent l'analyse de ce terme reste identique à l'ordre un au terme qui figure dans MMT.

Il y a un autre terme qui a été ajouté par rapport à MMT : $-(R+\eta-\pi)\,H\left(\frac{\epsilon}{R+\eta-\pi}\right)$.
Faisons donc le développement limité de ce terme afin d'évaluer son impact.

$-(R+\eta-\pi)\,H\left(\frac{\epsilon}{R+\eta-\pi}\right) =$
$-\frac{R+\eta-\pi}{\ln(2)}\left(-\frac{\epsilon}{R+\eta-\pi}\ln\left(\frac{\epsilon}{R+\eta-\pi}\right) + \frac{\epsilon}{R+\eta-\pi} + o(\epsilon)\right)$
$\frac{1}{\ln(2)}\left(\epsilon\left(\ln(\epsilon) - \ln(R+\eta-\pi)\right) - \epsilon + o(\epsilon)\right)$
$\frac{1}{\ln(2)}\left(\epsilon\left(\ln(\epsilon) - \ln(R) - \ln\left(1+\frac{\eta-\pi}{R}\right)\right) - \epsilon + o(\epsilon)\right)$
$\frac{1}{\ln(2)}\left(\epsilon\ln\left(\frac{\epsilon}{R}\right) - \epsilon\left(\frac{\eta-\pi}{R} + o(\eta+\pi)\right) - \epsilon + o(\epsilon)\right)$
$O(\epsilon)$
$o(\pi)$
Par conséquent, ce terme n'intervient pas à l'ordre 1 non plus.

On en déduit que l'avantage gagné par l'utilisation de la technique des représentations étendue est infime dans ce cas.

B.0.6 Algorithme $BJMMQW^*$

Il s'agit de l'expression simplfiée de l'algorithme $BJMMQW$ avec $\epsilon_1 = \epsilon_2 = 0$, où l'exposant est donné par la fonction
$F_0(\pi,\eta) = F(\pi,\eta) + \frac{64(R+\eta)}{27}H(\frac{\pi}{4(R+\eta)}) - \frac{10}{6}\pi$
où $F(\pi,\eta) = H(\tau) - (1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta}) - (R+\eta)H(\frac{\pi}{R+\eta})$.
La relation de dépendance entre les paramètres π et η est de la forme suivante :
$G(\pi,\eta) = 0$ avec $G(\pi,\eta) = \eta - s(R+\eta)H(\frac{\pi}{4(R+\eta)})$ où $s = 4/3$.

En reprenant les calculs déjà faits et en modifiant un peu les choses, on obtient :
$(R+\eta)H(\frac{\pi}{4(R+\eta)}) = \frac{1}{4\ln(2)}\left(-\pi\ln(\frac{\pi}{R}) + \pi\left(1+2\ln(2)\right) + \frac{\pi\eta}{R} + o(\pi+\eta)\right)$

Ainsi :

$$\begin{aligned} G(\pi,\eta) &= \eta - s(R+\eta)H(\frac{\pi}{4(R+\eta)}) \\ &= \eta - \frac{s}{4\ln(2)}\left(-\pi\ln(\frac{\pi}{R}) + \pi\left(1+2\ln(2)\right) + \frac{\pi\eta}{R} + o(\pi+\eta)\right) \\ &= \eta(1 - \frac{s}{4\ln(2)}\frac{\pi}{R}) - \frac{s}{4\ln(2)}\left(-\pi\ln(\frac{\pi}{R}) + \pi\left(1+2\ln(2)\right)\right) + o(\pi+\eta) \end{aligned}$$

D'où $\eta = \frac{s}{4\ln(2)}\frac{-\pi\ln(\frac{\pi}{R})+\pi(1+2\ln(2))+o(\pi+\eta)}{1-\frac{s}{4\ln(2)}\frac{\pi}{R}}$

On en déduit :

$$\begin{aligned} \eta &= \frac{s}{4\ln(2)}\left(-\pi\ln\left(\frac{\pi}{R}\right) + \pi\left(1+2\ln(2)\right) + o(\pi+\eta)\right)\left(1 + \frac{s\pi}{4\ln(2)R} + o(\pi+\eta)\right) \\ &= \frac{s}{4\ln(2)}\left(-\pi\ln\left(\frac{\pi}{R}\right) + \pi\left(1+2\ln(2)\right)\right) + o(\pi) \end{aligned}$$

Conclusion (en injectant $s = \frac{4}{3}$) :

$$\boxed{\eta = \frac{1}{3\ln(2)}\left(-\pi\ln\left(\frac{\pi}{R}\right) + \pi\left(1+2\ln(2)\right)\right) + o(\pi)}$$

De plus, on a l'égalité suivantes entre les développements limités de $(R+\eta)H(\frac{\pi}{4(R+\eta)})$ et de $(R+\eta)H(\frac{\pi}{R+\eta})$:

$$\begin{aligned} (R+\eta)H(\frac{\pi}{4(R+\eta)}) &= \frac{1}{4\ln(2)}\left(-\pi\ln(\frac{\pi}{R}) + \pi\left(1+2\ln(2)\right)\right) + o(\pi) \\ &= \frac{1}{4}\frac{1}{\ln(2)}\left(-\pi\ln(\frac{\pi}{R}) + \pi\right) + \frac{\pi}{2} + o(\pi) \\ &= \frac{1}{4}(R+\eta)H(\frac{\pi}{R+\eta}) + \frac{\pi}{2} + o(\pi) \end{aligned}$$

Alors, $\eta = s(R+\eta)H(\frac{\pi}{4(R+\eta)}) \Leftrightarrow \eta = \frac{s}{4}(R+\eta)H(\frac{\pi}{R+\eta}) + s\frac{\pi}{2} + o(\pi)$
Plus exactement, on peut écrire $\eta = s_0(R+\eta)H(\frac{\pi}{R+\eta})$ avec $s_0 = \frac{s}{4} + s\frac{\pi}{2(R+\eta)H(\frac{\pi}{R+\eta})}$.

Développement limité de $F_0(\pi,\eta)$
Pour faire ce calcul, on va utiliser les deux égalités suivantes :

1. $\eta = \frac{4}{3}(R+\eta)H(\frac{\pi}{4(R+\eta)}) \Leftrightarrow 2\eta = \frac{8}{3}(R+\eta)H(\frac{\pi}{4(R+\eta)})$.
2. $\eta = \frac{s}{4}(R+\eta)H(\frac{\pi}{R+\eta}) + s\frac{\pi}{2} + o(\pi)$ avec $s = \frac{4}{3}$
   $\eta = \frac{1}{3}(R+\eta)H(\frac{\pi}{R+\eta}) + \frac{2}{3}\pi + o(\pi)$
   $\eta - \frac{2}{3}\pi = \frac{1}{3}(R+\eta)H(\frac{\pi}{R+\eta}) + o(\pi)$
   $3\eta - 2\pi = (R+\eta)H(\frac{\pi}{R+\eta}) + o(\pi)$

$F_0(\pi,\eta) =$
$H(\tau) - (1-R-\eta)H(\frac{\tau-\pi}{1-R-\eta}) - (R+\eta)H(\frac{\pi}{R+\eta}) + \frac{8}{3}(R+\eta)H(\frac{\pi}{2(R+\eta)}) - 2\pi =$
$H(\tau) - (1-R)H(\frac{\tau}{1-R}) - (1-R)\epsilon\alpha + \eta H(\frac{\tau}{1-R}) - 3\eta + 2\pi + 2\eta - 2\pi + o(\pi)$
$H(\tau) - (1-R)H(\frac{\tau}{1-R}) - (1-R)\left(-\frac{\pi}{1-R} + \frac{\tau\eta}{(1-R)^2}\right)\alpha + \eta\left(H(\frac{\tau}{1-R}) - 1\right) + o(\pi)$
$H(\tau) - (1-R)H(\frac{\tau}{1-R}) + \alpha\pi + \eta\left(H(\frac{\tau}{1-R}) - 1 - \frac{\alpha\tau}{(1-R)}\right) + o(\pi)$
$H(\tau) - (1-R)H(\frac{\tau}{1-R}) + \alpha\pi + \frac{1}{3\ln(2)}\left(-\pi\ln\left(\frac{\pi}{R}\right) + \pi\left(1+2\ln(2)\right)\right)\left(H(\frac{\tau}{1-R}) - 1 - \frac{\alpha\tau}{(1-R)}\right) + o(\pi)$
On pose $\upsilon := H(\frac{\tau}{1-R}) - 1 - \frac{\alpha\tau}{(1-R)}$
$H(\tau) - (1-R)H(\frac{\tau}{1-R}) + \alpha\pi + \frac{1}{3\ln(2)}\left(-\pi\ln\left(\frac{\pi}{R}\right) + \pi\left(1+2\ln(2)\right)\right)\upsilon + o(\pi)$
$H(\tau) - (1-R)H(\frac{\tau}{1-R}) + \left(\alpha + \frac{2}{3}\upsilon + \frac{\upsilon}{3\ln(2)}\right)\pi + \left(-\frac{\upsilon}{3\ln(2)}\right)\pi\ln\left(\frac{\pi}{R}\right) + o(\pi)$
$H(\tau) - (1-R)H(\frac{\tau}{1-R}) + (\delta - d)\,\pi + d\pi\ln\left(\frac{\pi}{R}\right) + o(\pi)$

$$\boxed{F(\pi,\eta) = \textsc{Prange} + d\pi\ln(\frac{\pi}{R}) + (\delta-d)\,\pi + o(\pi)}$$

$$\text{Avec } d = -\frac{\upsilon}{3\ln(2)} = \frac{1}{3}\frac{\frac{\alpha\tau}{(1-R)} - 1 - H(\frac{\tau}{1-R})}{\ln(2)}$$

$$\text{Et } \delta = \alpha + \frac{2}{3}\upsilon = \log_2\left(\frac{1-R-\tau}{\tau}\right) + \frac{2}{3}\left(\frac{\alpha\tau}{(1-R)} - 1 - H(\frac{\tau}{1-R})\right)$$

B.0.7 Tableau récapitulatif des valeurs de c et s

Nous pourrons ajouter les résultats obtenus pour MMT et $MMTQW$ au tableau des valeurs de s et c.

Algorithme	s	c
D/SS	$1/2$	$= s$
DG	1	$1/3$
SSG	$1/2$	$1/4$
$SSQW$	$2/5$	$= s$
MMT	$0{,}54 < . < 0{,}59$	$= s$
$MMTQW$	$2/5 + \epsilon$	$= s$