Comment utiliser des algorithmes quantiques pour remplir son sac à dos et décoder des syndromes

Rémi Bricout

▶ **To cite this version:**

Rémi Bricout. Comment utiliser des algorithmes quantiques pour remplir son sac à dos et décoder des syndromes. Cryptographie et sécurité [cs.CR]. Sorbonne Université, 2021. Français. NNT : . tel-03536935

HAL Id: tel-03536935
https://inria.hal.science/tel-03536935

Submitted on 20 Jan 2022

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.

Sorbonne Université

École doctorale Informatique, Télécommunications et Électronique (Paris)

Inria de Paris / Équipe-projet COSMIQ

Comment utiliser des algorithmes quantiques pour remplir son sac à dos et décoder des syndromes

Thèse de doctorat d'informatique

présentée par

Rémi Bricout

dirigée par Anthony Leverrier et André Chailloux

soutenue publiquement le mardi 30 mars 2021

devant un jury composé de :

Anthony Leverrier	Inria	Directeur
André Chailloux	Inria	Directeur
Gilles Zémor	Institut de Mathématiques de Bordeaux	Rapporteur
Omar Fawzi	École Normale Supérieure de Lyon	Rapporteur
Magali Bardet	LITIS, Rouen	Examinatrice
Jean-Claude Bajard	Sorbonne Université	Examinateur
Jean-Pierre Tillich	Inria	Examinateur

Remerciements

Avant de commencer, je tiens à remercier tous ceux qui m'ont entouré durant ces quelques années.

Bien entendu, je pense particulièrement à Anthony Leverrier et à André Chailloux, qui m'ont accompagné/supporté durant ces trois ans (et même plus, en prenant en compte mon stage de M2).

Je remercie Gilles Zémor et Omar Fawzi, qui ont accepté de rapporter cette thèse. Je remercie également les membres du jury, Magali Bardet, Jean-Claude Bajard et Jean-Pierre Tillich.

Je pense aussi à tous mes collègues du bureau C211, qu'ils soient déjà partis ou qu'ils occupent encore les lieux. Vivien, Antoine, Nicky, Kaushik, Ghazal, Anirudh, Mariem, Simona, Lucien, Johanna, merci à vous.

Je remercie plus généralement tous les membres de l'équipe Secret, et de l'équipe Cosmiq qui lui a succédé. Que vous soyez encore dans cette équipe, Anne, Nicolas, Pascale, Gaëtan, María, Léo, Christelle, Christina, Ivan, Chstistophe, André, Daniel, Andréa, Rocco, Ferdinand, Valentin, Antonio, Clémence, Clara, Nicolas ; ou que vous soyez déjà partis vers d'autres horizons, Stisty, Yann, Sébastien, Virginie, Xavier, Thomas, Kévin, Shizhu, Matthieu ; à vous tous, merci !

Un grand merci à mes parents, à mes frères, à mes grands-parents, qui m'ont toujours encouragé (et m'ont toujours écouté poliment quand ils ne comprenaient pas grand chose à mon charabia).

Merci aussi à vous, les copains. Thomas, sans qui mon année de 5/2 aurait été bien triste, mais aussi Brice, Audrey, Marianne, Marina, Edwige, (Maxime, mérites-tu ta place dans ces remerciements ?) Un grand merci à Zénon et aux habitués de La Roque. Merci à Diane, à Raphaël et à Willy, pour les bons moments que j'ai passés avec vous.

Je m'excuse auprès de ceux que j'ai pu oublier. Du fond du cœur, merci !

Table des matières

Introduction

Cadre historique

La cryptologie est, par définition, la science des communications secrètes. Autrement dit, ce domaine étudie comment deux personnes peuvent communiquer de façon sûre, y compris en présence d'individus mal intentionnés (ou simplement trop curieux).

Historiquement, l'apparition des premiers chiffrements a été motivée par un besoin de *confidentialité*. Ainsi, de nombreux chiffrements anciens (chiffrements de César, de Vigenère, ...) ont été créés dans de but de protéger une information des yeux indésirables. Cette idée d'"écriture secrète" est l'origine étymologique de cryptographie.

Depuis, le champ de la cryptographie s'est élargi, et recouvre, en plus de la *confidentialité*, l'*authenticité* et l'*intégrité*. Là où l'objectif de la confidentialité est d'assurer que seul le destinataire légitime peut accéder au contenu d'un message, l'authenticité veut garantir l'identité de l'auteur (et donc prévenir une usurpation d'identité), et l'intégrité vérifie que le message n'a pas été altéré ou corrompu en cours de route.

L'utilisation systématique de la cryptographie a parallèlement motivé le développement de la cryptanalyse, c'est-à-dire l'étude d'attaques ou de méthodes générales pour lire des documents chiffrés. La science de la cryptologie est constituée du regroupement de la cryptographie (la conception et la certification de mécanismes de chiffrement) et de la cryptanalyse (l'attaque de ces chiffrements).

L'ère de la cryptologie moderne a commencé avec Claude Shannon et la théorie de l'information [Sha49]. La cryptologie est désormais fondée sur des bases mathématiques, et non plus sur des intuitions. Shannon définit alors deux niveaux de sécurité : la sécurité inconditionnelle et la sécurité calculatoire.

Un système cryptographique est inconditionnellement sûr s'il ne peut pas être attaqué par quelqu'un qui ne connaît pas la clef, et ce indépendamment

de la puissance de calcul dont il dispose. Il n'existe que peu de systèmes qui entrent dans cette catégorie, et ils sont bien trop inefficaces pour être utilisés couramment.

La notion de sécurité inconditionnelle étant malheureusement très peu pertinente en pratique, on utilise la notion, plus faible, de sécurité calculatoire. Un système cryptographique est sûr au sens calculatoire s'il est très difficile d'obtenir de l'information secrète pour un attaquant dont les ressources sont limitées. Typiquement, pour prouver qu'un système est sûr au sens calculatoire, on utilise une réduction, c'est-à-dire qu'on montre que si quelqu'un peut attaquer ce système, alors il est capable de résoudre un problème notoirement difficile. C'est par exemple le cas du célèbre système de chiffrement RSA [RSA78], dont la sécurité est au moins égale à la difficulté du problème de factorisation des entiers.

C'est dans ce cadre qu'intervient l'informatique quantique. Là où le problème de factorisation des entiers est actuellement difficile à résoudre (temps exponentiel en la taille du problème), l'algorithme de Shor [Sho94] montre qu'un ordinateur quantique pourrait résoudre ce problème facilement (en un temps polynomial). Ainsi, un problème difficile à résoudre avec une puissance de calcul restreinte pourrait devenir facile avec une puissance de calcul **quantique** restreinte.

Aujourd'hui, les conséquences concrètes de l'informatique quantique sont encore très incertaines. On ne sait pas si on pourra exécuter l'algorithme de Shor dans 5 ans, dans 150 ans, ou jamais. Cependant, face à cette menace potentielle se développe la cryptographie dite *post-quantique*, c'est-à-dire des systèmes cryptographiques qui résisteraient à l'avènement d'un ordinateur quantique.

Il existe plusieurs familles de systèmes cryptographiques post-quantiques. Les plus grandes sont basées sur l'utilisation des réseaux euclidiens, des codes correcteurs d'erreurs, des polynômes multivariés, des isogénies ou encore des fonctions de hachage. Le cas qui nous intéresse ici est celui des codes correcteurs.

Le remplissage de sac à dos

Avant de nous pencher sur les codes correcteurs d'erreurs, nous allons commencer par étudier le problème moins spécifique de remplissage de sac à dos. Ce problème, très général, se présente comme suit. On dispose d'un sac à dos d'une certaine capacité t, et d'un ensemble d'objets de poids x_1 à x_n. On veut trouver une façon de remplir le sac à dos, c'est-à-dire un sous-ensemble des objets dont le poids total est t.

Dans un premier temps, l'accélération de la recherche de solutions s'est faite par l'utilisation de collisions [HS74, SS81]. L'idée est de décomposer le problème en problèmes plus petits, de résoudre ces problèmes, puis de recombiner les morceaux.

La résolution du problème de sac à dos n'a alors pas connu de progrès notable pendant près de 30 ans, jusqu'à EUROCRYPT 2010. [HJ10] introduit alors la technique des représentations, un outil majeur pour résoudre plus efficacement le problème de remplissage de sac à dos, ainsi que d'autres problèmes similaires. Dans les grandes lignes, cette technique permet de se donner plusieurs façons de trouver la solution. On peut alors simplifier la recherche en imposant des contraintes arbitraires, tant qu'il reste au moins une façon de trouver la solution. Cette technique a ensuite été améliorée par [BCJ11].

Dans l'optique de la cryptographie post-quantique, la résolution du problème de remplissage de sac à dos par des ordinateurs quantiques a également été étudiée. Si l'algorithme de recherche de Grover [Gro96] est très efficace pour accélérer les recherches naïves, il ne s'applique pas directement aux recherches plus sophistiquées et qui utilisent beaucoup de mémoire, telles que celles employées dans les meilleurs algorithmes classiques de remplissage de sac à dos. L'utilisation des marches quantiques, une généralisation de la recherche de Grover, permet d'obtenir des accélérations quantiques importantes [BJLM13, HM18].

Le décodage de syndromes

Le décodage d'erreurs, et l'utilisation de codes correcteurs, proviennent de la théorie de l'information. Lors d'une communication, il n'est généralement pas possible de s'assurer qu'aucune erreur ne vient altérer l'information transmise, il y aura toujours du bruit. Il est alors nécessaire d'être capable de corriger ces erreurs.

Un excellente façon de faire est l'utilisation de codes correcteurs d'erreurs. Lors de la réception d'un message, le destinataire va calculer une quantité vectorielle appelée *syndrome*. Le problème de décodage consiste à retrouver l'erreur (le bruit qui s'est ajouté au message) à partir de ce syndrome.

Or il se trouve que, dans le cas général, le problème de décodage est difficile. Il a alors été détourné de son usage premier et utilisé à des fins cryptographiques [McE78]. Ce problème étant même supposé difficile à résoudre pour un ordinateur quantique, il est utilisé en cryptographie post-quantique.

Le problème du décodage est très proche du problème de remplissage de sac à dos, et les algorithmes qui les résolvent utilisent souvent des techniques similaires. Cependant, tandis que le remplissage de sac à dos se place dans $\mathbb{Z}$, le problème de décodage se situe dans un corps fini $\mathbb{F}_q$. Cela permet l'utilisation de techniques différentes, telles que l'élimination gaussienne. Les meilleurs algorithmes de décodage combinent ainsi des techniques de remplissage de sac à dos avec des techniques propres au problème de décodage.

Résultats

Remplissage de sac à dos

Dans une première partie, nous allons améliorer les algorithmes de remplissage de sac à dos. Cela concerne le cas classique comme le cas quantique. Dans le cas classique, nous étendrons les représentations utilisées par [BCJ11] : là où ils utilisaient des vecteurs dans $\{-1, 0, 1\}$, nous allons passer à $\{-1, 0, 1, 2\}$. Nous allons également montrer qu'une des contraintes qu'ils s'imposaient est en fait superflue.

Les algorithmes quantiques de remplissage de sac à dos combinent les algorithmes classiques avec des marches quantiques. Les améliorations des algorithmes classiques se transposent alors naturellement au cas quantique. Nous allons également détailler des améliorations spécifiques au cas quantique. La première consiste à utiliser l'algorithme de recherche de Grover pour accélérer certaines opérations. La seconde consiste à déséquilibrer une étape de l'algorithme, pour assouplir celui-ci et pouvoir mieux l'adapter au problème. Ces résultats ont été publiés dans [BBSS20].

La figure 1 récapitule les complexités de divers algorithmes de remplissage de sac à dos.

Algorithme	Complexité	Classique	Quantique
[SS81]	$\tilde{\mathcal{O}}\left(2^{n/2}\right)$	$\surd$	
[HJ10]	$\tilde{\mathcal{O}}\left(2^{0.337n}\right)$	$\surd$	
[BCJ11]	$\tilde{\mathcal{O}}\left(2^{0.291n}\right)$	$\surd$	
[BBSS20]	$\tilde{\mathcal{O}}\left(2^{0.283n}\right)$	$\surd$	
[BJLM13]	$\tilde{\mathcal{O}}\left(2^{0.241n}\right)$		$\surd$
[HM18]	$\tilde{\mathcal{O}}\left(2^{0.226n}\right)$		$\surd$
[BBSS20]	$\tilde{\mathcal{O}}\left(2^{0.216n}\right)$		$\surd$

FIGURE 1 – Complexité asymptotique de différents algorithmes de résolution du problème du sac à dos

Décodage ternaire

Le problème du décodage binaire, fondamental dans la théorie de l'information et dans les télécommunications, a été très soigneusement étudié. Ce n'est pas le cas du décodage ternaire, et encore moins si on cherche des erreurs de gros poids. Si certains papiers regardent le comportement du décodage dans des corps non binaires, le cas des erreurs de gros poids est systématiquement ignoré. Motivés par le schéma de signature Wave, dont la sécurité repose sur la difficulté du décodage ternaire avec des erreurs de gros

poids, nous allons essayer de trouver les meilleurs algorithmes possibles pour résoudre le décodage dans ce cas.

Notre étude du problème de décodage ternaire avec des erreurs de gros poids concernera deux types de paramètres. Dans la tradition des études du décodage, nous allons déterminer les paramètres pour lesquels le problème est le plus difficile pour notre algorithme. Nous allons également étudier la complexité du décodage dans des zones de paramètres pour lesquels il existe de nombreuses solutions, et en particulier pour les paramètres de Wave.

La cryptographie utilisant les codes correcteurs est un bon candidat pour créer des systèmes de chiffrement résistant aux ordinateurs quantiques. Nous analyserons donc également les performances des algorithmes quantiques pour résoudre le décodage ternaire à gros poids d'erreur.

La figure 2 résume les résultats obtenus lors de cette étude. La partie classique de ces résultats a été publiée dans [BCDL19].

	Classique	Quantique
Cas le plus difficile	$\tilde{\mathcal{O}}\left(2^{0.2362n}\right)$	$\tilde{\mathcal{O}}\left(2^{0.1368n}\right)$
Paramètres de Wave	$\tilde{\mathcal{O}}\left(2^{0.0155n}\right)$	$\tilde{\mathcal{O}}\left(2^{0.0097n}\right)$ [1]

FIGURE 2 – Complexité asymptotique du problème de décodage ternaire à gros poids, pour des algorithmes classiques et quantiques, pour différents paramètres

Le problème DOOM

Pour certains protocoles de cryptographie utilisant des codes correcteurs d'erreurs, la sécurité ne se réduit pas au simple problème de décodage, mais à un problème légèrement différent, appelé DOOM, pour *Decode One Out of Many*. Dans ce problème DOOM, il n'est plus demandé de retrouver l'erreur associée à un syndrome donné. À la place, il est demandé de choisir un syndrome dans une liste, et de donner l'erreur correspondant à ce syndrome.

Le problème DOOM est strictement plus facile à résoudre que le problème de décodage de syndrome usuel. C'est à ce problème que se réduisent divers schémas de signature basés sur des code correcteurs [CFS01, DST19]. Il n'a malheureusement été que très peu étudié.

Dans le cas classique, nous proposerons un algorithme dont la complexité est $\tilde{\mathcal{O}}\left(2^{0.0822n}\right)$ dans le pire cas, quand précédemment l'algorithme de Sen-

1. Cette complexité correspond aux performances d'un algorithme quantique sur les paramètres recommandés pour que Wave résiste à un attaquant classique. Si on voulait utiliser Wave en se prémunissant contre un attaquant quantique, on choisirait probablement des paramètres légèrement différents, en plus d'utiliser une clef plus longue.

drier [Sen11] donnait $\tilde{\mathcal{O}}\left(2^{0.0872n}\right)$. Pour le cas quantique, nous nous placerons dans un modèle fort, en nous autorisant un accès quantique aux syndromes. Nous pourrons alors obtenir une complexité $\tilde{\mathcal{O}}\left(2^{0.0481n}\right)$ dans le pire cas.

Chapitre 0

Préliminaires

0.1 Notations

0.1.1 Complexité

Nous allons travailler avec des problèmes difficiles, voire NP-complets. Par conséquent, tous les algorithmes étudiés auront une complexité en temps exponentielle en la taille n des entrées. Plus précisément, la complexité de ces algorithmes pourra, dans une large majorité des cas, ce mettre sous la forme $\mathcal{P}(n)2^{\alpha n}$, où $\mathcal{P}$ est un polynôme. L'étude de l'exposant de complexité asymptotique, α, sera déjà bien assez fastidieuse. Nous allons donc utiliser la notation $\tilde{\mathcal{O}}\left(2^{\alpha n}\right)$ qui sous-entend l'existence des éventuels facteurs polynomiaux. Ces polynômes, qui auraient évidemment leur importance dans une implémentation, seront ici entièrement ignorés.

En particulier, on pourra utiliser $\tilde{\mathcal{O}}(1)$ pour désigner une complexité polynomiale en n.

0.1.2 Binomiales, trinomiales, ... multinomiales

Souvent, nous aurons besoin de dénombrer des ensembles pour évaluer la complexité de certains algorithmes, ou encore pour vérifier leur correction. Le coefficient binomial $\binom{n}{k} := \frac{n!}{k!\,(n-k)!}$ est évidemment connu. Nous aurons également besoin des généralisations $\binom{n}{k,l} := \frac{n!}{k!\,l!\,(n-k-l)!}$ et $\binom{n}{k_1,k_2,\ldots k_i} := \frac{n!}{k_1!\,k_2!\,\ldots\,k_i!\,(n-k_1-\ldots-k_i)!}$.

0.1.3 Entropie

Avec l'utilisation de la notation $\tilde{\mathcal{O}}$, les quantités intéressantes sont les exposants, tandis que les facteurs polynomiaux sont sous-entendus. Les quantités importantes lors de dénombrements ne sont donc pas exactement les binomiales et autres multinomiales, mais seulement leurs logarithmes. Grâce à la formule de Stirling d'approximation des factorielles, nous disposons de l'équivalent usuel : $\binom{n}{k} = \tilde{\mathcal{O}}\left(2^{nh\left(\frac{k}{n}\right)}\right)$, où $h(x) := -x\log_2(x) - (1-x)\log_2(1-x)$ désigne l'entropie binaire. De même que pour les binomiales, nous aurons besoin de généraliser ces notations, et nous utiliserons :

$$\tilde{h}(x_1, \ldots, x_n) := -\sum_{i=1}^{n} x_i \log_2(x_i) - \left(1 - \sum_{i=1}^{n} x_i\right) \log_2\left(1 - \sum_{i=1}^{n} x_i\right)$$

Cette fonction n'a pas un nombre d'arguments fixe. Dans le cas où un seul argument lui est donné, on retrouve l'entropie binaire h.

Remarque. Par convention (et par continuité), on se permettra d'utiliser : $x \log_2(x) = 0$.

La fonction d'entropie $\tilde{h}$ permet d'utiliser l'approximation :

$$\binom{n}{k_1, \ldots, k_i} = \tilde{\mathcal{O}}\left(2^{n\tilde{h}\left(\frac{k_1}{n}, \ldots, \frac{k_i}{n}\right)}\right)$$

0.1.4 Distributions

Définition. *On note $D^n[\alpha]$ l'ensemble des vecteurs de taille n dont $\lfloor \alpha n \rfloor$ coordonnées sont des 1 et les autres sont des 0.*

Propriété 1. *L'espace $D^n[\alpha]$ est de taille $\binom{n}{\lfloor \alpha n \rfloor} = \tilde{\mathcal{O}}\left(2^{nh(\alpha)}\right)$*

Remarque. Par définition, le nombre de 1 dans un vecteur de $D^n[\alpha]$ est $\lfloor \alpha n \rfloor$. Cependant, on aurait pu choisir $\lceil \alpha n \rceil$, ou encore arrondir à l'entier le plus proche. Dans tous ces cas, on obtient que le cardinal de $D^n[\alpha]$ est $\tilde{\mathcal{O}}\left(2^{nh(\alpha)}\right)$. On se permettra un abus de notation en parlant d'un vecteur de densité α, ou d'un vecteur dont le nombre de 1 est αn, étant sous-entendu qu'on arrondit à l'entier inférieur. Le choix de la façon de se ramener à un entier n'a aucune incidence sur les exposants asymptotiques.

0.1.5 Fusion de listes et contrainte modulaire

La fusion de listes est une sous-routine qui sera abondamment utilisée dans toute la suite, que ce soit pour construire des algorithmes de remplissage de sac à dos ou des algorithmes de décodage.

Dans le cas du remplissage de sac à dos, on part de deux listes de vecteurs d'entiers $L_1 = \{x_1, \ldots, x_m\}$ et $L_2 = \{y_1, \ldots, y_n\}$. On se donne également une fonction linéaire $\mathcal{L}$ à image dans les entiers, une condition de modulo t modulo M, ainsi qu'une distribution cible D. On veut récupérer les sommes $x_i + y_j$ qui vérifient :

$$\begin{cases} \mathcal{L}(x_i + y_j) \equiv t \ [M] \\ x_i + y_j \in D \end{cases}$$

Pour réaliser cette opération, on s'y prend de la façon suivante :

Algorithm 1: Fusion de listes

Data: $L_1 = \{x_1, \ldots, x_m\}$, $L_2 = \{y_1, \ldots, y_n\}$, t, M, D
$L_{aux} := NULL$
$L_{res} := NULL$
for $x_i \in L_1$ **do**
  $L_{aux} := L_{aux}, (\mathcal{L}(x_i) \mod M, x_i)$
end
sort L_{aux} selon la première coordonnée
for $y_j \in L_2$ **do**
  search $(t - \mathcal{L}(y_j) \mod M, x_i)$ in L_{aux}
  For each $(\mathcal{L}(x_i) \mod M, x_i)$ *found* **do**
    if $x_i + y_j \in D$ **then**
      $L_{res} := L_{res}, (x_i + y_j)$
    end
  end
end
return L_{res}

De cette façon, on énumère facilement tous les éléments de la forme $x_i + y_j$ qui vérifient $\mathcal{L}(x_i + y_j) \equiv t \ [M]$. Si les $\mathcal{L}(x_i)$ et les $\mathcal{L}(y_j)$ sont uniformément distribués modulo M, il y a en moyenne $\frac{|L_1| \times |L_2|}{M}$ tels éléments. On ne retient alors que ceux qui vérifient également la seconde contrainte.

Puisqu'il faut également lire les listes L_1 et L_2, le coût de cette opération est alors $\tilde{\mathcal{O}}\left(\max\left(|L_1|, |L_2|, \frac{|L_1| \times |L_2|}{M}\right)\right)$, indépendamment de la contrainte $x_i + y_j \in D$.

Le cas du décodage présente quelques différences. Les x_i et les y_j n'appartiennent plus à $\mathbb{Z}^n$, mais à $\mathbb{F}_q^n$, et l'image de la fonction $\mathcal{L}$ n'est plus dans $\mathbb{Z}$, mais dans $\mathbb{F}_q^m$. On se permettra un abus de langage pour les contraintes de modulo : $\mathcal{L}(x_i) \equiv t \ [q^c]$ signifie que les c dernières coordonnées de $\mathcal{L}(x_i)$ et de t sont identiques. Tout le reste (sous-routine de fusion, complexité) fonctionne exactement de la même façon qu'avec $\mathbb{Z}$.

Heuristique 1. *Quand on fusionne deux listes $L_1 \subset D_1$ et $L_2 \subset D_2$, si les vecteurs des listes à fusionner sont uniformément distribués et ne contiennent*

pas de doublons, alors on peut considérer que les vecteurs qu'on produit sont uniformément distribués dans $D \cap \{x|\mathcal{L}(x) \equiv t\ [M]\}$.

Remarque 0.1. Les algorithmes proposés par [HJ10] et [BCJ11] utilisent déjà cette heuristique. Ces algorithmes ont été implémentés et les résultats expérimentaux montrent que cela est parfaitement raisonnable en pratique.

0.2 Bases du calcul quantique

La mécanique quantique est apparue au début de XX^{e} siècle pour résoudre plusieurs problèmes en physique fondamentale, insolubles jusqu'alors. La loi de Planck pour le rayonnement du corps noir, obtenue en 1900, décrit parfaitement les mesures expérimentales. Cependant, les théories physiques étaient alors incapables de justifier cette loi : cela se comporte "comme si" l'énergie échangé entre les atomes et les rayonnements était une quantité discrète (un "quanta"). Les développements de la physique quantique ont conduit à réfuter certains principes de la physique classique, qui apparaissaient pourtant comme des principes de bon sens évident. Les exemples les plus notoires sont les suivants :

La quantification. À une très petite échelle, les échanges d'énergie ne peuvent prendre que des valeurs discrètes. En 1905, Einstein explique l'effet photo-électrique en supposant que l'énergie transportée pas la lumière est elle-même quantifiée (le terme "photon" n'apparaîtra que 20 ans plus tard). En 1913, Bohr propose un modèle atomique expliquant les observations des spectres d'absorption : le moment cinétique des électrons est quantifié et ne peut prendre que certaines valeurs. Autrement dit, les électrons d'un atome ne peuvent être que dans certains *états* (orbitales) bien particuliers.

Les superpositions d'états. En 1925 est découverte l'équation de Schrödinger, pour décrire l'évolution d'une particule. Cette équation est linéaire. Par conséquent, si elle admet plusieurs solutions, toutes les combinaisons linéaires des ces solutions sont alors également des solutions. Si une particule admet plusieurs états stables, les combinaisons linéaires, appelées superpositions, de ces états sont également stables. L'exemple le plus connu est l'expérience de pensée de Schrödinger, dans laquelle il décrit un chat dans une superposition de l'état "vivant" et de l'état "mort". Les superpositions d'états quantiques ne sont pas uniquement de solutions théoriques de l'équation de Schrödinger : des états en superposition ont été mis en évidence expérimentalement.

L'intrication. Comme indiqué dans le point précédent, une particule peut être dans une superposition d'état. Il y a pire encore : un système de plusieurs

particules peut être dans une superposition d'états, telle qu'il est impossible de décrire chacune de ses particules indépendamment les unes de autres. L'exemple le plus simple est la paire de Bell : un système de deux particules qui se trouve dans la superposition de l'état $|AA\rangle$ (les deux particules sont dans l'état A) et $|BB\rangle$. Mesurer n'importe laquelle de ces deux particules donne alors immédiatement l'information sur l'état de l'autre particule.

Caractère destructif des mesures. Observer un système quantique peut perturber celui-ci. En particulier, imaginons qu'un système quantique soit dans une superposition de deux état quantiques (appelons ces états $|A\rangle$ et $|B\rangle$), et qu'on mesure cette particule. Si la mesure nous affirme que la particule est dans l'état $|A\rangle$, alors la particule sera, à partir du moment de la mesure, dans l'état $|A\rangle$. Le comportement est similaire si le résultat de la mesure est $|B\rangle$. Le simple fait d'observer la particule a provoqué l'effondrement de la superposition. Bien évidemment, il a été vérifié expérimentalement que la particule était auparavant dans une superposition, et qu'elle n'était pas déjà dans l'un des états dégénérés $|A\rangle$ ou $|B\rangle$.

Ce qui nous intéresse particulièrement ici est le principe de superposition. L'idée, plutôt alléchante, est la suivante. Si on doit réaliser un même calcul sur de nombreuses données, on commence par créer une superposition de ces données, puis on effectue ledit calcul sur cette superposition. Ainsi, on peut faire tous les calculs simultanément.

Malheureusement, le caractère destructif des mesures nous empêche d'exploiter les superpositions aussi facilement : il est impossible, à partir de la superposition des résultats, de retrouver les résultats de chacun des calculs. En effet, les mesures détruisent les superpositions d'états quantiques. Si on mesure naïvement la superposition de tous les résultats, cette superposition va s'effondrer et on ne va obtenir que l'un des résultats.

En particulier, une limite absolue est la suivante. Considérons un système de n qubits en superposition. Quels que soient la sophistication de ce système, le nombre d'états formant la superposition et le nombre de variables nécessaires à la description de cet état, la mesure de ce système ne pourra jamais apporter plus de n bits d'information ([Hol73]).

Notons au passage qu'il n'est pas possible de tricher en copiant préventivement le système avant de le mesurer : il n'est pas possible en général de créer une copie d'un qubit ([WZ82, Die82]).

Toute la subtilité des algorithmes quantiques consiste donc à réaliser de nombreux calculs en superposition, puis à combiner les résultats de ces calculs avant de mesurer le système. Ainsi, lors de la mesure, nous obtenons l'information qui nous est vraiment utile, tandis que les valeurs et résultats intermédiaires sont détruits à jamais.

Différentes technologies sont envisageables pour créer physiquement des

qubits (jonction Josephson, cavités résonnantes, ions piégés, ...), et pour les manipuler. Nous allons faire abstraction de tout cela et utiliser un modèle mathématique des qubits et des opérations autorisées. Ce modèle mathématique veut décrire ce qu'on pense possible de réaliser comme ensembles de qubits et comme manipulations sur ces systèmes (en oubliant les difficultés techniques). Nous n'allons en faire qu'un survol rapide afin d'introduire les outils dont nous allons avoir besoin. Plus de détails peuvent être trouvés dans [NC00].

0.2.1 Des bits aux qubits

L'unité élémentaire d'information classique est le bit, qui peut prendre la valeur 0 ou la valeur 1. Le qubit (quantum bit) en est une généralisation. De même qu'un bit, un qubit peut prendre la valeur 0 (on dit alors qu'il est dans l'état $|0\rangle$, en utilisant la notation *ket* de Dirac), ou encore la valeur 1 (état $|1\rangle$). Cependant, selon le principe de superposition, il est également possible pour un qubit d'être dans une combinaison linéaire des états $|0\rangle$ et $|1\rangle$. Le forme générale d'un qubit est donc $\alpha_0|0\rangle + \alpha_1|1\rangle$, où α_0 et α_1 sont des coefficients complexes. On se restreindra aux états normalisés, c'est-à-dire tels que $|\alpha_0|^2 + |\alpha_1|^2 = 1$.

Si on mesure un qubit dans la base canonique $(|0\rangle, |1\rangle)$, il y a deux possibilités :

— Soit le résultat de la mesure est $|0\rangle$. Cela arrive avec probabilité $|\alpha_0|^2$. Le qubit s'effondre alors dans l'état $|0\rangle$.

— Soit le résultat de la mesure est $|1\rangle$. Cela arrive avec probabilité $|\alpha_1|^2$. Le qubit s'effondre alors dans l'état $|1\rangle$.

De façon générale, pour un ensemble de n bits, il y a classiquement 2^n valeurs possibles, qui correspondent aux entiers de 0 à $2^n - 1$. Pour un ensemble de n qubits, ces 2^n valeurs sont autant d'états possibles. On notera ces états $|i\rangle$, pour $i \in [\![0, 2^n - 1]\!]$. Comme dans le cas d'un seul qubit, on peut faire toutes les combinaison linéaires de ces états. La forme générale d'un système de n qubits est donc : $\sum_{i=0}^{2^n-1} \alpha_i|i\rangle$, avec $\sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1$. L'état d'un système de n qubits est donc décrit par un vecteur normalisé de l'espace de Hilbert $\mathbb{C}^{2^n}$, là où il suffisait d'un simple entier pour décrire n bits.

0.2.2 Manipulation des qubits

L'évolution d'un système quantique est gouvernée par l'équation de Schrödinger, qui relie l'évolution à une quantité appelée Hamiltonien. S'il est possible de modifier ce Hamiltonien pour contrôler l'évolution du système, il est

impossible de s'affranchir de certaines contraintes. Plus précisément, l'équation de Schrödinger nous limite aux opérations qui sont des transformations unitaires de l'espace de Hilbert $\mathbb{C}^{2^n}$.

D'un autre côté, on s'autorisera toutes les transformations unitaires. On ne sait pas s'il sera possible techniquement de réaliser parfaitement toutes ces opérations. Cela dit, on supposera qu'on sera capable de construire un système de portes universel, c'est-à-dire un ensemble de portes permettant d'approcher n'importe quelle opérateur. Le théorème de Solovay-Kitaev ([Kit97]) permet de minorer l'efficacité d'une telle approximation.

Puisque les transformations des systèmes de qubits sont des transformations unitaires, on les assimilera souvent à leur matrice dans la base canonique $(|0\rangle, |1\rangle, ...)$.

Par ailleurs, on peut remarquer que les opérations unitaires (qui sont les seules opérations autorisées), sont inversibles. De fait, toute matrice unitaire U admet un inverse (également unitaire), noté $U^\dagger$, est qui correspond à U qu'on a transposé et dont on a pris le conjugué complexe ($U^\dagger = \overline{U^T} = \overline{U}^T$). Par conséquent, on peut annuler toute opération réalisée sur des qubits, tant qu'on n'a pas effectué de mesure.

0.2.3 Opérations notables

Porte logique Not

Pour des circuits classiques, la porte logique Not est un opérateur qui agit sur un bit, et qui transforme les 0 en 1 et inversement.

Pour obtenir un équivalent quantique, on voudrait un opérateur qui envoie l'état $|0\rangle$ sur $|1\rangle$ et inversement. Il existe exactement une transformation unitaire qui vérifie cette condition : elle est caractérisée par le fait que l'image de la base $(|0\rangle, |1\rangle)$ est la base $(|1\rangle, |0\rangle)$. Cet opérateur sera souvent noté $X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, en référence à l'opérateur σ_x de Pauli.

Portes logiques And et Or

Les opérations autorisées sur les qubits sont exclusivement constituées de transformations unitaires, donc inversibles. Les portes classiques And et Or, qui prennent deux bits en entrée et n'en renvoient qu'un, ne vérifient pas cette contrainte. Si on veut pouvoir défaire un calcul comportant une porte And, il est nécessaire d'avoir conservé les entrées de cette porte.

L'équivalent quantique de la porte And sera donc une porte sur trois qubits. Les deux premiers qubits seront les entrées et ne seront pas modifiés par la porte. Le dernier qubit servira pour écrire le résultat de l'opération. Cette porte peut être décrite par l'image de la base canonique : le vecteur $|a\rangle|b\rangle|c\rangle$ est envoyé sur $|a\rangle|b\rangle|c \oplus (a \wedge b)\rangle$. Cette porte est connue sous le nom de porte de Toffoli [Tof80].

De façon similaire, on peut définir un équivalent de la porte Or par $|a\rangle|b\rangle|c\rangle \mapsto |a\rangle|b\rangle|c \oplus (a \vee b)\rangle$.

Dans la base canonique, ces transformations sont représentées par les matrices :

$$\begin{pmatrix} 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \end{pmatrix} \quad \text{et} \quad \begin{pmatrix} 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \end{pmatrix}$$

Fonctions booléennes quelconques

De même que les portes And et Or, la plupart des fonctions booléennes ne sont pas réversibles. Cependant, nous savons comment faire des portes Not, et comment simuler les portes And et Or en ajoutant des qubits auxiliaires. Par ailleurs, toute fonction booléenne peut être décomposée à l'aide de ces trois portes. Ainsi, si on ne peut pas transposer directement une fonction booléenne f quelconque, il est néanmoins possible de construire un équivalent quantique $\mathcal{O}_f$ réalisant l'opération :

$$\forall i, \mathcal{O}_f : |i\rangle|x\rangle \mapsto |i\rangle|x \oplus f(i)\rangle$$

Cela ne décrit $\mathcal{O}_f$ que sur une base, sa linéarité nous permettant de l'étendre aux superpositions :

$$\mathcal{O}_f : \sum_i \alpha_i |i\rangle|x\rangle \mapsto \sum_i \alpha_i |i\rangle|x \oplus f(i)\rangle$$

Ainsi, pour toute fonction booléenne classique f, on peut définir un opérateur quantique $\mathcal{O}_f$ qui simule cette fonction tout en conservant les superpositions. De plus, si on peut implémenter f avec un circuit classique comportant N portes logiques, on peut en déduire un circuit quantique pour $\mathcal{O}_f$ comportant $\mathcal{O}(N)$ portes (voir par exemple [NC00]).

0.2.4 Algorithmes de recherche quantique

Nous allons ici décrire l'algorithme de recherche quantique de Grover ([Gro96]). Dès que nous voulons rechercher un élément vérifiant une propriété particulière dans une liste, il est très probable que l'algorithme de Grover permette d'obtenir un gain significatif par rapport à un algorithme classique. Sa polyvalence fait qu'il est très largement utilisé.

Problème de recherche

Le problème de recherche est le suivant : parmi un ensemble d'éléments, on veut en trouver un qui vérifie une certaine propriété. Autrement dit, si on se donne une fonction de test (un fonction qui revoie 1 si et seulement si son entrée vérifie la propriété), on cherche une pré-image de 1 par cette fonction de test.

Problème : Recherche

Entrée :	$f : [\![0, N-1]\!] \to \{0,1\}$	fonction de test
Sortie :	$i \in [\![0, N-1]\!]$ tel que $f(i) = 1$	s'il existe une solution
	"impossible"	sinon

Algorithme de Grover

On note M le nombre de pré-images de 1 par f. L'algorithme de Grover est composé de l'application répétée de deux opérateurs :

— L'appel à l'oracle : $S_{Bad} : |i\rangle \mapsto (-1)^{f(i)}|i\rangle$. Pour obtenir cet opérateur, on construit à l'aide de f un oracle $\mathcal{O}_f : |i\rangle|x\rangle \mapsto |i\rangle|x \oplus f(i)\rangle$ et on l'applique en choisissant $|x\rangle = |-\rangle := (|1\rangle - |0\rangle)/\sqrt{2}$.
De cette façon, le résultat de l'appel à $\mathcal{O}_f$ est $|i\rangle|-\rangle \mapsto |i\rangle(|1 - f(i)\rangle - |f(i)\rangle)/\sqrt{2} = (-1)^{f(i)}|i\rangle|-\rangle$. On oublie alors le qubit auxiliaire dont la valeur est toujours $|-\rangle$.

— L'opérateur de symétrie par rapport à $|\phi_0\rangle := \dfrac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle$, noté $S_{|\phi_0\rangle}$, et qui est indépendant de f.

On considère maintenant les deux états de superpositions suivants :

— Le vecteur $|Good\rangle := \dfrac{1}{\sqrt{M}} \sum_{f(i)=1} |i\rangle$. Si on mesure ce vecteur, on obtient l'un des i vérifiant $f(i) = 1$. Le but est donc de construire $|Good\rangle$ (où du moins un vecteur très proche), puis de le mesurer.

— Le vecteur $|Bad\rangle := \dfrac{1}{\sqrt{N-M}} \sum_{f(i)=0} |i\rangle$.

Ces deux vecteurs sont normés et orthogonaux. Par ailleurs, en notant θ l'unique élément de $[0, 2\pi[$ tel que $\cos\theta = \sqrt{\dfrac{N-M}{N}}$ et $\sin\theta = \sqrt{\dfrac{M}{N}}$, on remarque que $|\phi_0\rangle = \cos\theta|Bad\rangle + \sin\theta|Good\rangle$. Enfin, on constate que le plan $(|Bad\rangle, |Good\rangle)$ est stable par les opérateurs S_{Bad} et $S_{|\phi_0\rangle}$. Les représenta-

tions matricielles de ces opérateurs dans la base $(|Bad\rangle, |Good\rangle)$ sont :

$$S_{Bad} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \quad \text{et} \quad S_{|\phi_0\rangle} = \begin{pmatrix} \cos 2\theta & \sin 2\theta \\ \sin 2\theta & -\cos 2\theta \end{pmatrix}$$

La composition de ces opérateurs donne :

$$S_{|\phi_0\rangle} S_{Bad} = \begin{pmatrix} \cos 2\theta & -\sin 2\theta \\ \sin 2\theta & \cos 2\theta \end{pmatrix} = R_{2\theta}$$

On reconnaît la rotation d'angle 2θ. En partant de $|\phi_0\rangle$, puis en appliquant $S_{|\phi_0\rangle} S_{Bad}$ un nombre de fois proche de $\frac{\pi}{4}\sqrt{\frac{N}{M}}$, on obtient un vecteur très proche de $|Good\rangle$, qu'il suffit alors de mesurer.

Amplification d'amplitude

L'amplification d'amplitude ([BHMT02]) est une généralisation très utile de l'algorithme de Grover. Là où l'algorithme de Grover cherche un entier dans $i \in [\![0, N-1]\!]$, l'amplification d'amplitude permet de trouver un élément parmi une liste produite par un circuit $\mathcal{A}$. Plus formellement :

Lemme 0.1. *Soit f une fonction de test, et $\mathcal{O}_f$ un circuit quantique correspondant. Soit $\mathcal{A}$ un circuit quantique. On note p la probabilité que $\mathcal{A}$ produise un bon élément (*i.e. *on obtient* 1 *avec probabilité p quand on mesure le résultat de $\mathcal{O}_f\mathcal{A}|0\rangle$). L'algorithme d'amplification d'amplitude trouve un bon élément avec forte probabilité, et demande $\mathcal{O}(1/\sqrt{p})$ appels à $\mathcal{A}$, $\mathcal{A}^\dagger$ et $\mathcal{O}_f$.*

0.2.5 Marches quantiques

Les marches quantiques ([Sze04]) sont une sorte de généralisation de l'amplification d'amplitude, dans le cas de données structurées. Nous allons ici utiliser les marches quantiques dans le formalisme de Magniez, Nayak, Roland et Santha ([MNRS11]). Le but n'est pas de redémontrer des résultats sur les marches quantiques, ni de les améliorer. Nous allons donc regarder comment fonctionnent les marches quantiques dans les grandes lignes, puis les utiliser en boite noire.

Considérons dans un premier temps une marche aléatoire (classique). Soit $G = (V, E)$ un graphe connexe, non orienté, régulier et apériodique (par exemple, les graphes de Johnson qu'on utilisera par la suite). Dans ce graphe, une fraction ε des sommets sont marqués, et l'objectif est de trouver l'un de ces sommets marqués. Une marche aléatoire se déroule de la façon suivante :

1. On calcule un sommet de départ (aléatoirement ou non, cela n'a pas d'importance.) Cette étape sera appelée Setup, et requiert un temps S.

2. On se déplace aléatoirement dans le graphe, en choisissant un sommet. Cela correspond à une étape d'un processus de Markov. Un tel déplacement sur le graphe est une UPDATE, et prend un temps U. En notant δ le trou spectral de la marche aléatoire associée, et en répétant cette étape $1/\delta$ fois, on se retrouve très proche de la distribution asymptotique, qui se trouve être le distribution uniforme (ce qui confirme au passage que le choix du sommet de départ n'est pas important.)
3. On regarde si le sommet actuel est marqué ou non. Ce test, ou CHECK, prend un temps C. Si le sommet est marqué, on a fini. S'il n'est pas marqué, on retourne au sommet de départ, et on reprend à l'étape 2.

L'étape 2 consiste en $1/\delta$ applications de UPDATE. L'étape 3 consiste en une application de CHECK. Or ces deux étapes sont répétées jusqu'à trouver un sommet marqué, soit en moyenne $1/\varepsilon$ fois. Par ailleurs, l'étape 1, à savoir SETUP, est exécutée exactement une fois au début. La complexité globale de la marche aléatoire (classique) est donc $\mathsf{S} + \frac{1}{\varepsilon}\left(\frac{1}{\delta}\mathsf{U} + \mathsf{C}\right)$.

Dans le cas d'une marche quantique, la complexité est :

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

Remarque 0.2. On se servira de cette formule en boite noire. On se contentera de décrire les fonctions de SETUP, UPDATE et CHECK comme si on faisait une marche aléatoire. Pour trouver des détails sur la façon de construire un algorithme quantique à partir de cela, consulter [MNRS11].

Nous utiliserons les marches quantiques uniquement sur des produits cartésiens de graphes de Johnson. Nous rappelons ici un résultat sur le trou spectral δ d'un tel graphe :

Définition 0.1 (Graphe de Johnson)**.** *Le graphe de Johnson de paramètres n et k, qu'on notera $J(n,k)$, est un graphe régulier non-orienté. Ses sommets sont les sous-ensembles de cardinal k d'un ensemble à n éléments. Deux sommets (*i.e. *deux sous-ensembles) s_1 et s_2 sont reliés par une arête si et seulement s'ils ne diffèrent que d'un élément, c'est-à-dire si et seulement si $|s_1 \bigcap s_2| = k-1$. Il est connu que son trou spectral est $\delta(J(n,k)) = \frac{n}{k(n-k)}$.*

Théorème 0.2 (Produit cartésien de graphes de Johnson, [KT17])**.** *En notant $J^m(n,k)$ le produit cartésien de m copies de $J(n,k)$, on peut minorer le trou spectral de $J^m(n,k)$ par $\delta(J^m(n,k)) \geqslant \frac{1}{m}\delta(J(n,k))$.*

Première partie

Le remplissage de sac à dos

Chapitre 1

Introduction au remplissage de sac à dos

1.1 Présentation du problème

Le problème du remplissage de sac à dos (en anglais *knapsack problem*, ou encore *subset-sum problem*) est bien connu. C'est en effet un bon exemple de problème NP-complet très facile à énoncer.

Puisque ce problème est difficile, il peut être tentant de s'en servir pour construire des schémas cryptographiques ([MH78, LPS10]). Cela a naturellement motivé la recherche d'algorithmes de plus en plus efficaces.

1.1.1 Définition

Le problème du remplissage de sac à dos, tel que nous allons le traiter, est le suivant. On se place dans le groupe $\mathbb{Z}$ des entiers relatifs. On reçoit en entrée un ensemble fini $\mathcal{X}$ d'entiers positifs, ainsi qu'une cible t, elle aussi un entier positif. Le but est de trouver un sous-ensemble $\mathcal{E}$ de $\mathcal{X}$ dont la somme des éléments est t si le problème admet une solution, ou d'indiquer que le problème n'admet pas de solution dans le cas contraire. Plus formellement :

Problème : Sac à dos (version ensembliste)

Entrée :	$\mathcal{X} = \{x_1, x_2, \ldots, x_n\}, x_i \in \mathbb{N}$	
	$t \in \mathbb{N}$	
Sortie :	$\mathcal{E} \subset \mathcal{X}$ tel que $\sum_{x \in \mathcal{E}} x = t$	s'il existe une solution
	"impossible"	sinon

1.1.2 Formalisme matriciel

Si la définition précédente est plus naturelle et plus proche de l'intuition du problème du sac à dos, on préférera utiliser la définition suivante (parfaitement équivalente) du problème : on ne revoie plus la solution sous la forme d'un ensemble, mais sous la forme d'un vecteur contenant des 0 et des 1.

Problème : Sac à dos (version matricielle)
Entrée : $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{N}^n$
$t \in \mathbb{N}$

Sortie : $\mathbf{e} \in \{0,1\}^n$ tel que $\sum_{i=1}^{n} \mathbf{e}_i x_i = t$ s'il existe une solution
"impossible" sinon

Le vecteur $\mathbf{e}$ renvoyé par un algorithme de résolution est constitué exclusivement de 0 et de 1. Cependant, rien n'interdit (au contraire, ce sera même utile) à un tel algorithme d'utiliser temporairement d'autres valeurs dans $\mathbf{e}$. C'est pour cette raison qu'on préférera le formalisme matriciel : il est plus facile de comprendre ce qu'on fait en manipulant un vecteur contenant des -1, plutôt que de manipuler un ensemble contenant -1 fois un élément.

1.1.3 Quelques variantes

Il existe de nombreuses variantes au problème du sac à dos. Par exemple, la variante décisionnelle du problème : on doit seulement indiquer si l'instance admet ou non une solution, sans devoir la calculer dans le cas positif. Il a été montré que cette variante constitue un problème NP-complet [GJ79].

Il existe également des problèmes de maximisation. Dans ces cas, au lieu de recevoir un ensemble $\mathcal{X}$ d'entiers, on reçoit un ensemble de couples (p, v) (c'est-à-dire un ensemble d'objets, chaque objet ayant un certain poids et une certaine valeur). Le sac à dos ne peut pas contenir plus qu'un poids t. On veut alors remplir le sac en maximisant la valeur du contenu, mais sans dépasser la limite de poids. Ces cas ne seront pas étudiés ici.

1.1.4 Génération des instances

Nous n'allons pas étudier la complexité du problème de remplissage de sac à dos dans le pire cas, mais en moyenne.

Pour créer des instances du problème de remplissage de sac à dos, nous choisissons deux paramètres : $n \in \mathbb{N}$, et $m \in \mathbb{R}^+$. Les entrées $x_1, \ldots, x_n$ sont alors tirés aléatoirement dans $[\![0, 2^m]\!]$, tandis que t est tiré aléatoirement dans $[\![0, n2^m]\!]$.

Nous nous intéressons à la complexité du problème de remplissage de sac à dos en fixant n et m, et en moyennant sur les x_i et sur t.

1.1.5 Densité

Nous définissons la densité d'une instance du problème du sac à dos comme :

$$d := \frac{n}{m}$$

Intuitivement, la densité d'une instance du problème de sac à dos correspond aux chances pour que cette instance possède une solution : si la densité est très petite, il est peu probable que l'instance admette une solution. Au contraire, si la densité est très grande, l'instance admet probablement beaucoup de solutions. Plus précisément, nous avons :

Proposition 1.1. *Une instance du problème de remplissage de sac à dos de densité d admet en moyenne $\tilde{\mathcal{O}}\left(2^{(1-1/d)n}\right)$ solutions.*

Démonstration. Avec n entiers $(x_1, \ldots, x_n)$ on peut créer 2^n sommes, ces sommes étant toutes comprises entre 0 et $n2^m$. Un poids cible t choisi aléatoirement entre 0 et $n2^m$ aura donc en moyenne $\frac{1}{n}2^{n-m}$ solutions. □

1.1.6 Cas difficile

Si d est petit, on peut utiliser un oracle qui résout le problème de plus court vecteur dans des réseaux (Shortest Vector Problem) comme cela a été montré dans [LO85] pour $d < 0.64$, puis dans [CJL+92] pour $d < 0.94$. Pour le moment, cette réduction ne semble pas être applicable pour une densité d très proche de 1.

D'autre part, pour d plus grand que 1, le problème du sac à dos a de fortes chances d'avoir plus d'une solution. Plus d est grand, plus le problème a de chances d'avoir beaucoup de solutions, et en trouver une est alors plus facile.

Nous considérons le cas pour lequel la densité est proche de 1. Des paramètres typiques sont alors :

— $\mathbf{x}$ est constitué de n entiers compris entre 0 et 2^n ;

— t est choisi aléatoirement entre 0 et $n2^n$.

Une façon naïve de résoudre ce problème serait d'énumérer tous les éléments de $\{0,1\}^n$ jusqu'à trouver un $\mathbf{e}$ tel que $\sum_{i=1}^{n} \mathbf{e}_i x_i = t$. Cela prendrait un temps $\mathcal{O}(2^n)$.

1.1.7 Conventions de complexité

Il n'a pas été prouvé que le problème de remplissage de sac à dos en moyenne soit NP-complet. Néanmoins, ce problème est difficile à résoudre et on ne connaît pas pour le moment d'algorithme polynomial ou sous-exponentiel pour le résoudre.

Tous les algorithmes présentés auront donc une complexité exponentielle en n, l'objectif étant de réduire l'exposant autant que possible. Plus précisément, la complexité de tous les algorithmes étudiés sera présentée sous la forme $\tilde{\mathcal{O}}(2^{\alpha n})$.

1.2 Recherche de collisions

Horowitz et Sahni ont montré ([HS74]) qu'il est possible de résoudre le problème du sac à dos beaucoup plus rapidement que la méthode naïve, en exploitant une méthode de type recherche de collisions. Cependant, cela se fait au prix de l'utilisation de beaucoup de mémoire.

Pour éviter d'alourdir les calculs, on suppose ici que n est pair (dans le cas impair, il faut remplacer les $n/2$ par $\lfloor n/2 \rfloor$). Au lieu d'énumérer tous les vecteurs $\mathbf{e} \in \{0,1\}^n$, et de calculer toutes les sommes $\sum_{i=1}^{n} \mathbf{e}_i x_i$ possibles, on calcule toutes les valeurs possibles pour $\sum_{i=1}^{n/2} \mathbf{e}_i x_i$, pour $\sum_{i=n/2+1}^{n} \mathbf{e}_i x_i$, et on cherche une collision. Autrement dit, au lieu de rechercher directement $\mathbf{e}$, on regarde toutes les moitiés gauches $\mathbf{e}^\ell \in \mathbb{N}^{n/2}$ possibles, toutes les moitiés droites $\mathbf{e}^r$, puis on cherche une moitié gauche et une moitié droite compatibles. Pour cela, on garde en mémoire toutes les moitiés gauches possibles $\mathbf{e}^\ell$, ainsi que les sommes $s^\ell = \sum_{i=1}^{n/2} \mathbf{e}_i^\ell x_i$ associées. Ces couples sont triés selon la valeur s^ℓ. Ensuite, pour chaque moitié droite possible $\mathbf{e}^r$, on calcule la somme s^r correspondante, et on cherche si $t - s^r$ est présent dans la liste des moitiés gauches. Si c'est le cas, on a trouvé une solution.

Dans cet algorithme, on réalise successivement deux énumération de $\{0,1\}^{n/2}$, ainsi qu'un tri d'une liste de taille $2^{n/2}$. Cet algorithme a donc une complexité en temps $\tilde{\mathcal{O}}(2^{n/2})$, et en mémoire $\mathcal{O}(2^{n/2})$.

1.2.1 Variante avec plus de solutions

Nous nous plaçons temporairement dans le cas d'une densité un peu plus élevée : $1 \leqslant d \leqslant 2$. L'espérance du nombre de solutions est $\tilde{\mathcal{O}}\left(2^{(1-1/d)n}\right)$ (proposition 1.1), et varie donc entre $\tilde{\mathcal{O}}(1)$ pour $d = 1$, et $\tilde{\mathcal{O}}\left(2^{n/2}\right)$ pour $d = 2$. Il est possible d'adapter très légèrement l'algorithme de Horowitz et Sahni pour renvoyer toutes les solutions : quand on trouve une solution, on la garde en mémoire et on continue au lieu de s'arrêter en renvoyant la solution trouvée. On peut alors trouver toutes les $\tilde{\mathcal{O}}\left(2^{(1-1/d)n}\right)$ solutions sans modifier les complexités en temps (qui reste à $\tilde{\mathcal{O}}(2^{n/2})$), ni en mémoire ($\mathcal{O}(2^{n/2})$).

Algorithm 2: Horowitz et Sahni

Data: $\mathbf{x} = (x_1, \ldots, x_n)$, t
$L_\ell := NULL$
for $\mathbf{e}^\ell \in \{0,1\}^{n/2}$ **do**
  $s^\ell := \sum_{i=1}^{n/2} \mathbf{e}_i^\ell x_i$
  $L_\ell := L_\ell, (s^\ell, \mathbf{e}^\ell)$
end
sort L_ℓ selon la première coordonnée
for $\mathbf{x}^r \in \{0,1\}^{n/2}$ **do**
  $s^r := \sum_{i=n/2+1}^{n} \mathbf{e}_i^r x_i$
  search $(t - s^r, \mathbf{e}^\ell)$ in L_ℓ
  if *found* **then**
    return $(\mathbf{e}^\ell, \mathbf{e^r})$
  end
end
return "no solution"

1.2.2 Variante modulaire

Une variante du problème du sac à dos, dont nous aurons besoin plus tard, est la suivante : $\mathbb{Z}$ a été remplacé par l'anneau modulaire $\mathbb{Z}/c\mathbb{Z}$.

Problème : Sac à dos modulaire
Entrée : $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{N}^n$
$t \in \mathbb{N}$
$c \in \mathbb{N}$

Sortie : $\mathbf{e} \in \{0,1\}^n$ tel que $\sum_{i=1}^{n} \mathbf{e}_i x_i \equiv t\ [c]$ s'il existe une solution
"impossible" sinon

L'algorithme de Horowitz et Sahni ne demande que de très légères modifications pour fonctionner dans ce cas, à savoir remplacer les opérations entières par des opérations modulaires.

Comme dans le cas de $\mathbb{Z}$, il est possible, pour la même complexité, d'obtenir la liste de toutes les solutions. Nous appellerons cet algorithme AllSolutionsMod.

1.2.3 Conditions intermédiaires

Nous revenons au cas de la densité $d = 1$. Nous avons vu que la recherche de collisions permet d'accélérer la résolution du problème du sac à dos : il est plus efficace d'énumérer les parties gauches et droites possibles d'un vecteur, et de chercher des collisions, plutôt que d'énumérer directement tous les vecteurs possibles. Il est alors naturel de se demander si on ne pourrait pas aller plus loin. Par exemple, ne pourrait-on pas découper la recherche en quatre plutôt qu'en deux ?

C'est ce qui est fait dans l'algorithme présenté par Schroeppel et Shamir [SS81]. L'idée de cet algorithme est la suivante : dans [HS74], on construit tous les sous-ensembles possibles pour les parties gauche et droite de la solution. On cherche ensuite une collision, c'est-à-dire une partie gauche et une partie droite dont les sommes des éléments sont respectivement s et $t - s$.

Dans [SS81], on se rajoute une contrainte arbitraire sur ce s. On procède alors comme précédemment, mais en n'énumérant que les parties gauches et droites qui satisfont cette nouvelle contrainte. S'il y a une solution, on a terminé. S'il n'y a pas de solution, on recommence avec une valeur différente pour la contrainte arbitraire, jusqu'à avoir épuisé toutes les possibilités.

Algorithm 3: Schroeppel et Shamir

Data: $\mathbf{x} = (x_1, \ldots, x_n)$, t
choose $c \simeq 2^{n/4}$
for t' *in* $[0, c-1]$ **do**
  $L_\ell :=$ AllSolutionsMod$((x_1, \ldots, x_{n/2}), t', c)$
  $L_r :=$ AllSolutionsMod$((x_{n/2+1}, \ldots, x_n), t - t', c)$
  sort L_ℓ selon la première coordonnée
  for $(s^r, \mathbf{e}^r)$ *in* L_r **do**
    search $(t - s^r, \mathbf{e}^\ell)$ in L_ℓ
    if *found* **then**
      return $(\mathbf{e}^\ell, \mathbf{e}^r)$
    end
  end
end
return "no solution"

Dans cet algorithme, la complexité de la sous-routine AllSolutionsMod est $\tilde{\mathcal{O}}(2^{n/4})$, en temps comme en mémoire. Par ailleurs, on s'attend, pour chaque choix de t', à ce que L_ℓ et L_r soient de taille environ $2^{n/4}$. Enfin, la boucle sur t' est exécutée en moyenne $\mathcal{O}(2^{n/4})$ fois avant de trouver une solution.

L'algorithme de Shroeppel et Shamir résout donc le problème du sac à dos en temps $\tilde{\mathcal{O}}(2^{n/2})$, et en mémoire $\mathcal{O}(2^{n/4})$.

1.2.4 Structure d'arbre

Chaque itération de l'algorithme de Schroeppel et Shamir peut être décrit sous la forme d'un arbre binaire, chaque nœud étant occupé par une liste de vecteurs. Les listes occupant les feuilles sont obtenues en énumérant toutes les possibilités, tandis que les autres listes sont obtenues en fusionnant les listes correspondant aux deux nœuds fils. Dans ce cadre, par convention, on numérotera les niveaux de l'arbre en partant de la racine (qui se voit attribuer le numéro 0). De cette façon, il y a 2^j listes au niveau j. Dans le cas de l'algorithme de Schroeppel et Shamir le dernier niveau est le niveau 2, mais la même logique s'appliquera pour décrire des algorithme plus profonds.

— le *niveau* 2 contient quatre listes, ce sont les feuilles, et chacune est obtenue en énumérant les $2^{n/4}$ possibilités pour les $n/4$ coordonnées qui lui sont attribuées ;

— le *niveau* 1 contient deux listes. Chacune est de taille $2^{n/4}$ et est obtenue en fusionnant deux des feuilles, sous une certaine contrainte de modulo de taille $2^{n/4}$;

— le *niveau 0* est la racine de l'arbre, c'est la liste qui contient les solutions (s'il en existe).

1.2.5 Cas d'une très grande densité : l'algorithme de Wagner

Lorsque la densité est bien plus grande que 1, le problème du sac à dos admet de nombreuses solutions, et en trouver une seule devient alors beaucoup plus facile. Encore faut-il savoir comment. Prenons l'algorithme de Schroeppel et Shamir, mais supposons maintenant que la densité vérifie $d \geqslant 4/3$. Il y a dans ce cas au moins $2^{n/4}$ solutions, qu'on suppose réparties indépendamment et uniformément sur les différentes valeurs de la contrainte arbitraire de modulo c. Ainsi, il suffit d'essayer un nombre polynomial de t' différents pour trouver une solution, au lieu de parcourir les $2^{n/4}$ possibilités. Ainsi, si la densité est suffisamment grande, l'algorithme de Schroeppel et Shamir trouve une solution en temps $\tilde{\mathcal{O}}\left(2^{n/4}\right)$.

Nous venons de constater que la contrainte de modulo, qui forçait une répétition dans le cas $d = 1$, n'est plus un problème quand on dispose de beaucoup de solutions. Quand il n'y avait qu'une solution, il fallait parcourir toutes les valeurs possibles de la contrainte de modulo jusqu'à trouver celle qui convient à la solution. Maintenant, il suffit d'en trouver une qui convient à l'une des solutions. Si le nombre de solutions dépasse le nombre de valeurs possibles pour la contrainte de modulo, il n'y a plus besoin d'un nombre exponentiel de répétitions.

Si le nombre de solutions dépasse **largement** le nombre de valeurs possibles, on peut même augmenter le nombre de contraintes de modulo arbitraires, et découper le problème en morceaux encore plus petits. C'est ce qui

est fait dans [Wag02] : Wagner présente une série d'algorithmes qui cherchent une solution en la découpant en morceaux de plus en plus petits, qui sont de plus en plus efficaces, mais exigent également de plus en plus de solutions. Les algorithmes de Wagner suivent la structure d'arbre exposée en 1.2.4, chaque choix k du numéro du dernier niveau produisant un algorithme différent. Pour de petites valeurs de k, cela donne des algorithmes connus :

— pour $k = 1$, il n'y a que les niveaux 0 et 1, il s'agit de l'algorithme de Horowitz et Sahni ;

— pour $k = 2$, il s'agit de la variante de l'algorithme de Schroeppel et Shamir sans répétition.

De façon générale, l'algorithme de Wagner pour $k+1$ étages est construit de la façon suivante :

— à l'étage k, on construit 2^k listes, la i^{e} liste étant l'énumération complète des $2^{n/2^k}$ possibilités pour les $n/2^k$ coordonnées, allant de la coordonnée $in/2^k$ à la coordonnée $(i+1)n/2^k - 1$.

— on fusionne ces listes deux par deux avec une contrainte de modulo sur $n/2^k$ bits, pour obtenir 2^{k-1} listes contenant chacune $2^{n/2^k}$ éléments.

— on répète cette opération jusqu'à n'obtenir plus que deux listes. On ajoute à chaque étage une contrainte modulaire sur $n/2^k$ bits, ce qui donne des listes de $2^{n/2^k}$ éléments.

— on fusionne ces deux listes avec une contrainte de modulo sur $2n/2^k$ bits (deux fois plus grande que pour les autres niveaux). On obtient alors une liste de taille polynomiale.

— Si cette liste contient un élément, c'est une solution et on la renvoie. Si elle n'en contient pas, on recommence en modifiant les contraintes de modulo. Cependant, l'espérance du nombre de répétitions est polynomiale, et n'intervient pas dans l'exposant de la complexité.

La complexité de cet algorithme est $\tilde{\mathcal{O}}\left(2^{n/2^k}\right)$ en temps et en mémoire, ce qui est le coût de la construction de chacune des listes. Le nombre de listes, 2^{k+1}, peut être un gros facteur multiplicatif, mais reste une constante qui disparaît dans la notation $\tilde{\mathcal{O}}$.

Comptons le nombre de contraintes de modulo arbitraires. Pour passer d'un niveau i à un niveau $i-1$, on fusionne 2^{i-1} couples de listes en 2^{i-1} listes. On utilise donc $2^{i-1} - 1$ contraintes de modulo arbitraires (la somme de toutes les contraintes est fixée par la cible t, la dernière contrainte est donc déterminée par les autres et ne peut pas être choisie arbitrairement). Chacune de ces contraintes de modulo étant une contrainte sur $n/2^k$ bits, le nombre total de bits de contrainte est $\dfrac{n}{2^k} \sum_{i=1}^{k} \left(2^{i-1} - 1\right)$, c'est-à-dire $\left(1 - \dfrac{k+1}{2^k}\right) n$.

On en déduit donc :

Proposition 1.2. *Pour une instance du problème de sac à dos de densité* $d = \frac{2^k}{k+1}$*, l'algorithme de Wagner à* $k+1$ *niveaux trouve une solution en temps* $\tilde{\mathcal{O}}\left(2^{n/2^k}\right)$.

Démonstration. D'après la proposition 1.1, comme la densité est $\frac{2^k}{k+1}$, on sait que l'espérance du nombre de solutions est $\tilde{\mathcal{O}}\left(2^{\left(1-\frac{k+1}{2^k}\right)n}\right)$. Puisque l'algorithme de Wagner sur $k+1$ étages utilise $\left(1-\frac{k+1}{2^k}\right)n$ bits de contraintes arbitraires, il trouve en moyenne $\tilde{\mathcal{O}}(1)$ solutions. Il suffit alors de le répéter un nombre de fois polynomial. □

Le cas $k = 3$ est illustré dans la figure 1.1. Dans de cas, la densité est $d = 2$, ce qui signifie que l'ordre de grandeur des vecteurs d'entrée et de la cible t est $2^{n/2}$, ce qui correspond effectivement à la contrainte imposée au niveau 0.

Remarque 1.1 (Vérification de la cohérence). D'un côté, à chaque niveau, on filtre sur $n/2^k$ bits, sauf au niveau 0 où on filtre sur $2n/2^k$ bits. Les éléments de la liste du niveau 0 sont donc contraints sur $\frac{k+1}{2^k}$ bits. De l'autre côté, puisque $d = \frac{2^k}{k+1}$, l'ordre de grandeur des x_i et de t est $\frac{k+1}{2^k}$ bits. La contrainte totale de modulo au niveau 0 correspond bien à la taille de t : la liste du niveau 0 ne contient que des solutions.

Remarque 1.2. On doit renvoyer un vecteur à n coordonnées, ce qui laisse 2^n possibilités. En particulier, si on a 2^n solutions, cela signifie qu'il n'y a que des solutions. Or les algorithmes de Wagner, quand on augmente le nombre de niveaux k, exigent très vite près de 2^n solutions. Déjà pour une dizaine de niveaux, cela demande un nombre de solution monstrueux. Quand on utilisera l'algorithme de Wagner, ce sera plutôt avec k de l'ordre de 3 ou 4, on poussera éventuellement jusqu'à 6 ou 7 dans certains cas particuliers.

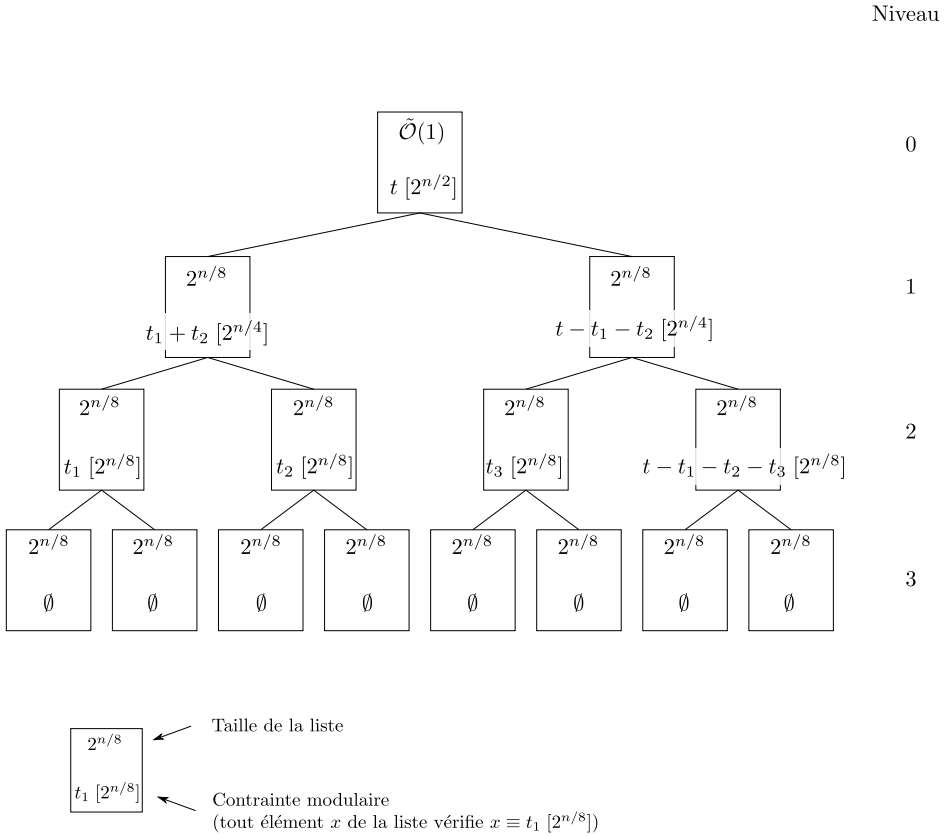

FIGURE 1.1 – Structure de l'algorithme de Wagner dans le cas où $k = 2$ (et donc $d = 2$).

1.2.6 Lissage de l'algorithme de Wagner

Précédemment, nous avons calculé que l'algorithme de Wagner, si le problème admet au moins $2^{\left(1-\frac{k+1}{2^k}\right)n}$ solutions, peut en trouver une en temps $\tilde{\mathcal{O}}\left(2^{n/2^k}\right)$. Cela correspond aux cas optimaux. Quand le nombre de solutions ne tombe pas sous la forme $2^{\left(1-\frac{k+1}{2^k}\right)n}$, il est toujours possible de choisir le plus grand k qui convient (après tout, pour k niveaux, il faut au moins $2^{\left(1-\frac{k}{2^{k-1}}\right)n}$, solutions, l'algorithme fonctionne encore s'il y en a plus). On peut cependant faire mieux en profitant de cet excédant de solutions : on peut créer une sorte d'interpolation entre les algorithmes à k et à $k+1$ niveaux.

Si le nombre de solutions est compris entre $2^{\left(1-\frac{k}{2^{k-1}}\right)n}$ et $2^{\left(1-\frac{k+1}{2^k}\right)n}$ (c'est-à-dire si la densité vérifie $\frac{2^{k-1}}{k} \leqslant d \leqslant \frac{2^k}{k+1}$), on peut utiliser l'algorithme suivant (le cas $k = 3$ est illustré dans la figure 1.2) :

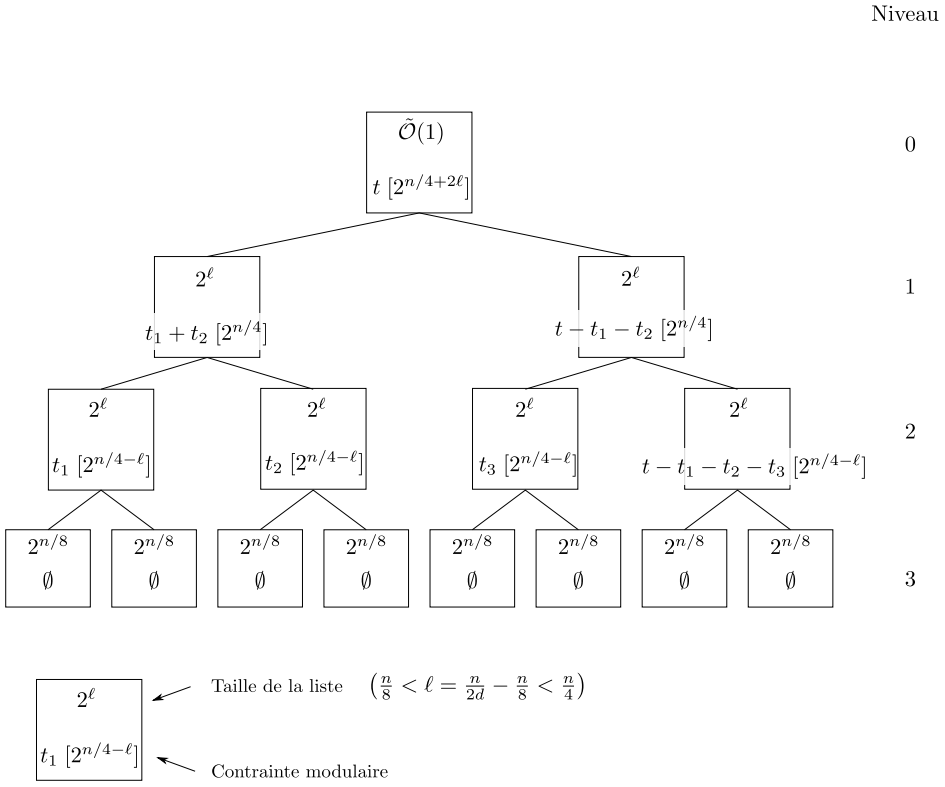

FIGURE 1.2 – Interpolation entre les algorithme de Wagner pour $k = 2$ et $k = 3$: il y a bien un niveau 3, mais les listes y sont plus petites, de même que la condition modulaire pour passer au niveau 2.

— Au niveau k, on construit 2^k listes, la i^{e} liste étant l'énumération complète des $2^{n/2^k}$ possibilités pour les $n/2^k$ coordonnées, allant de la coordonnée $in/2^k$ à la coordonnée $(i+1)n/2^k - 1$ (cette étape ne diffère pas de l'algorithme à $k+1$ niveaux.)

— Pour le niveau $k-1$, on fusionne les listes du niveau k deux par deux, avec une contrainte de modulo sur $cn = \frac{n}{k-1}\left(\frac{k}{2^{k-1}} - \frac{1}{d}\right)$ bits. On obtient alors 2^{k-1} listes de taille $\tilde{\mathcal{O}}\left(2^{\ell n}\right)$, pour $\ell = \frac{1}{k-1}\left(\frac{1}{d} - \frac{1}{2^{k-1}}\right)$.

— Pour les niveaux $k-2$ jusque 1, on fusionne les listes en ajoutant une contrainte de modulo sur ℓn bits, on obtient des listes de taille $\tilde{\mathcal{O}}\left(2^{\ell n}\right)$.

— pour le niveau 0, on fusionne les deux listes du niveau 1 avec une contrainte de modulo sur $2\ell n$ bits. Cela crée une liste dont l'espérance de la taille est polynomiale.

— Si la liste du niveau 0 possède une solution, on la renvoie. Sinon, on recommence avec d'autres valeurs pour les contraintes de modulo arbitraires. Ici encore, l'espérance du nombre de répétitions est polynomial

en n et n'entre pas en compte dans l'exposant de la complexité.

Les listes du niveau k sont de taille $\tilde{\mathcal{O}}\left(2^{n/2^k}\right)$, et prennent un temps $\tilde{\mathcal{O}}\left(2^{n/2^k}\right)$ à construire. Les listes des niveaux $k-1$ à 1 sont de taille $\tilde{\mathcal{O}}\left(2^{\ell n}\right)$ où $\ell = \frac{1}{k-1}\left(\frac{1}{d} - \frac{1}{2^{k-1}}\right) \geqslant 1/2^k$. La construction des listes du niveau k est donc négligeable devant le temps requis pour les listes des autres niveaux. La complexité de cet algorithme est $\tilde{\mathcal{O}}\left(2^{\frac{1}{k-1}\left(\frac{1}{d}-\frac{1}{2^{k-1}}\right)n}\right)$, ce qui prouve :

Proposition 1.3. *Pour une instance du problème de sac à dos de densité* $\frac{2^{k-1}}{k} \leqslant d \leqslant \frac{2^k}{k+1}$*, l'algorithme de Wagner à* $k+1$ *niveaux trouve une solution en temps* $\tilde{\mathcal{O}}\left(2^{\frac{1}{k-1}\left(\frac{1}{d}-\frac{1}{2^{k-1}}\right)n}\right)$.

La figure 1.3 montre la complexité de cet algorithme en fonction de la densité.

Remarque 1.3. Dans le cas extrême $d = \frac{2^k}{k+1}$ (c'est-à-dire $2^{\left(1-\frac{k+1}{2^k}\right)n}$ solutions), la contrainte de modulo entre les niveaux k et $k-1$ devient $cn = \frac{n}{2^k}$ et la taille des listes se simplifie en $\ell n = \frac{n}{2^k}$: on a retrouvé l'algorithme de Wagner à $k+1$ niveaux.

Dans l'autre cas extrême $d = \frac{2^{k-1}}{k}$ (c'est-à-dire $2^{\left(1-\frac{k}{2^{k-1}}\right)n}$ solutions), on obtient $c = 0$ et $\ell n = \frac{n}{2^{k-1}}$. Il s'agit de l'algorithme de Wagner à $k-1$ niveaux, à ceci près que les listes du niveau $k-1$ sont construites comme le produit cartésien de l'énumération des moitiés gauches et droites, au lieu d'énumérer directement sur les $\frac{n}{2^{k-1}}$ coordonnées.

Remarque 1.4. Comme on peut le constater sur la figure 1.3, le lissage de l'algorithme de Wagner n'est pas "parfait" : la pente subit une discontinuité à chaque changement du nombre de niveaux.

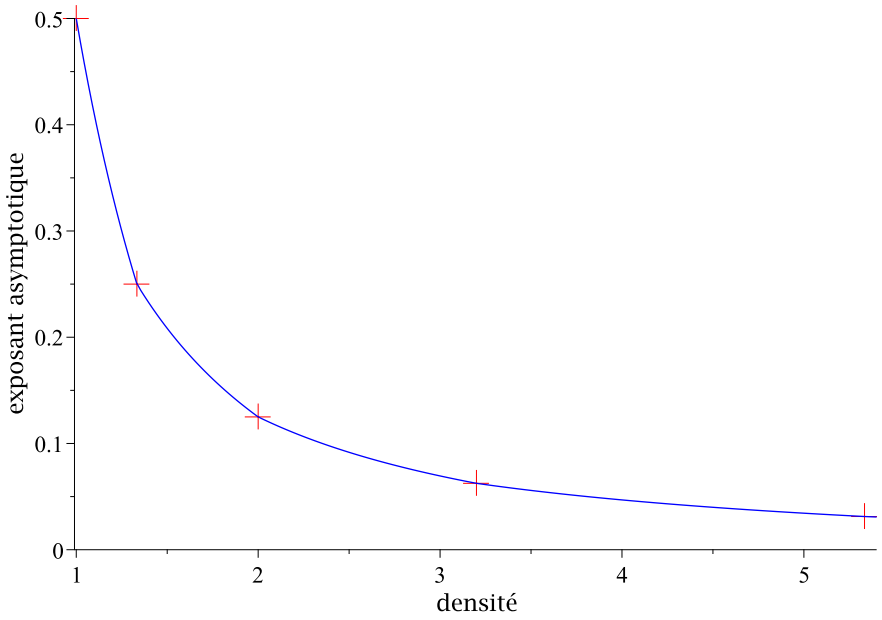

FIGURE 1.3 – Croix rouges : exposant de la complexité de l'algorithme de Wagner pour un nombre de niveaux variant de 2 à 6 – Courbe bleue : exposant de la complexité de l'algorithme de Wagner lissé

1.3 Représentations

Dans la section précédente, nous avons vu au travers de l'algorithme de Wagner que le problème de remplissage de sac à dos est d'autant plus facile qu'il y a beaucoup de solutions. Nous allons donc étudier un moyen d'ajouter artificiellement un grand nombre de solutions : les représentations. Cette technique a été introduite par Howgrave-Graham et Joux dans [HJ10].

1.3.1 Définition

Jusqu'à maintenant, quand nous avons fait des collisions, nous avons toujours cherché le vecteur solution $\mathbf{e}$ comme la concaténation de sa partie gauche et sa partie droite. De cette façon, un vecteur admet toujours exactement un découpage. À cause de cette unicité, quand on rajoute une contrainte arbitraire (par exemple la condition $t' \mod c$ de l'algorithme de Schroeppel et Shamir), une seule valeur peut convenir pour ce t', à savoir le t' qui correspond au découpage de la solution. Avec l'algorithme de Wagner, nous avons vu que cela n'est pas un problème quand le nombre de solution est élevé. Cependant, quand la densité est $d = 1$, le nombre de solutions n'est pas suffisant.

Pour contourner ce problème, on va ajouter artificiellement des solutions

en utilisant la technique des représentations. Pour ce faire, on ne cherche plus un vecteur de taille n comme la concaténation de deux vecteurs de taille $n/2$, mais comme la somme de deux vecteurs de taille n (mais qui doivent vérifier d'autres propriétés). Un vecteur peut être décomposé de plusieurs (beaucoup de) façons différentes comme la somme de deux autres vecteurs, chaque décomposition étant appelée une *représentation*. Chacune de ces décompositions étant suffisante pour retrouver la solution, on peut se contenter d'en trouver une seule. Ainsi, si on utilise des contraintes de modulo t' arbitraires, il n'est plus nécessaire d'essayer toutes les valeurs possibles, pour trouver celle qui convient à **la** décomposition : il suffit d'en essayer assez pour trouver celle qui convient à **l'une des** représentations.

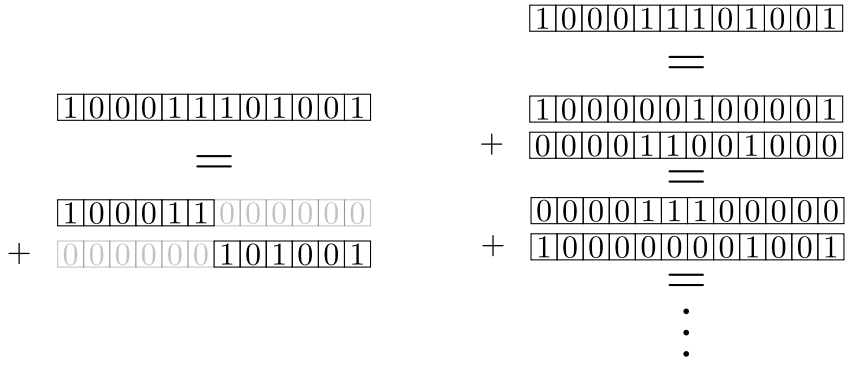

FIGURE 1.4 – À gauche : décomposition d'un vecteur en sa partie gauche et sa partie droite. À droite : décomposition du même vecteur de différentes façons à l'aide de représentations.

On rappelle la définition d'une distribution, introduite en 0.1.4 : la distribution $D^n[\alpha]$ est l'ensemble des vecteurs de taille n dont $\lfloor \alpha n \rfloor$ coordonnées sont des 1 et les autres sont des 0.

Définition 1.1 (Représentations)**.** *Soit* $\mathbf{e}$ *un vecteur de distribution* D*. Une paire* $(\mathbf{e}_1, \mathbf{e}_2)$ *est une* $(D, D_1, D_2)-$représentation *de* $\mathbf{e}$ *si* $\mathbf{e} = \mathbf{e}_1 + \mathbf{e}_2$*, et si* $\mathbf{e}_1 \in D_1$ *et* $\mathbf{e}_2 \in D_2$*.*

Définition 1.2 (Représentations)**.** *On note alors* $Rep_{D_1,D_2}(\mathbf{e})$ *l'ensemble des* $(D, D_1, D_2)-$*représentation de* $\mathbf{e}$*.*

Par exemple, la figure 1.4 illustre la décomposition d'un vecteur de distribution $D^{12}[1/2]$. On peut le rechercher comme la combinaison de sa partie gauche et de sa partie droite. Une autre façon de faire est de la rechercher comme la somme de deux vecteurs de $D^{12}[1/4]$ (c'est-à-dire utiliser des $(D^{12}[1/2], D^{12}[1/4], D^{12}[1/4])-$représentations). Il y a alors de nombreuses décompositions (20, pour être précis), dont 2 sont représentées sur la figure de droite.

Pour résoudre le problème de remplissage de sac à dos, nous ne cherchons pas des vecteurs de taille 12, mais de taille n. Nous allons donc construire deux listes de vecteurs de taille n dont $n/4$ coordonnées sont des 1, et les $3n/4$ autres sont des 0, puis chercher des collisions entre ces listes.

1.3.2 Éléments mal formés

Malheureusement, quand on additionne deux vecteurs $\mathbf{e}_1$ et $\mathbf{e}_2$ pris aléatoirement dans $D^n[1/4]$, la probabilité d'obtenir un vecteur de $D^n[1/2]$ est faible : si, à une certaine coordonnée, $\mathbf{e}_1$ et $\mathbf{e}_2$ ont tous les deux un 1, on obtient un 2 dans $\mathbf{e_1} + \mathbf{e_2}$. Ainsi, même si le vecteur $\mathbf{e_1} + \mathbf{e_2}$ a la bonne somme (c'est-à-dire $\sum_{i=1}^{n}(\mathbf{e}_1 + \mathbf{e}_2)_i x_i = t$), ce vecteur ne peut pas être retenu car il ne contient pas uniquement des 0 et des 1.

Définition 1.3 (Élement mal formé)**.** *Soient D, D_1 et D_2 des distributions. Supposons qu'on utilise des $(D, D_1, D_2)-$représentations : on cherche des collisions entre une liste de vecteurs de D_1 et une liste de vecteurs de D_2, pour trouver des vecteurs de D vérifiant une certaine condition de modulo. Un vecteur $\mathbf{e} = \mathbf{e}_1 + \mathbf{e}_2$ est un* élément mal formé *si $\mathbf{e}_1 \in D_1$, $\mathbf{e}_2 \in D_2$, $\mathbf{e}$ vérifie la contrainte modulaire, mais $\mathbf{e} \notin D$.*

Définition 1.4 (Élement bien formé)**.** *Si au contraire $\mathbf{e}$ vérifie la contrainte modulaire ainsi que la contrainte de distribution $\mathbf{e} \in D$, on parlera d'*élément bien formé*.*

Le problème des éléments mal formés est qu'on ne sait pas s'en débarrasser facilement. Dans l'algorithme de Schroeppel et Shamir, lors de la construction des listes intermédiaires, il n'y a pas de représentations (et donc pas d'éléments mal formés). À cette étape, on peut construire des listes de taille $2^{n/4}$ en temps $\tilde{\mathcal{O}}(2^{n/4})$. Dans le cas des représentations, on ne sait pas construire directement la liste des éléments bien formés. Pour obtenir cette liste, on commencera par construire la liste de tous les éléments qui satisfont la condition de modulo, puis on éliminera les éléments mal formés [1]. Il pourra donc y avoir une différence d'ordre de grandeur entre les tailles des listes et le temps requis pour les construire.

1. En réalité, il est possible d'être plus efficace et ne pas payer la totalité de la liste des éléments mal formés. Il existe des techniques, appelées recherche de *plus proches voisins*, de *presque-collisions* ou encore *filtrage contraint*, qui permettent d'améliorer asymptotiquement la complexité de la fusion de deux listes (par exemple [Oze16]). Ces techniques étaient, jusqu'à récemment, très pénibles à utiliser tout en ajoutant des sur-coûts superpolynomiaux. Néanmoins, une étude complète des algorithmes de remplissage de sac à dos requerrait la prise en compte de ces méthodes et permettrait de concevoir des algorithmes encore plus efficaces asymptotiquement.

Lemme 1.4. *Soient α et β deux réels positifs tels que $\alpha+\beta<1$. Le nombre de façon de choisir $\mathbf{e}_1 \in D^n[\alpha]$ et $\mathbf{e}_2 \in D^n[\beta]$ tels que $\mathbf{e_1}+\mathbf{e_2} \in D^n[\alpha+\beta]$ est :*

$$\tilde{\mathcal{O}}\left(2^{n\tilde{h}(\alpha,\beta)}\right)$$

Démonstration. Soient $\mathbf{e}_1 \in D^n[\alpha]$ et $\mathbf{e}_2 \in D^n[\beta]$. Si les 1 de $\mathbf{e_1}$ et ceux de $\mathbf{e_2}$ n'occupent pas des positions différentes, cela crée des 2 dans la somme $\mathbf{e_1}+\mathbf{e_2}$. Puisqu'il n'y a pas de 2 dans un vecteur de $D^n[\alpha+\beta]$, il s'agit d'un vecteur mal formé.

Un couple $(\mathbf{e_1},\mathbf{e_2})$ compatible est donc caractérisé par deux ensembles disjoints : l'ensemble des αn positions occupées par les 1 de $\mathbf{e_1}$, et l'ensemble des βn positions occupées par les 1 de $\mathbf{e_2}$. Il y a $\binom{n}{\alpha n,\beta n} = \tilde{\mathcal{O}}\left(2^{n\tilde{h}(\alpha,\beta)}\right)$ façons de choisir ces ensembles. □

Ce lemme va nous servir à montrer deux propositions. La proposition 1.5 donne le nombre de représentations d'un vecteur. La proposition 1.6 permet de calculer la taille de la liste des vecteurs bien formés en fonction de la taille de la liste de toutes les collisions.

Proposition 1.5. *Soient α et β deux réels positifs tels que $\alpha+\beta<1$. Soit $\mathbf{e} \in D^n[\alpha+\beta]$. Le nombre de $(D^n[\alpha+\beta], D^n[\alpha], D^n[\beta])-$représentations de $\mathbf{e}$ est :*

$$Rep_{D^n[\alpha],D^n[\beta]}(\mathbf{e}) = \tilde{\mathcal{O}}\left(2^{n(\tilde{h}(\alpha,\beta)-h(\alpha+\beta))}\right)$$

Démonstration. Soit $\mathbf{f} \in D^n[\alpha+\beta]$. Comme $\mathbf{e}$ et $\mathbf{f}$ ont les même nombres de 0 et de 1, on peut passer de l'un à l'autre par une permutation de coordonnées. Cette même permutation permet d'établir une bijection entre les représentations de $\mathbf{e}$ et celles de $\mathbf{f}$. Tous les éléments de $D^n[\alpha+\beta]$ ont donc le même nombre de représentations.

Par ailleurs, le lemme 1.4 indique que le nombre total de représentations pour l'ensemble des vecteurs de $D^n[\alpha+\beta]$ est $\tilde{\mathcal{O}}\left(2^{n\tilde{h}(\alpha,\beta)}\right)$. Puisqu'il y a $\tilde{\mathcal{O}}\left(2^{n\tilde{h}(\alpha+\beta)}\right)$ tels vecteurs, on obtient :

$$Rep_{D^n[\alpha],D^n[\beta]}(\mathbf{e}) = \tilde{\mathcal{O}}\left(2^{n(\tilde{h}(\alpha,\beta)-h(\alpha+\beta))}\right)$$

□

Proposition 1.6. *Quand on fusionne deux listes en utilisant des $(D^n[\alpha+\beta], D^n[\alpha], D^n[\beta])-$représentations, la proportion d'éléments bien formés dans l'ensemble des collisions est :*

$$\tilde{\mathcal{O}}\left(2^{n(\tilde{h}(\alpha,\beta)-\tilde{h}(\alpha)-\tilde{h}(\beta))}\right)$$

Démonstration. Le nombre total de couples $(\mathbf{e_1}, \mathbf{e_2}) \in D^n[\alpha] \times D^n[\beta]$ est $\tilde{\mathcal{O}}\left(2^{n(h(\alpha)+h(\beta))}\right)$. Le lemme 1.4 indique que parmi tous ces couples, seuls $\tilde{\mathcal{O}}\left(2^{n\tilde{h}(\alpha,\beta)}\right)$ produisent un élément de $D^n[\alpha+\beta]$, les autres donnant des éléments mal formés. La proportion des éléments bien formés lorsqu'on somme deux vecteurs $\mathbf{e}_1 \in D^n[\alpha]$ et $\mathbf{e}_2 \in D^n[\beta]$ est donc :

$$\tilde{\mathcal{O}}\left(2^{n(\tilde{h}(\alpha,\beta)-h(\alpha)-h(\beta))}\right)$$

Par ailleurs, pour un couple $(\mathbf{e_1}, \mathbf{e_2}) \in D^n[\alpha] \times D^n[\beta]$, les conditions "$\mathbf{e_1} + \mathbf{e_2} \in D^n[\alpha + \beta]$" et "$\mathbf{e_1} + \mathbf{e_2}$ vérifie la contrainte de modulo" sont indépendantes (heuristique 1). Par conséquent, la proportion des éléments bien formés parmi la listes des éléments qui satisfont la contrainte modulaire est également $\tilde{\mathcal{O}}\left(2^{n(\tilde{h}(\alpha,\beta)-h(\alpha)-h(\beta))}\right)$. □

1.3.3 Utilisation des représentations

Pour illustrer l'utilité des représentations, nous construisons maintenant un algorithme simple de remplissage de sac à dos. La structure de cet algorithme sera illustré sur la figure 1.5. Il ne s'agit pas de la meilleure façon d'exploiter les représentations, mais de la plus simple.

On cherche un vecteur de taille n et de densité $1/2$ [2]. Pour cela, on utilise des représentations : on cherche ce vecteur comme la somme de deux vecteurs de taille n et de densité $1/4$. Ces vecteurs de densité $1/4$ seront, eux, recherchés en combinant des parties gauches et des parties droites. Pour simplifier les expressions en évitant les parties entières $\lfloor \cdot \rfloor$, on suppose que n est un multiple de 8.

Au niveau 2 : on construit quatre listes. Pour la première et la troisième liste, on énumère tous les vecteurs de poids $n/8$, dont les 1 sont placés parmi les $n/2$ premières coordonnées. Pour la deuxième et la quatrième liste, les 1 sont placés parmi les $n/2$ dernières coordonnées. Le nombre de vecteurs dans chacune de ces listes est $2^{\ell_2 n} = \tilde{\mathcal{O}}\left(2^{h(1/4)n/2}\right) \simeq \tilde{\mathcal{O}}\left(2^{0.405n}\right)$.

Au niveau 1 : on fusionne les listes du niveau 2 deux par deux. On utilise une contrainte modulaire sur $c_1 n$ bits, avec $c_1 = 1/2$. Les deux listes obtenues sont de taille $\tilde{\mathcal{O}}\left(2^{\ell_1 n}\right)$, où $\ell_1 = 2\ell_2 - c_1 \simeq 0.31$. Ces listes sont constituées de vecteurs de taille n, et de densité $1/4$.

2. À priori, il n'y a pas de raison pour que la solution, si elle existe, soit de densité $1/2$. Il ne s'agit que du cas le plus difficile : s'il existe une solution déséquilibrée, il est possible de la trouver plus facilement. Dans le cas de [HJ10], les auteurs détaillent comment, à partir de leur algorithme pour la densité $1/2$, obtenir un algorithme pour le cas déséquilibré. On admet qu'il est possible de faire la même chose dans notre cas.

Au niveau 0 **:** on fusionne les deux listes du niveau 1. Puisqu'on veut des solutions au problème de remplissage de sac à dos, la taille de la contrainte modulaire est sur $c_0 n = n$ bits, c'est-à-dire $n/2$ bits de plus qu'au niveau 1. La taille de cette liste serait donc $\tilde{\ell_0} = 2\ell_1 - (c_0 - c_1) = 0.12$. Cependant, il y a des représentations, ce qui produit des vecteurs mal formés (*i.e.* des vecteurs qui contiennent des 2). Selon le théorème 1.6, la proportion des vecteurs bien formés est $\tilde{\mathcal{O}}\left(2^{p_0 n}\right) = \tilde{\mathcal{O}}\left(2^{n(\tilde{h}(1/4,1/4)-2h(1/4)}\right) \simeq \tilde{\mathcal{O}}\left(2^{-0.12n}\right)$. Ainsi, parmi les $\tilde{\mathcal{O}}\left(2^{0.12n}\right)$ éléments qui vérifient la contrainte modulaire, l'espérance du nombre de vecteurs bien formés est $\tilde{\mathcal{O}}\left(2^{(\tilde{\ell_0}+p_0)n}\right) = \tilde{\mathcal{O}}(1)$.

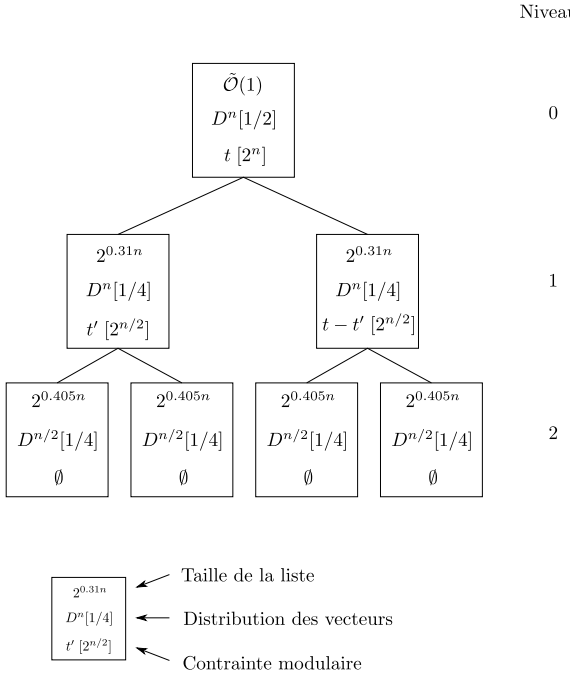

FIGURE 1.5 – Algorithme simple utilisant des représentations entre les niveaux 0 et 1.

Dans cet algorithme, les étapes les plus coûteuses peuvent être effectuées en temps $\tilde{\mathcal{O}}\left(2^{0.405n}\right)$. C'est le cas de la construction des listes du niveau 2 (qui sont de taille $\tilde{\mathcal{O}}\left(2^{0.405n}\right)$), et c'est aussi le cas du calcul des listes du niveau 1 (qui demande de lire les listes du niveau 2).

Par ailleurs, l'espérance du nombre de solutions trouvées (*i.e.* la taille de la liste du niveau 0) est $\tilde{\mathcal{O}}(1)$. Cela signifie qu'il suffit de répéter l'algorithme avec un nombre polynomial de valeurs différentes pour t'.

Finalement, la complexité en temps de cet algorithme est $\tilde{\mathcal{O}}\left(2^{0.405n}\right)$, ce qui est une amélioration significative par rapport à l'algorithme de Schroeppel et Shamir.

Encore une fois, il s'agit de l'algorithme le plus simple possible qui ex-

ploite les représentations, et il est possible de faire bien mieux. En introduisant les représentations, [HJ10] a directement proposé un algorithme dont la complexité est $\tilde{\mathcal{O}}\left(2^{0.337n}\right)$. Leur algorithme comporte un niveau supplémentaire. Les vecteurs de $D^n[1/4]$ du niveau 1 sont alors recherchés comme la somme de deux vecteurs de densité 1/8. Ces vecteurs de densité 1/8 sont recherchés comme la combinaison d'une partie gauche et d'une partie droite.

1.3.4 Doublons

L'utilisation de la technique des représentations peut, si les paramètres ne sont pas bien choisis, rendre un algorithme très mauvais en produisant des *doublons*. C'est ce qui se produit par exemple dans l'algorithme suivant, illustré sur la figure 1.6

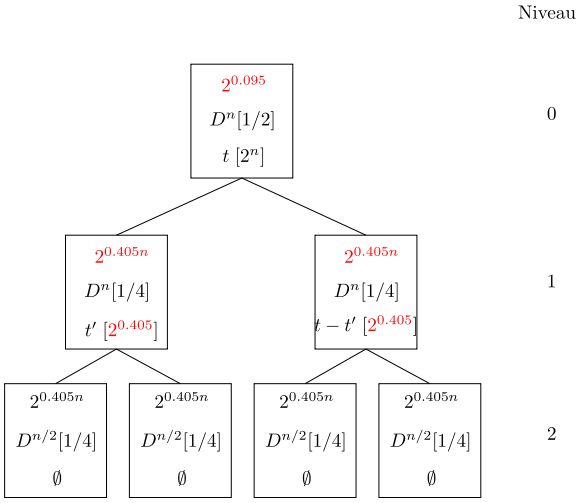

FIGURE 1.6 – Variante de l'algorithme précédent, utilisant des paramètres qui conduisent à la création de doublons.

Il s'agit d'une légère modification de l'algorithme présenté juste avant. Au lieu d'utiliser une contrainte modulaire $c_1 = 0.5$ pour le niveau 1, puis une contrainte $c_0 - c_1 = 0.5$ au niveau 0, on utilise une contrainte plus faible pour le niveau 1 : $c_1 = 0.405$, puis on complète avec une contrainte $c_0 - c_1 = 0.595$.

Cela change les tailles des listes : les listes du niveau 1 on maintenant une taille $\ell_1 = 0.405$, et la liste du niveau 0 une taille $\ell_0 = 0.095$. Le facteur limitant pour la complexité est toujours $\tilde{\mathcal{O}}\left(2^{0.405n}\right)$. Par ailleurs, puisque la liste du niveau 0 contient $2^{0.095n}$, cela signifie que cet algorithme trouve en moyenne $2^{0.095n}$ solutions.

Ce serait très intéressant si ce n'était pas **complètement faux !** Le problème n'admet en moyenne qu'une seule solution : on a donc ici trouvé $2^{0.095n}$ **copies** de la même solution.

Dans ce cas précis, ce n'est pas trop grave. Cela devient problématique quand on utilise des algorithmes qui contiennent plus d'étages. En effet, la présence de doublons fait que l'heuristique 1 n'est plus valable. Les fusions de listes successives produisent alors des listes dont les éléments ne correspondent pas à une distribution uniforme.

Le problème vient d'une incohérence entre le nombre de représentations et la contrainte modulaire. En cherchant un vecteur de poids $n/2$ comme la somme de deux vecteurs de poids $n/4$, la solution admet $2^{n/2}$ représentations (proposition 1.5). Dans l'algorithme correct (celui de la figure 1.5), la contrainte modulaire du niveau 1 est $c_1 = 0.5$. Ainsi, on conserve en moyenne une représentation de la solution. En revanche, dans l'algorithme incorrect (figure 1.6), on a choisit $c_1 = 0.405$. On a donc gardé en moyenne $2^{0.095n}$ représentations de la solution.

1.3.5 Amélioration des représentations avec des -1

Les représentations peuvent être utilisées plus efficacement en ajoutant des -1, comme l'ont montré Becker, Coron et Joux [BCJ11]. Au lieu de manipuler des vecteurs constitués uniquement de 0 et de 1, nous allons ajouter des -1. Pour cela, nous introduisons la notation provenant de [HM18] :

Définition 1.5 (Distribution)**.** *On note $D^n[s, \beta]$ l'ensemble des vecteurs de taille n, dont la somme des coefficients est $\lfloor sn \rfloor$, et dont $\lfloor \beta n \rfloor$ coordonnées sont des -1. Le nombre de 1 dans un tel vecteur est alors $\lfloor sn \rfloor + \lfloor \beta n \rfloor$.*

On se permettra d'omettre le deuxième paramètre dans le cas où celui-ci est nul. On retrouve alors la définition précédente de $D^n[\alpha]$.

Remarque 1.5. On a choisi de paramétrer les distributions avec la somme des coefficients et le nombre de -1. On aurait pu choisir plutôt les nombres de 1 et de -1. L'intérêt d'utiliser la somme des coordonnées est qu'il s'agit d'un paramètre additif : quand on utilise des $(D^n[s, \beta], D^n[s_1, \beta_1], D^n[s_2, \beta_2])-$ représentations, on a systématiquement $s = s_1 + s_2$.

En particulier, dans l'algorithme de [BCJ11], qui est symétrique, on a toujours $(s_1, \beta_1) = (s_2, \beta_2)$. On en déduit immédiatement que $s_1 = s/2$. Comme on cherche une solution dans $D^n[1/2, 0]$, les paramètres s_i sont fixés, il n'y a plus qu'à optimiser sur les β_i.

L'énoncé du problème n'a pas changé, on cherche toujours, comme solution, un vecteur contenant uniquement des 0 et des 1. On le cherchera, comme dans le cas de [HJ10], dans $D^n[1/2]$. Cependant, alors qu'on cherchait ce vecteur comme la somme de deux éléments de $D^n[1/4]$, on le cherchera maintenant comme la somme de deux éléments de $D^n[1/4, \beta]$, pour un β à optimiser.

Avec l'ajout des -1, il y a plus de façons de créer des éléments mal formés : la somme de deux -1 donne un -2, ce qu'on interdit. Par ailleurs, puisqu'il

faut trouver une solution ne contenant que des 0 et des 1, tout vecteur contenant encore un -1 au niveau 0 sera donc classé comme mal formé. Cependant, l'ajout des -1 permet également d'augmenter considérablement le nombre de représentations de la solution.

Théorème 1.7. *Soient $D^n[2s, \beta]$ et $D^n[s, \gamma]$ des distributions, quand on fusionne deux listes avec des $(D^n[2s, \beta], D^n[s, \gamma], D^n[s, \gamma])-$représentations, la proportion des éléments bien formés parmi l'ensemble des collisions est $\tilde{\mathcal{O}}\left(2^{pn}\right)$, où :*

$$p = \tilde{h}(\beta/2, \beta/2, \gamma - \beta/2, \gamma - \beta/2, s + \beta/2, s + \beta/2, 1 - 2s - \beta - 2\gamma) \\ - 2\tilde{h}(\gamma, s + \gamma, 1 - s - 2\gamma)$$

Démonstration. On omet la démonstration de ce théorème. Il s'agit d'un cas dégénéré du théorème 2.3 qui sera démontré dans le chapitre suivant. □

Sachant cela, on peut décrire l'algorithme de [BCJ11] et calculer sa complexité.

— *niveau* 4 : Il y a 16 listes à ce niveau. Pour les listes impaires, on énumère les éléments de $D^{n/2}[1/16, 0.0180] \times 0^{n/2}$, et pour les listes paires $0^{n/2} \times D^{n/2}[1/16, 0.0180]$. Ces listes contiennent $\tilde{\mathcal{O}}\left(2^{\ell_4 n}\right)$ éléments, où $\ell_4 = \tilde{h}(1/16 + 0.0180, 0.0180)/2 \simeq 0.2658$, et le temps requis pour les construire est $\mathcal{O}(2^{2658n})$.

— *niveau* 3 : On fusionne les listes du niveau 4 deux par deux, avec une contrainte de modulo $c_3 = 0.2407$. On obtient ainsi huit listes de taille $\tilde{\mathcal{O}}\left(2^{\ell_3 n}\right)$ où $\ell_3 = 2\ell_4 - c_3 \simeq 0.2909$, chacune contenant des vecteurs échantillonnés selon $D^n[1/16, 0.0180]$. Le complexité de la construction de ces listes est $\tilde{\mathcal{O}}(2^{0.2909n})$.

— *niveau* 2 : On fusionne les listes du niveau 3 avec une contrainte de modulo $c_2 = 0.5315$, en ne conservant que les éléments de $D^n[1/8, 0.0302]$. Un vecteur sera bien formé avec probabilité $\tilde{\mathcal{O}}\left(2^{p_2 n}\right)$ où $p_2 = -0.0120$, et on obtiendra quatre listes de taille $\tilde{\mathcal{O}}\left(2^{\ell_2 n}\right)$ où $\ell_2 = 2\ell_3 - (c_2 - c_3) + p_2 \simeq 0.2789$. Cependant, il faut également payer le coût de construction des éléments mal formés. La taille de l'ensemble des éléments mal formés est $\tilde{\ell_2} = 2\ell_3 - (c_2 - c_3) \simeq 0.2909$. Le coût de cette étape est donc $\tilde{\mathcal{O}}(2^{0.2909n})$.

— *niveau* 1 : On utilise une contrainte de modulo $c_1 = 0.7984$, et on cherche des vecteurs dans $D^n[1/4, 0.0267]$. On crée ainsi deux listes de taille $\tilde{\mathcal{O}}\left(2^{\ell_1 n}\right)$ où $\ell_1 = 2\ell_2 - (c_1 - c_2) + p_1 \simeq 0.2165$. Cette étape demande un temps $\tilde{\mathcal{O}}(2^{\tilde{\ell_1} n}) = \tilde{\mathcal{O}}(2^{0.2909n})$.

— *niveau* 0 : On fusionne les deux listes du niveau précédant. On obtient une liste de taille polynomiale (*i.e.* $\ell_0 = 0$). Cela nécessite cependant

de construire les $\tilde{\mathcal{O}}\left(2^{\tilde{\ell_0}n}\right) = \tilde{\mathcal{O}}\left(2^{(\ell_0-p_0)n}\right) \simeq 2^{0.2330n}$ éléments mal formés.

La complexité de cet algorithme est donc $\tilde{\mathcal{O}}(2^{0.2909n})$, ce qui correspond à la complexité des niveaux 1, 2 et 3.

1.4 Algorithmes quantiques

Même pour des algorithmes quantiques, nous ne savons pas résoudre le problème de remplissage de sac à dos plus rapidement qu'en temps exponentiel. Il est cependant possible d'obtenir des accélérations notables par rapport aux algorithmes classiques.

Avec des ressources quantiques, la meilleure chose à faire pour accélérer les algorithmes de résolution du problème de sac à dos est (de ce qu'on sait) l'utilisation de marches quantiques. Ainsi, Bernstein, Jeffery, Lange and Meurer ont publié dans [BJLM13] un algorithme quantique qui s'inspire de l'algorithme classique de [HJ10], et le combine avec une marche quantique. Cela leur permet d'obtenir un algorithme dont la complexité est $\tilde{\mathcal{O}}\left(2^{0.241n}\right)$. Plus tard, Helm et May ont présenté ([HM18]) un algorithme en $\tilde{\mathcal{O}}\left(2^{0.226n}\right)$, basé, lui, sur l'algorithme de [BCJ11].

1.4.1 Un algorithme quantique simple

Pour illustrer comment associer les marches quantiques et les algorithmes classiques de remplissage de sac à dos, nous nous servirons de l'algorithme simple décrit dans 1.3.3. Le but est de construire une marche quantique suivant de formalisme de [MNRS11] rappelé en 0.2.5.

Dans l'algorithme de 1.3.3, le facteur limitant est la taille des listes du niveau 2. Pour obtenir une complexité plus faible, il est nécessaire d'utiliser des listes plus petites. Nous distinguerons alors ℓ_i^c, l'exposant de la taille des listes dans l'algorithme classique de 1.3.3, et ℓ_i^q, qui lui servira pour la taille des listes dans l'algorithme quantique que nous allons décrire.

La structure de données utilisée dans l'algorithme quantique est la même que celle de l'algorithme classique : un arbre à trois niveaux. Dans l'algorithme classique, les listes du niveau 2 étaient construites en énumérant tous les vecteurs de taille $n/2$ et de densité $1/4$. Il y a $\tilde{\mathcal{O}}\left(2^{\ell_2^c n}\right)$ tels vecteurs, ce qui est trop. Dans l'algorithme quantique, on se contente d'échantillonner cet ensemble, en choisissant $\tilde{\mathcal{O}}\left(2^{\ell_2^q n}\right)$ de ses éléments, où $\ell_2^q = \ell_2^c - x$ pour un x qui reste à déterminer.

En ce qui concerne les autres niveaux, on conserve les même contraintes modulaires. Par conséquent, les listes seront de taille $\tilde{\mathcal{O}}\left(2^{\ell_1^q n}\right) = \tilde{\mathcal{O}}\left(2^{(\ell_1^c-2x)n}\right)$ au niveau 1, et $\tilde{\mathcal{O}}\left(2^{\ell_0^q n}\right) = \tilde{\mathcal{O}}\left(2^{(\ell_0^c-4x)n}\right)$ au niveau 0.

On peut maintenant décrire le graphe de Johnson sur lequel on va marcher :

Les sommets Au niveau 2, chaque liste est produite en sélectionnant $\tilde{\mathcal{O}}\left(2^{\ell_2^q n}\right)$ éléments parmi un ensemble de cardinal $\tilde{\mathcal{O}}\left(2^{\ell_2^c n}\right)$. Chaque choix possibles pour ce quadruplet de listes est un sommet du graphe.

Les arêtes Deux sommets sont voisins si on peut passer de l'un à l'autre en changeant un seul élément dans l'une des listes du niveau 2 (les éventuelles différences aux niveaux 0 et 1 n'interviennent pas dans cette définition).

Nous reconnaissons ici le produit cartésien de quatre graphes de Johnson (un pour chaque liste du niveau 2), $J^4(2^{\ell_2^c n}, 2^{\ell_2^q n})$. Le théorème 0.2 permet de minorer le trou spectral de ce graphe par :

$$\delta \geqslant \frac{1}{4}\frac{2^{\ell_2^c n}}{2^{\ell_2^q n}(2^{\ell_2^c n} - 2^{\ell_2^q n})} = \tilde{\mathcal{O}}\left(2^{-\ell_2^q n}\right) = \tilde{\mathcal{O}}\left(2^{(-\ell_2^c + x)n}\right)$$

Sommets marqués L'espérance de la taille de la liste du niveau 0 est $\tilde{\mathcal{O}}\left(2^{(\ell_0^c - 4x)n}\right) = \tilde{\mathcal{O}}\left(2^{4xn}\right)$. Cette liste est donc vide pour la plupart des sommets. Cependant, pour quelques rares sommets (1 sur $\tilde{\mathcal{O}}\left(2^{4xn}\right)$), cette liste contient un élément, c'est-à-dire une solution au problème. Ce sont les sommets marqués. Nous avons au passage remarqué que la proportion des sommets marqués est :

$$\varepsilon = \tilde{\mathcal{O}}\left(2^{-4xn}\right)$$

1.4.2 Analyse de la complexité

Nous connaissons maintenant le graphe qui supportera la marche quantique, il reste à décrire les fonctions de Setup, Update et Check.

Setup : il s'agit de choisir un sommet du graphe, et de construire la structure de données associée. La complexité d'une telle fonction se calcule exactement comme on calculait la complexité des algorithmes classiques. Le facteur limitant est la taille des listes du niveau 2, à savoir :

$$\mathsf{S} = \tilde{\mathcal{O}}\left(2^{\ell_2^q n}\right) = \tilde{\mathcal{O}}\left(2^{(\ell_2^c - x)n}\right)$$

Update : on veut, à partir d'un sommet et de la structure de données associée, calculer un voisin ainsi que la structure de données de ce voisin. Pour cela, on choisit une liste du niveau 2. Dans cette liste, on remplace l'un des $2^{\ell_2^q n}$ élément par l'un des $2^{\ell_2^c n} - 2^{\ell_2^q n}$ vecteurs qui n'avait pas été

choisi. Il faut alors mettre à jour le reste de la structure de données. Un vecteur du niveau 2 est impliqué en moyenne dans $\tilde{\mathcal{O}}\left(2^{(\ell_1^q-\ell_2^q)n}\right)$ vecteurs du niveau 1. Un vecteur du niveau 1 est à son tour impliqué en moyenne dans $\tilde{\mathcal{O}}\left(2^{(\ell_0^q-\ell_1^q)n}\right)$ vecteurs du niveau 0. Puisque $\ell_1^q - \ell_2^q < 0$ et $\ell_0^q - \ell_1^q < 0$, l'UPDATE demande en moyenne un temps :

$$\mathsf{U} = \tilde{\mathcal{O}}(1)$$

CHECK : cela consiste à vérifier si un sommet est marqué. Il suffit de regarder si la liste du niveau 0 est non vide. Cela demande un temps :

$$\mathsf{C} = \tilde{\mathcal{O}}(1)$$

Comme annoncé dans 0.2.5, la complexité de la marche quantique est alors :

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

Dans le cas présent, cela donne $\tilde{\mathcal{O}}\left(2^{(\ell_2^c-x)n}\right) + \tilde{\mathcal{O}}\left(2^{(\ell_2^c+3x)n/2}\right)$, ce qui est minimisé pour $x = \ell_2^c/5$. On obtient alors finalement pour complexité :

$$\tilde{\mathcal{O}}\left(2^{0.324}\right)$$

Ce qui est mieux que la complexité de l'algorithme classique dont on s'est inspiré, $\tilde{\mathcal{O}}\left(2^{0.405n}\right)$.

Naturellement, de meilleurs algorithmes classiques au départ permettent de construire des algorithmes quantiques plus efficaces.

1.4.3 Complexité de l'UPDATE

Dans le calcul de complexité précédent, nous avons triché en nous contentant du temps moyen de la fonction d'UPDATE.

Considérons les niveaux i et $i-1$. En moyenne, chaque élément d'une liste du niveau i intervient dans la création de $\tilde{\mathcal{O}}\left(2^{\ell_{i-1}-\ell_i}\right)$ éléments du niveau $i-1$. Ainsi, si $\ell_{i-1} > \ell_i$, l'ajout ou la suppression d'un seul élément du niveau i entraîne la modification d'un nombre exponentiel d'éléments au niveau $i-1$. Pour éviter de faire exploser le coût de l'étape d'UPDATE, on imposera la condition suivante : si une liste est susceptible d'être modifiée par l'UPDATE, elle doit être au moins aussi grande que les listes des niveaux supérieurs.

Cependant, même dans le cas où les tailles des listes sont bien choisies, il peut y avoir des cascades exponentielles. C'est ce qui arrive si, par hasard, les éléments d'une des listes ne sont pas bien répartis. Ces cas sont extrêmement rares (ce qui fait que l'heuristique 1 fonctionne en pratique). Malheureusement, dans le cas d'une marche quantique, le but est d'appliquer l'UPDATE à la superposition de tous les états, y compris les états problématiques. Nous allons ignorer ce problème et utiliser :

Heuristique 2. *Dans les cas considérés, l'*Update *demande un temps moyen* U*. On suppose qu'on peut construire des marches quantiques comme s'il s'agissait d'un temps exact* U*, au prix d'un sur-coût polynomial.*

Remarque 1.6. Cette heuristique a été introduite par [HM18]. Elle avait auparavant été utilisée implicitement [BJLM13, KT17].

Chapitre 2

Optimisation du remplissage de sac à dos

Dans le chapitre précédent, nous avons présenté le problème du remplissage de sac à dos, et décrit des travaux et techniques permettant de résoudre ce problème de plus en plus efficacement. Ainsi, en partant d'un algorithme naïf, et en ajoutant successivement des collisions, des représentations, des -1, nous avons abouti à une complexité $\tilde{\mathcal{O}}\left(2^{0.2909n}\right)$.

Dans le chapitre présent, nous allons voir comment aller plus loin, et améliorer ces résultats. Pour cela, nous allons décrire deux points qu'il est possible d'optimiser pour obtenir une meilleure complexité. Le premier point est une compréhension différente de la technique des représentations, qui permet de s'affranchir de certaines contraintes superflues. Le deuxième point est l'extension des représentations, en ajoutant un symbole supplémentaire : les 2. À l'aide de ces deux changements, il nous sera possible de décrire un algorithme dont la complexité asymptotique est $\tilde{\mathcal{O}}\left(2^{0.2830n}\right)$.

Après cela, nous allons réaliser un travail similaire dans le cas des algorithmes quantiques. Les techniques présentées dans le chapitre précédent permettent d'obtenir une complexité $\tilde{\mathcal{O}}\left(2^{0.226n}\right)$. En plus de profiter des changements apportés au cas classique, nous allons donner deux façons d'améliorer les algorithmes de résolution du problème de sac à dos à l'aide de marches quantiques. D'une part, nous allons constater que certaines opérations peuvent être accélérées à l'aide d'une amplification d'amplitude. D'autre part, nous allons montrer comment introduire de l'asymétrie au dernier étage pour concilier les besoins de l'Update et ceux du Setup. Cela permettra d'aboutir à une complexité $\tilde{\mathcal{O}}\left(2^{0.2156n}\right)$.

Ces résultats sont contenus dans [BBSS20].

2.1 Doublons, filtrage et saturation

Dans les algorithmes utilisant des représentations ([HJ10], [BCJ11]), les représentations sont toujours équilibrées de la même façon. Le nombre de représentations d'un vecteur est systématiquement l'inverse de la contrainte modulaire de l'étage du dessous. De cette façon, en moyenne une représentation de ce vecteur survit à cette contrainte modulaire.

Prenons comme exemple l'algorithme de [BCJ11], dont les paramètres sont rappelés en 1.3.5. Dans cet algorithme, une liste du niveau 2 est obtenue en fusionnant deux listes du niveau 3 à l'aide de représentations : un vecteur de $D^n[1/8, 0.0302]$ est recherché comme la somme de deux vecteurs de $D^n[1/16, 0.0180]$. Chaque vecteur de cette liste du niveau 2 admet 0.2407 décompositions. Or, toutes ces décompositions ne sont pas calculées : seules sont calculées celles qui passent la contrainte de modulo arbitraire c_3. Or il se trouve que $c_3 = 0.2407$: en moyenne une décomposition passe la contrainte de modulo. Ainsi, au niveau 2, chaque vecteur qui vérifie la contrainte de distribution et la contrainte de modulo va effectivement être trouvé, car l'une de ses décompositions passe la contrainte de modulo c_3 ; on ne va pas non plus créer de doublons : la contrainte c_3 compensant exactement le nombre de décompositions.

Cette décision est parfaitement raisonnable :

— Si on conserve, en moyenne, un nombre exponentiel de représentations pour chaque vecteur : on perd beaucoup de temps et de mémoire en calculant, de différentes façons, de multiples copies des mêmes éléments (voir 1.3.4).

— Si on conserve en moyenne moins d'une représentation par vecteur, la probabilité d'avoir un vecteur particulier est exponentiellement faible. Il faut alors répéter l'algorithme en changeant les contraintes de modulo intermédiaires jusqu'à en trouver qui soient compatibles avec le vecteur problématique si on veut le trouver (comme c'était fait dans [SS81], où on ne disposait pas de représentations).

2.1.1 Relaxation du filtrage des représentations

Cependant, ce dernier point n'est pas un problème absolu. Si le vecteur dont on a trop filtré les représentations est la solution, il est nécessaire de répéter l'algorithme en changeant les contraintes de modulo, et le nombre de répétitions est définitivement perdu. Par contre, si le vecteur en question n'est qu'une représentation, il n'est pas désastreux de ne pas le trouver (c'est bien là l'intérêt des représentations, il n'y a pas besoin de toutes les trouver.)

Ainsi, il est possible d'être un peu plus subtil : quand on utilise des représentations sur plusieurs étages, on peut se servir de toutes les contraintes de modulo des étages inférieurs pour équilibrer le nombre de représentations,

au lieu de se forcer à n'utiliser que l'étage directement inférieur, comme illustré :

— figure 2.1 (haut) : C'est un équilibrage à la façon de [BCJ11]. La solution admet de nombreuses représentations (5 sont représentées sur cette figure), mais sur toutes ces représentations, une seule passe la contrainte de modulo c_1, et apparaîtra effectivement au niveau 1. (Les listes calculées au niveau 1 contiennent également de nombreux vecteurs qui ne sont pas des représentations de la solution, et qui n'ont pas été représentés sur cette figure.)

  Chaque vecteur du niveau 1 admet à son tour un certain nombre de représentations, dont une seule passe la contrainte de modulo c_2.

— figure 2.1 (bas) : la contrainte de filtrage du niveau 2 est plus grande que le nombre de représentations d'un vecteur du niveau 1 : certains de ces vecteurs ne seront pas trouvés car aucune de leurs représentations ne passe les contraintes de modulo.

  D'un autre côté, la solution admet bien plus de représentations que ce qui est filtré au niveau 1. Si on trouvait toutes ces représentations, on obtiendrait de multiples copies de la solution.

  Ces faits se compensent : on ne trouve pas tous les vecteurs du niveau 1 qui vérifient la contrainte de modulo c_1 et qui ont la bonne distribution. C'est un bonne chose car parmi ces vecteurs se cachent plusieurs représentations de la solution, et il serait inutile d'en trouver plus qu'une.

L'intérêt d'une telle opération, décaler l'équilibrage des représentations et des contraintes de modulo, n'est pas évident. Cependant, cela donne plus de liberté pour les paramètres de l'algorithme, et on constate que cela permet de légères améliorations. Par exemple, dans le cas de l'algorithme de [BCJ11], on peut améliorer la complexité de $\tilde{\mathcal{O}}\left(2^{0.291n}\right)$ à $\tilde{\mathcal{O}}\left(2^{0.289n}\right)$.

Remarque 2.1. Cela ne fonctionne que dans un sens. On peut se servir de c_2 pour filtrer une partie des représentations entre les niveaux 1 et 0 (alors que dans l'algorithme de [BCJ11], c_2 compense exactement les représentations entre les niveaux 2 et 1). Il n'est en revanche en aucun cas possible de se servir de c_1 pour filtrer des représentations entre les niveaux 2 et 1.

Autrement dit, le niveau 1 peut contenir moins de vecteurs que ce qui est autorisé par l'entropie de la distribution et la contrainte de modulo, mais ne peut sûrement pas en contenir plus. C'est ce que nous allons utiliser par la suite.

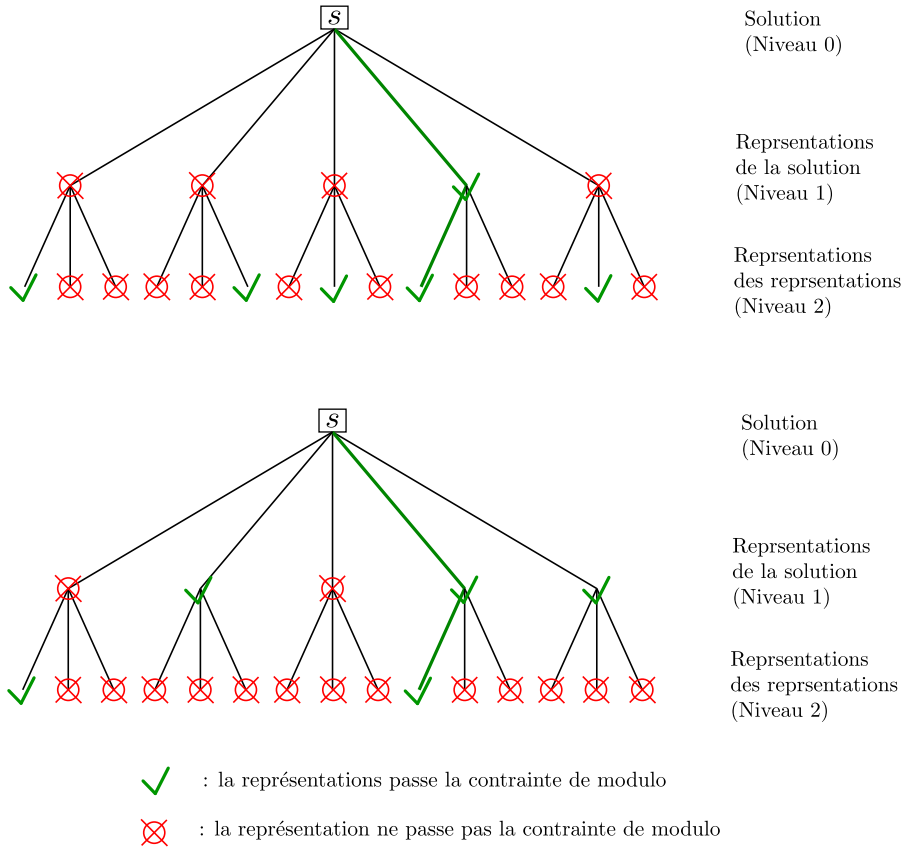

FIGURE 2.1 – En haut : on garde exactement une représentation à chaque étage. En bas : on impose une contrainte de modulo plus importante à l'étage inférieur, ce qui permet de filtrer moins à l'étage supérieur.

2.1.2 Contrôler les doublons par l'entropie

Le nombre de vecteurs de longueur n et de distribution D^n est $\tilde{\mathcal{O}}\left(2^{\tilde{h}(D)n}\right)$. Nous allons utiliser cette quantité pour majorer les tailles des listes. Considérons une liste, qui contient $2^{\ell n}$ vecteurs, tous de distribution D^n, et qui vérifient une contrainte modulaire $M \simeq 2^{cn}$. Le nombre de tous les vecteurs de distribution D^n est $\tilde{\mathcal{O}}\left(2^{\tilde{h}(D)n}\right)$. Le nombre de tous les vecteurs de distribution D^n qui vérifient la contrainte de modulo est, sous l'heuristique 1, $\tilde{\mathcal{O}}\left(2^{(\tilde{h}(D)-c)n}\right)$. Si la liste est plus grande que cela, elle contient nécessairement des doublons. Ainsi, dans un algorithme de résolution de sac à dos utilisant la structure d'arbre de 1.2.4, il est nécessaire que chaque liste vérifie $\ell \leqslant \tilde{h}(D) - c$ pour ne pas être submergé par les doublons. Nous voulons montrer que cette condition est suffisante.

Théorème 2.1. *Sous l'heuristique 1, une condition nécessaire et suffisante*

pour ne pas produire de doublons est : pour toute liste de taille ℓ et de distribution D, on a $\ell \leqslant \tilde{h}(D) - c$.

Démonstration. Si, pour une liste, on a $\ell > \tilde{h}(D) - c$, cette liste contient plus de vecteurs que le nombre de vecteurs possibles. Cette liste contient alors des doublons, chaque vecteur étant présent en moyenne $\tilde{\mathcal{O}}\left(2^{(\ell-\tilde{h}(D)-c)n}\right)$ fois (avec $\ell - \tilde{h}(D) - c > 0$).

Dans le cas contraire, on raisonne par récurrence en partant du niveau le plus bas, en montrant qu'à chaque niveau, les vecteurs sont uniformément distribués et ne présentent pas de doublons :

— au niveau le plus bas, les listes sont produites par énumération de tous les vecteurs de la distribution. Les vecteurs sont donc immédiatement uniformément distribués et les listes ne contiennent pas de doublons.

— au niveau j, on suppose que les listes du niveau $j+1$ ne contiennent pas de doublons et que leurs vecteurs sont uniformément distribués parmi les vecteurs qui ont la bonne distribution et qui vérifient la contrainte de modulo. On peut alors utiliser l'heuristique 1, selon laquelle les 2^ℓ vecteurs produits sont uniformément distribués, dans un ensemble de cardinal $\tilde{\mathcal{O}}\left(2^{(\tilde{h}(D)-c)n}\right) > 2^{\ell n}$. Il n'y a donc pas de doublons.

□

Remarque 2.2. Si on force partout l'égalité $\ell = \tilde{h}(D) - c$, on retrouve le choix d'équilibrage de [BCJ11].

Démonstration. On se place à un niveau i. Les vecteurs du niveau i ont pour distribution D_i, et vérifient la contrainte de modulo c_i. Ces vecteurs sont obtenus à l'aide de représentations, les vecteurs de niveau inférieur, le niveau $i+1$, ayant pour distribution D_{i+1} et vérifiant c_{i+1}. Par hypothèse, les tailles des listes sont :

$$\begin{aligned} \ell_i &= \tilde{h}(D_i) - c_i \\ \ell_{i+1} &= \tilde{h}(D_{i+1}) - c_{i+1} \end{aligned}$$

Par ailleurs, en notant p_i la probabilité qu'une collision soit un vecteur bien formé, les tailles de liste vérifient :

$$\ell_i = 2\ell_{i+1} - (c_i - c_{i+1}) + p_i$$

D'autre part, il y a $2\tilde{h}(D_{i+1})$ choix de couples de vecteur de distribution D_{i+1}. Parmi ceux-ci, seuls $2\tilde{h}(D_{i+1}) - p_i$ sont bien formés et constituent des représentations. Ces représentations se répartissant également entre tous les vecteurs de distribution D_i, chacun de ces vecteurs admet $Rep_{D_{i+1},D_{i+1}}$ représentations, où :

$$\log_2\left(Rep_{D_{i+1},D_{i+1}}\right) = 2\tilde{h}(D_{i+1}) - p_i - \tilde{h}(D_i)$$

En remplaçant $\tilde{h}(D_{i+1})$, $\tilde{h}(D_i)$ et p_i, l'expression précédente se simplifie en :

$$\log_2\left(Rep_{D_{i+1},D_{i+1}}\right) = c_{i+1}$$

La contrainte modulaire compense exactement le nombre de représentations. □

2.2 Plus de symboles ?

L'ajout des -1, dans l'algorithme de [BCJ11], permet de diminuer la complexité de la résolution de problème du sac à dos de $\tilde{\mathcal{O}}(2^{0.337n})$ à $\tilde{\mathcal{O}}(2^{0.291n})$. Il a-t-il une raison fondamentale pour s'arrêter là, alors qu'on pourrait par exemple ajouter des 2 ?

À vrai dire, il y a même deux raisons à cela. D'une part, l'optimisation des paramètres devient brutalement très pénible, principalement à cause de l'augmentation du nombre de variables. D'autre part, le gain apporté par l'ajout d'un symbole supplémentaire a de fortes chances d'être très faible.

Cela dit, étant donné que nous avons déjà une (faible) amélioration grâce à une meilleure optimisation des paramètres, autant essayer de pousser jusqu'au bout. Nous allons donc ajouter des 2 dans les représentations. Pour cela, la première étape est de trouver un moyen de calculer la probabilité qu'un élément soit bien formé lorsqu'on utilise des représentations.

2.2.1 Cas typique

Pour compter efficacement les nombres de décompositions et de vecteurs mal formés, il nous suffit de calculer le *cas typique*. Commençons par un exemple simple : considérons le cas des $(D^n[0,1/3], D^n[0,1/3], D^n[0,1/3])-$représentations. Nous avons un vecteur de taille n composé d'autant de 0, de 1 et de -1, et nous cherchons à le décomposer en deux vecteurs de distribution similaire. Il y a plusieurs possibilités :

- Les -1 sont produits par $(-1)+(0)$ (c'est-à-dire : le premier vecteur donne un -1 et le second un 0), les 0 sont produits par $(1)+(-1)$ et les 1 par $(0)+(1)$. Il n'y a qu'une seule telle décomposition.
- le cas contraire : les -1 sont produit par $(0)+(-1)$, les 0 par $(-1)+(1)$ et les 1 par $(1)+(0)$. Il n'y a également qu'une seule décomposition de ce type.
- le cas médian : les cas $(-1)+(0)$, $(0)+(-1)$, $(-1)+(1)$, $(1)+(-1)$, $(0)+(1)$ et $(1)+(0)$ arrivent chacun $n/6$ fois. Il y a $\binom{n/3}{n/6}^3 = \tilde{\mathcal{O}}(2^n)$ telles décompositions.
- tous les autres cas intermédiaires.

Le nombre total de façons de faire cette décomposition est la somme de tous ces cas. Or le nombre de cas est un polynôme en n (car les nombres de $(0)+(1)$, etc... sont des entiers). Par ailleurs, certains cas sont exponentiellement plus représentés que d'autres. Pour connaître l'exposant du nombre total de décompositions, il suffit donc d'avoir l'exposant du nombre de décompositions dans le cas le plus représenté, qu'on appellera le *cas typique*.

2.2.2 Symétrie du cas typique

Plus nous utilisons de symboles, plus le cas typique devient difficile à calculer, en raison de l'explosion du nombre de variables qui interviennent. Pour rendre cette cette tâche plus simple, nous commencerons par montrer que, sous certaines hypothèses, il suffit de chercher le cas typique parmi les décompositions symétriques.

Théorème 2.2 (Symétrie du cas typique)**.** *Dans le cas de $(D, D_1, D_1)-$représentations, avec D et D_1 choisies de façon compatibles, le cas typique est symétrique.*

Par "symétrique", on entend que le symbole 1 est obtenu aussi souvent comme $(0)+(1)$ que comme $(1)+(0)$, etc....

"Compatibles" signifie qu'il est possible de décomposer un vecteur de D en la somme de deux vecteurs de D_1.

On se limite aux distributions qui contiennent un nombre fini de symboles.

Démonstration. Pour chaque symbole i, on note α_i (*resp.* β_i) la proportion des coordonnées occupées par ce symbole dans un vecteur de D (*resp.* D_1).

On considère deux vecteurs $\mathbf{e_1}$ et $\mathbf{e_2}$, de distribution D_1, dont la somme est un certain $\mathbf{e}$ de distribution D. Pour deux symboles i et j, on note $x_{i,j}$ la proportion du vecteur sur laquelle $\mathbf{e_1}$ contient un i et $\mathbf{e_2}$ un j ($\mathbf{e}$ contient alors le symbole $i+j$).

Le cas typique est donné par les valeurs des $x_{i,j}$ qui maximisent la quantité $\tilde{h}((x_{i,j})) = -\sum_{i,j} x_{i,j} \log_2(x_{i,j})$.

Les $x_{i,j}$ vérifient un certain nombre de contraintes d'égalité linéaires, à savoir :

$$\begin{array}{lll}
\forall i & \sum_j x_{i,j} = \beta_i & \text{(provient de } \mathbf{e_1} \in D_1 \text{)} \\
\forall j & \sum_i x_{i,j} = \beta_j & \text{(provient de } \mathbf{e_2} \in D_1 \text{)} \\
\forall k & \sum_{i+j=k} x_{i,j} = \alpha_k & \text{(provient de } \mathbf{e} \in D \text{)} \\
 & \sum_{i,j} x_{i,j} = 1 & \text{(chaque case de } \mathbf{e}, \mathbf{e_1} \text{ et } \mathbf{e_2} \text{ contient quelque chose)}
\end{array}$$

L'ensemble des familles $(x_{i,j})$ qui vérifient ces égalités forme un sous-espace affine.

D'autre part, les $x_{i,j}$ correspondent à des proportions d'un vecteur. Pour chaque $x_{i,j}$, on doit donc se trouver dans le demi-espace $x_{i,j} \geqslant 0$ (on a également les contraintes $x_{i,j} \leqslant 1$, mais celle-ci sont redondantes.)

On appelle $\mathcal{A}$ l'intersection de ces demi-espaces et du sous-espace affine. On remarque alors que :

— $\mathcal{A}$ est non vide. C'est la traduction de l'hypothèse selon laquelle les distributions D et D_1 sont compatibles.

— $\mathcal{A}$ est inclus dans l'hypercube défini par $\forall(i,j), 0 \leqslant x_{i,j} \leqslant 1$. En particulier, $\mathcal{A}$ est borné.

— $\mathcal{A}$ étant défini comme une intersection de fermés, $\mathcal{A}$ est en particulier fermé.

— $\mathcal{A}$ est défini comme une intersection de convexes, et est donc lui-même convexe.

— $\mathcal{A}$ est fermé et borné en dimension fini, et est donc compact.

La fonction $(x_{i,j}) \mapsto \tilde{h}((x_{i,j}))$ est une fonction continue définie sur un compact, et admet donc un maximum (le cas typique).

On considère maintenant $(x_{i,j})$, un point quelconque de $\mathcal{A}$. Par symétrie des rôles joués par $\mathbf{e_1}$ et $\mathbf{e_2}$, le point $(x_{j,i})$ est également dans $\mathcal{A}$. Par convexité de $\mathcal{A}$, le point $((x_{i,j}) + (x_{j,i}))/2$ appartient encore à $\mathcal{A}$. Par symétrie de la fonction entropie $\tilde{h}$, on a : $\tilde{h}((x_{i,j})) = \tilde{h}((x_{j,i}))$. Par ailleurs, on remarque, par convexité de la fonction $x \mapsto x \log_2(x)$, que :

$$\forall(i,j), -\frac{x_{i,j}+x_{j,i}}{2}\log_2\left(\frac{x_{i,j}+x_{j,i}}{2}\right) \geqslant \frac{1}{2}\left(-x_{i,j}\log_2(x_{i,j}) - x_{j,i}\log_2(x_{j,i})\right)$$

d'où :

$$\tilde{h}\left(\frac{(x_{i,j})+(x_{j,i})}{2}\right) \geqslant \frac{1}{2}(\tilde{h}((x_{i,j})) + \tilde{h}((x_{j,i}))) = \tilde{h}((x_{i,j}))$$

Quelque soit le point choisi, il y aura toujours un cas symétrique pour faire au moins aussi bien. □

2.2.3 Les représentations dans {-1, 0, 1, 2}

Comme annoncé, nous allons à présent étendre les représentations en ajoutant des 2. Pour cela, nous commençons par adapter la notion de distribution :

Définition 2.1 (Distribution)**.** *On note $D^n[s,\beta,\gamma]$ l'ensemble des vecteurs de taille n, dont la somme des coefficients est $\lfloor sn \rfloor$, dont $\lfloor \beta n \rfloor$ coordonnées sont des -1, et $\lfloor \gamma n \rfloor$ sont des 2.*

Remarque 2.3. Il s'agit encore d'une généralisation, la notation $D^n[s, \beta]$ précédente correspondant au cas où $\gamma = 0$.

Pour analyser correctement l'impact des représentations contenant des 2, il nous faut pouvoir calculer le nombre d'éléments mal formés produits :

Théorème 2.3. *Quand on fusionne deux listes en utilisant des représentations de paramètres* $(D^n[2s_1, \beta, \gamma], D^n[s_1, \beta_1, \gamma_1], D^n[s_1, \beta_1, \gamma_1])$*. La probabilité qu'un vecteur obtenu comme collision soit bien formé est* $\tilde{\mathcal{O}}\left(2^{pn}\right)$*, où :*

$$p = \max_{x \in [x_{\min}, x_{\max}]} \tilde{h}(x_1, x_1, x_2, x_2, x_3, x_4, x_4, x_5, x_5, x_6, x_6, x_7) \\ - 2\tilde{h}(\beta_1, \gamma_1, s_1 + \beta_1 - 2\gamma_1, 1 - s_1 - 2\beta_1 + \gamma_1)$$

avec :

$$\begin{cases} x_1 = \beta/2 \\ x_2 = \beta_1 - x - \beta/2 \\ x_3 = 1 - \beta + \gamma - 2\beta_1 - 2s_1 + 2x \\ x_4 = x \\ x_5 = \beta/2 + s_1 - \gamma - x \\ x_6 = \gamma_1 - x \\ x_7 = \gamma - 2\gamma_1 + 2x \\ \\ x_{\min} = \max\{\beta/2 + \beta_1 - s_1 - \gamma/2 - 1/2, 0, \gamma_1 - \gamma/2\} \\ x_{\max} = \min\{\beta_1 - \beta/2, \beta/2 + s_1 - \gamma, \gamma_1\} \end{cases}$$

et le maximum est atteint par l'unique x *qui vérifie :*

$$x_2 x_5 x_6 = x_3 x_4 x_7$$

Démonstration. Nous cherchons à déterminer le *cas typique* de la décomposition d'un vecteur $\mathbf{e} \in D^n[2s_1, \beta, \gamma]$ en la somme de deux vecteurs $\mathbf{e_1}$ et $\mathbf{e_2} \in D^n[s_1, \beta_1, \gamma_1]$.

Tout d'abord, nous sommes dans un cas où les deux distributions des vecteurs à sommer sont identiques. Le théorème 2.2 s'applique donc. Dans le cas typique, autant de -1 de $\mathbf{e}$ seront produits comme la somme d'un -1 de $\mathbf{e_1}$ et d'un 0 de $\mathbf{e_2}$ que comme la somme d'un 0 de $\mathbf{e_1}$ et d'un -1 de $\mathbf{e_2}$, etc...

Notons :

— x_1 le nombre de -1 de $\mathbf{e}$ produits comme $(-1) + (0)$

— x_2 le nombre de 0 de $\mathbf{e}$ produits comme $(-1) + (1)$

— x_3 le nombre de 0 de $\mathbf{e}$ produits comme $(0) + (0)$

— x_4 le nombre de 1 de $\mathbf{e}$ produits comme $(-1) + (2)$

— x_5 le nombre de 1 de $\mathbf{e}$ produits comme $(1)+(0)$

— x_6 le nombre de 2 de $\mathbf{e}$ produits comme $(2)+(0)$

— x_7 le nombre de 2 de $\mathbf{e}$ produits comme $(1)+(1)$

Les symétriques des cas présentés sont implicites. Ces variables doivent vérifier :

$$\begin{cases} 2s_1 = (2x_4 + 2x_5) - (2x_1) + 2(2x_6 + x_7) \\ \beta = 2x_1 \\ \gamma = 2x_6 + x_7 \end{cases}$$

par définition de $\mathbf{e} \in D^n[2s_1, \beta, \gamma]$;

$$\begin{cases} s_1 = (x_2 + x_5 + x_7) - (x_1 + x_2 + x_4) + 2(x_4 + x_6) \\ \beta_1 = x_1 + x_2 + x_4 \\ \gamma_1 = x_4 + x_6 \end{cases}$$

par définition de $\mathbf{e_1} \in D^n[s_1, \beta_1, \gamma_1]$;

et enfin $2x_1 + 2x_2 + x_3 + 2x_4 + 2x_5 + 2x_6 + x_7 = 1$ pour l'exhaustivité.

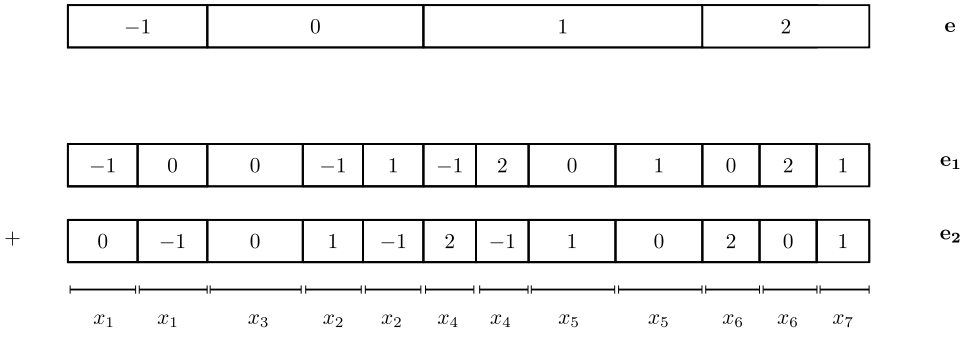

FIGURE 2.2 – Décomposition de $\mathbf{e}$ en $\mathbf{e_1} + \mathbf{e_2}$

Ces relations forment un système linéaire redondant : il y a un degré de liberté. Nous pouvons donc exprimer toutes ces variables en fonction d'un paramètre libre, par exemple x_4, qu'on renommera x. Nous avons ainsi :

$$\begin{cases} x_1 = \beta/2 \\ x_2 = \beta_1 - x - \beta/2 \\ x_3 = 1 - \beta + \gamma - 2\beta_1 - 2s_1 + 2x \\ x_4 = x \\ x_5 = \beta/2 + s_1 - \gamma - x \\ x_6 = \gamma_1 - x \\ x_7 = \gamma - 2\gamma_1 + 2x \end{cases}$$

Le nombre de façons de choisir deux vecteurs $\mathbf{e_1}$ et $\mathbf{e_2}$ dans $D^n[s_1, \beta_1, \gamma_1]$ est $\tilde{\mathcal{O}}\left(2^{2\tilde{h}(\beta_1, \gamma_1, s_1+\beta_1-2\gamma_1, 1-s_1-2\beta_1+\gamma_1)n}\right)$.

Le nombre de façon de choisir deux vecteurs $\mathbf{e_1}$ et $\mathbf{e_2}$ dont la somme est un vecteur de $D^n[2s_1, \beta, \gamma]$ et pour lesquels la somme $(-1) + (2)$ apparaît xn fois est $\tilde{\mathcal{O}}\left(2^{\tilde{h}(x_1,x_1,x_2,x_2,x_3,x_4,x_4,x_5,x_5,x_6,x_6,x_7)n}\right)$. Le cas typique correspond au maximum de cette expression quand x parcours les valeurs autorisées. Comme les variables x_1 à x_7 doivent être positives, x est contraint dans $[x_{\min}, x_{\max}]$, où :

$$\begin{cases} x_{\min} = \max\{\beta/2 + \beta_1 - s_1 - \gamma/2 - 1/2, 0, \gamma_1 - \gamma/2\} \\ x_{\max} = \min\{\beta_1 - \beta/2, \beta/2 + s_1 - \gamma, \gamma_1\} \end{cases}$$

La probabilité que la somme de deux vecteurs de $D^n[s_1, \beta_1, \gamma_1]$ soit dans $D^n[2s_1, \beta, \gamma]$ est donc $\tilde{\mathcal{O}}\left(2^{pn}\right)$ avec :

$$\begin{aligned} p = \max_{x \in [x_{\min}, x_{\max}]} & \tilde{h}(\beta/2, \beta/2, \beta_1 - x - \beta/2, \beta_1 - x - \beta/2, 1 - \beta + \gamma - 2\beta_1 - 2s_1 + 2x, \\ & x, x, \beta/2 + s_1 - \gamma - x, \beta/2 + s_1 - \gamma - x, \gamma_1 - x, \gamma_1 - x, \gamma - 2\gamma_1 + 2x) \\ & - 2\tilde{h}(\beta_1, \gamma_1, s_1 + \beta_1 - 2\gamma_1, 1 - s_1 - 2\beta_1 + \gamma_1) \end{aligned}$$

Nous cherchons à trouver le x qui maximise cette expression. Sa dérivée est :

$$\log_2\left(\left(\frac{(\beta + 2x - 2\beta_1)(2s_1 + \beta - 2x - 2\gamma + \beta)(\gamma_1 - x)}{4x(\beta + 2s_1 + 2\beta_1 - 2x - \gamma - 1)(\gamma - 2\gamma_1 + 2x)}\right)^2\right)$$

c'est-à-dire :

$$2\log_2\left(\frac{x_2 x_5 x_6}{x_4 x_3 x_7}\right)$$

La fonction à maximiser étant strictement concave, le maximum est atteint soit sur un bord de l'intervalle, soit en un point d'annulation de la dérivée. Le x du cas typique vérifie donc :

— soit $x = x_{\min}$

— soit $x = x_{\max}$

— soit $x_2 x_5 x_6 = x_3 x_4 x_7$

La dérivée étant $+\infty$ en $x_{\min}$, et $-\infty$ et $x_{\max}$, le maximum ne peut pas être atteint sur un bord de l'intervalle.

Le x recherché est donc la seule solution de l'équation d'annulation de la dérivée, $x_2 x_5 x_6 = x_3 x_4 x_7$, dans l'intervalle $[x_{\min}, x_{\max}]$. Il s'agit d'une équation polynomiale de degré 3. Cependant, la fonction à maximiser étant concave, on sait qu'il n'y a pas d'autre solutions dans cet intervalle.

□

Remarque 2.4. Il est possible de calculer explicitement les solutions de l'équation $x_2x_5x_6 = x_3x_4x_7$, puisque ce n'est qu'une équation polynomiale de degré 3. Cependant, les formes explicites sont monstrueuses. Par ailleurs, comme $\tilde{h}(x_1, x_1, x_2, x_2, x_3, x_4, x_4, x_5, x_5, x_6, x_6, x_7)$ est une fonction concave et infiniment dérivable de x, il est en pratique plus simple et bien plus rapide de chercher le maximum de cette fonction à l'aide d'une technique d'optimisation, par exemple la méthode de Newton.

2.2.4 Encore plus de symboles ?

En rajoutant des 2, nous avons réussi à diminuer encore la complexité de la résolution du problème du sac à dos. Cependant, le gain obtenu en ajoutant les 2 est déjà faible, en comparaison de ce qui avait été obtenu avec les -1. Ajouter encore des symboles supplémentaires (-2, 3, ...) permettrait très certainement d'améliorer ces algorithmes, mais il est également fort possible que le gain se perde dans la précision des optimisations numériques. Par ailleurs, les calculs deviendraient encore plus pénibles, et les optimisations des paramètres beaucoup plus instables.

Nous notons au passage que, puisqu'il faut trouver des solutions dans $\{0, 1\}$, le problème ne présente pas de symétrie par rapport à 0. Il n'y a aucune raison pour laquelle l'ajout des -2 donnerait un gain similaire à celui apporté par les 2.

2.2.5 Complexité

Avec ces deux nouvelles techniques, nous pouvons décrire un algorithme dont la complexité est $\tilde{\mathcal{O}}\left(2^{0.2830n}\right)$, avec les paramètres suivants :

— *niveau* 4 : on construit 16 listes. Pour les listes impaires, on énumère les éléments de $D^{n/2}[1/16, 0.0202, 0.0001] \times 0^{n/2}$, et pour les listes paires : $0^{n/2} \times D^{n/2}[1/16, 0.0202, 0.0001]$. Ces listes sont de taille $\ell_4 = 0.2673$.

— *niveau* 3 : on fusionne les listes du niveau 4 deux par deux, avec une contrainte de modulo $c_3 = 0.2680$. On obtient alors 8 listes de distribution $D^n[1/16, 0.0202, 0.0001]$, et de taille $\ell_3 = 0.2830$.

— *niveau* 2 : on utilise des représentations pour construire 4 listes de distribution $D^n[1/8, 0.0311, 0.0006]$. La contrainte de modulo utilisée est $c_2 = 0.5509$, et les listes obtenues sont de taille $\ell_2 = 0.2694$. Cependant, la complexité du calcul de ces listes est $\tilde{\mathcal{O}}\left(2^{0.2830n}\right)$ à cause des vecteurs mal formés.

— *niveau* 1 : on construit 2 listes de distribution $D^n[1/4, 0.0340, 0.0041]$ avec une contrainte de modulo $c_1 = 0.8067$. Ces listes sont de taille $\ell_1 = 0.2382$ et requièrent un temps de calcul $\tilde{\mathcal{O}}\left(2^{0.2830n}\right)$.

— *niveau* 0 : on recherche une collision entre les deux listes du niveau 1.

De façon amusante, il y a une synergie entre les deux améliorations que nous venons de décrire. En effet, le gain obtenu en combinant la relaxation des contraintes de filtrage et l'ajout des 2 est plus grand que la somme des gains obtenus avec chacune de ces améliorations seule.

	$\{-1,0,1\}$	$\{-1,0,1,2\}$
contraintes strictes	[BCJ11] $\tilde{\mathcal{O}}(2^{0.2909n})$	$\tilde{\mathcal{O}}(2^{0.287n})$
contraintes relaxées	$\tilde{\mathcal{O}}(2^{0.289n})$	[BBSS20] $\tilde{\mathcal{O}}(2^{0.2830n})$

FIGURE 2.3 – Complexité asymptotique des algorithmes de sac à dos en fonction des différentes techniques utilisées

2.3 Améliorations quantiques

L'algorithme de [HM18] est une adaptation quantique de l'algorithme de [BCJ11]. Il est donc naturel de chercher à adapter au cas quantique le travail réalisé dans le cas classique (non-saturation des entropies, ajout des 2 dans les représentations). Cela ne présente pas de difficultés particulières.

Cependant, il et possible d'aller plus loin, en ajoutant certaines optimisations spécifiques au cas quantique.

2.3.1 Filtrage quantique

Comme nous l'avons déjà constaté, l'utilisation de représentations lors de la fusion de listes produit des éléments mal formés. Le calcul de ces éléments mal formés constitue une perte de temps, que nous aimerions limiter. Nous allons voir qu'il est possible de réduire le coût de ces éléments mal formés lors de l'utilisation d'algorithmes quantiques, tant dans le SETUP que dans l'UPDATE.

Pour fixer les idées, nous nous plaçons dans le cas de la fusions de deux listes L_1 et L_2 d'un niveau $i+1$ en une liste L du niveau i. L_1 et L_2 sont de taille $\tilde{\mathcal{O}}\left(2^{\ell_{i+1}n}\right)$, tandis que L est de taille $\tilde{\mathcal{O}}\left(2^{\ell_i n}\right)$. Les éléments des listes du niveau $i+1$ vérifient une contrainte modulaire sur $c_{i+1}n$ bits, et ceux du niveau i sur $c_i n$ bits.

La fusion de L_1 et L_2 produit $\tilde{\mathcal{O}}\left(2^{\tilde{\ell}_i n}\right)$ collisions, où $\tilde{\ell}_i = 2\ell_{i+1} - (c_i - c_{i+1})$. Chacune de ces collisions est bien formée avec probabilité $\tilde{\mathcal{O}}\left(2^{p_i n}\right)$. On a alors $\ell_i = \tilde{\ell}_i + p_i$.

Pour éviter les cas d'UPDATE exponentiellement long, nous nous plaçons dans le cas où $\ell_{i-1} \geqslant \ell_i$ (voir 1.4.3.)

Dans de Setup

Dans l'étape de Setup, il faut construire L à partir de L_1 et L_2. Dans le cas classique, nous faisions cela est calculant toutes les $\tilde{\mathcal{O}}\left(2^{\tilde{\ell}_i n}\right)$ collisions, et en sélectionnant les éléments bien formés.

Dans le cas d'un algorithme quantique, nous pouvons utiliser une amplification d'amplitude. Trouver $\tilde{\mathcal{O}}\left(2^{\ell_i n}\right)$ éléments bien formés parmi un ensemble de cardinal $\tilde{\mathcal{O}}\left(2^{\tilde{\ell}_i n}\right)$ peut se faire en temps $\tilde{\mathcal{O}}\left(2^{\frac{\tilde{\ell}_i+\ell_i}{2} n}\right)$.

Dans l'Update

Lors de l'Update, il faut répercuter au niveau i les éventuelles modifications du niveau $i+1$. Par exemple, on ajoute un élément x dans L_1 et il faut modifier L pour conserver la cohérence. Puisque $\ell_{i+1} \geq \ell_i$, il faut ajouter, en moyenne, au plus un élément z dans L, qui sera une collision de x avec un certain élément y déjà présent dans L_2.

Or x, comme tous les autres éléments de L_1, intervient en moyenne dans $\tilde{\mathcal{O}}\left(2^{(\ell_{i+1}-(c_i-c_{i+1}))n}\right)$ collisions. Dans le cas classique, nous énumérions toutes ces collisions.

Dans le cas quantique, nous pouvons accélérer cette recherche grâce à une amplification d'amplitude. Cela permet de trouver cet éventuel z en un temps $\tilde{\mathcal{O}}\left(2^{\frac{\ell_{i+1}-(c_i-c_{i+1})}{2} n}\right)$.

2.3.2 Dernier étage asymétrique

Pour simplifier les explications, nous allons nous placer dans le cas d'un algorithme à 6 étages (numérotés de 0 à 5), c'est le nombre d'étages de l'algorithme qui sera présenté dans la section suivante. Comme nous l'avons vu dans la partie 1.4.3, l'update demande des listes de plus en plus grandes au fur et à mesure qu'on descend dans l'arbre. En particulier, les listes de l'étage 5, le dernier étage, sont très grandes et il leur faut donc une grande contrainte de modulo. Or cela n'est pas bon pour la phase de Setup (plus les modulo sont bas dans l'arbre, plus ils génèrent de contraintes arbitraire, et plus on perd de solutions.) Pour concilier les étapes d'Update et de Setup, on rend asymétrique le dernier étage : chaque liste de l'étage 4 sera produite à partir d'une grande liste (la liste "gauche") et d'une petite liste ("droite"). Les listes de l'étage 4 sont de taille ℓ_4, et on notera ℓ_5^l et ℓ_5^r les tailles des listes "gauches" et "droites" à l'étage 5.

On ne marchera que sur les listes gauches (les grandes listes) et on choisira $\ell_5^l = \ell_4$. De cette façon, il n'y a pas de problème pour l'Update, un élément d'une liste gauche du niveau 5 intervenant en moyenne dans un seul élément au niveau 4. Les listes droites, pour leur part, seront fixes, et

contiennent l'énumération complète des vecteurs compatibles. Puisqu'à ce niveau on n'utilise pas de représentations, il n'y a pas d'éléments mal formés. La contrainte de modulo à utiliser est donc directement $c_4 = \ell_5^l + \ell_5^r - \ell_4 = \ell_5^r$. La contrainte de modulo est petite, comme ℓ_5^r, ce qui permet d'obtenir de bons paramètres pour le SETUP.

Il nous faut maintenant un moyen de produire des listes asymétriques. On rappelle qu'on n'utilise jamais de représentations au dernier étage, mais un découpage partie gauche/partie droite. Pour obtenir des listes déséquilibrées, il suffit d'ajuster le nombre de coordonnées allouées à chacune de ces deux parties. Si on prenait un découpage symétrique, on chercherait les partie gauches dans $D^{n/2}[1/32, \beta_4, \gamma_4] \times 0^{n/2}$, et les parties droites dans $0^{n/2} \times D^{n/2}[1/32, \beta_4, \gamma_4]$. Pour avoir un découpage asymétrique, on cherche les parties gauches dans $D^{\eta n}[1/32, \beta_4, \gamma_4] \times 0^{(1-\eta)n}$, tandis que pour les parties droites on énumère entièrement $0^{\eta n} \times D^{(1-\eta)n}[1/32, \beta_4, \gamma_4]$, pour un paramètre η à déterminer.

2.3.3 Complexité

Combinons maintenant toutes ces idées dans un seul algorithme. Le meilleur algorithme obtenu utilise un arbre contenant 6 niveaux.

SETUP : Commençons par décrire l'étape de SETUP, c'est-à-dire la construction de l'arbre complet de toutes les listes.

— au niveau 5, il y a 32 listes. Puisque le dernier étage est asymétrique, on distingue les listes paires, qui seront des listes gauches, et les listes impaires, qui seront des listes droites. Pour les listes droites, on énumère entièrement $0^{0.9288n} \times D^{0.0712n}[1/32, 0.0060]$. Ces listes sont donc de taille $\ell_5^r = 0.0201$. Les listes gauches, pour leur part, sont constituées de $\ell_5^l = 0.2145$ vecteurs tirés aléatoirement dans $D^{0.9288n}[1/32, 0.0060] \times 0^{0.0712n}$.

— pour le niveau 4, chacune des 16 listes est obtenue par la fusion d'une liste gauche et d'une liste droite du niveau 5, avec une contrainte de modulo $c_4 = 0.0201$. On obtient ainsi 16 listes de taille $\ell_4 = 0.2145$, et contenant des éléments de $D^n[1/32, 0.0060]$.

— au niveau 3, on construit les 8 listes en fusionnant les listes du niveau 4 deux par deux, avec une contrainte de modulo $c_3 = 0.2325$. On a ici des représentations, et on impose aux vecteurs de ce niveau d'être dans $D^n[1/16, 0.01159]$. La probabilité pour un vecteur de vérifier cette distribution est $p_3 = -0.0021$. Les listes obtenues sont alors de taille $\ell_3 = 2\ell_4 - (c_3 - c_4) + p_3 = 0.2145$. Cependant, le coût de la construction de ces listes est plus élevé en raison des éléments mal formés : l'exposant est $\ell_3 - p_3/2$, ce qui donne $\tilde{\mathcal{O}}\left(2^{0.2156n}\right)$.

— au niveau 2, on veut des vecteurs de distribution $D^n[1/8, 0.0145]$. La probabilité d'obtenir cette distribution est $p_2 = -0.0202$. Puisqu'on utilise une condition de modulo 0.4358, on obtient des listes de taille $\ell_2 = 0.2055$. Le coût de construction de ces listes est $\tilde{\mathcal{O}}\left(2^{0.2156n}\right)$.

— au niveau 1, on choisit une condition de modulo $c_1 = 0.6154$, une distribution $D^n[1/4, 0.0167, 0.0018]$. Cela donne une probabilité $p_1 = -0.0317$, et des listes de taille $\ell_1 = 0.1997$. Ces listes coûtent également $\tilde{\mathcal{O}}\left(2^{0.2156}\right)$ à construire.

— au niveau 0, on veut (on n'a pas le choix) des vecteurs de distribution $D^n[1/2]$ et qui satisfont à la condition de modulo $c_0 = 1$. La probabilité pour un vecteur d'être bien formé est $p_0 = -0.2056$, ce qui donne une liste dont l'espérance de la taille est $\ell_0 = -0.1907$.

Le coût de l'étape de Setup est donc :

$$\mathsf{S} = \tilde{\mathcal{O}}\left(2^{0.2156n}\right)$$

Remarque 2.5. On n'a mis de 2 qu'au niveau 1. Il est possible d'en mettre également aux niveaux inférieurs, mais le gain apporté est négligeable.

Update : On considère à présent l'étape d'Update :

— Au niveau 5. On choisit l'une des listes gauches. Un s'agit d'un sous-ensemble strict de $D^{0.9288n}[1/32, 0.0060] \times 0^{0.0712n}$. On remplace l'un des éléments de la liste par un élément de $D^{0.9288n}[1/32, 0.0060] \times 0^{0.0712n}$ qui ne fait pas partie de la liste. Cette opération se fait en temps $\tilde{\mathcal{O}}(1)$.

— Au niveau 4. Puisqu'on a modifié le niveau 5, il faut faire quelques modifications au niveau 4 pour conserver un arbre cohérent. Le nouvel élément de la liste du niveau 5 donne en moyenne une collision, il suffit de l'ajouter. Le temps requis par cette opération est $\tilde{\mathcal{O}}(1)$.

— Au niveau 3, il y a des représentations. L'élément ajouté au niveau 4 va donner en moyenne $\ell_4 - (c_3 - c_4)$ collisions. Cependant, parmi celles-ci, en moyenne une seule sera bien formée. Il faut donc trouver cette collision bien formée, ce qui se fait à l'aide d'une amplification d'amplitude. Le temps de cette opération est $\tilde{\mathcal{O}}\left(2^{0.0011n}\right)$.

— Au niveau 2, on réalise la même opération qu'au niveau 3, ce qui demande un temps $\tilde{\mathcal{O}}\left(2^{0.0066n}\right)$.

— Le niveau 1 est similaire, et demande un temps $\tilde{\mathcal{O}}\left(2^{0.0142n}\right)$.

— Pour le niveau 0, il est peu probable que l'élément ajouté au niveau 1 (s'il y en a un) donne une collision. Si cela se produit, on la trouve en temps $\tilde{\mathcal{O}}(1)$.

Finalement, le coût de l'étape d'Update est :

$$\mathsf{U} = \tilde{\mathcal{O}}\left(2^{0.0142n}\right)$$

Remarque 2.6. Au niveau 5, on a remplacé un élément par un autre. Pour les niveaux supérieurs, on a uniquement décrit comment faire les modifications induites par l'ajout. Les modifications à effectuer pour tenir compte de la suppression demandent le même temps.

Check : L'étape de Check consiste à vérifier si la liste du niveau 0 est non vide. Cette étape demande donc un temps :

$$\mathsf{C} = \tilde{\mathcal{O}}(1)$$

Proportion des sommets marqués : Le paramètre ε est la probabilité qu'un choix des listes du niveau 5 donne une solution. Ce paramètre est directement donné par la probabilité que la liste du niveau 0 soit non vide, c'est-à-dire ℓ_0. Ainsi :

$$\varepsilon = \tilde{\mathcal{O}}\left(2^{-0.1907n}\right)$$

Trou spectral : Enfin, le paramètre δ est donné par la taille des listes sur lesquelles on marche, à savoir les listes gauches du niveau 5. On a donc :

$$\delta = \tilde{\mathcal{O}}\left(2^{-0.2145n}\right)$$

Complexité : La complexité de cet algorithme est donc $\tilde{\mathcal{O}}\left(2^{0.2156n}\right)$.

[BJLM13]	$\tilde{\mathcal{O}}\left(2^{0.241n}\right)$
[HM18]	$\tilde{\mathcal{O}}\left(2^{0.226n}\right)$
[BBSS20]	$\tilde{\mathcal{O}}\left(2^{0.2156n}\right)$

Figure 2.4 – Complexité de différents algorithmes quantiques de résolution du problème du sac à dos

Deuxième partie

Le décodage de syndrome

Chapitre 3

Décodage

Dans ce chapitre, nous allons examiner un problème voisin du remplissage de sac à dos : le problème du décodage de syndrome. Le problème du décodage de syndrome est hérité de la théorie de l'information et des canaux de communication bruités. Dans [McE78], McEliece a détourné ce problème de son domaine original pour en dériver un protocole cryptographique.

Le problème du décodage de syndrome est difficile et a été prouvé NP-complet par [BMvT78]. De ce fait, un pan de la cryptographie se base sur les codes correcteurs, et utilise la difficulté du décodage de syndrome comme hypothèse calculatoire. C'est par exemple le cas, outre [McE78], de [Ste93, Nie86]. Par ailleurs, le décodage n'a pas montré de signe particulier de faiblesse face à d'éventuelles attaques quantiques. Cela en fait un candidat pour concevoir des algorithmes cryptographiques résistant aux ordinateurs quantiques. C'est notamment le cas de [ACP+17, AMAB+17a, AMAB+17b, BBC+19, BCL+17], proposés au processus de standardisation du NIST.

Comme dans le cas du remplissage de sac à dos, tous les algorithmes (classiques et quantiques) présentés auront une complexité exponentielle en la taille du problème, complexité qu'on cherchera à minimiser.

3.1 Le problème de décodage

Le cas le plus étudié, et de loin, est le cas du décodage binaire. C'est par là que nous allons commencer.

Problème : Décodage binaire

Entrée :	$\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$ de rang $(n-k)$	matrice de parité
	$\mathbf{s} \in \mathbb{F}_2^{n-k}$	syndrome
	$\omega \in [0, 1/2]$	contrainte de poids

Sortie : $\mathbf{e} \in \mathbb{F}_2^n$ tel que $\begin{cases} \mathbf{He} = \mathbf{s} \\ |\mathbf{e}| = W \text{ où } W = \lceil \omega n \rceil \end{cases}$

Ce problème est très similaire au sac à dos. Il présente néanmoins plusieurs différences notables :

— Le vecteur à renvoyer prend maintenant ses valeurs dans $\mathbb{F}_2$, et non plus dans $\{0, 1\}$ vu comme sous-ensemble de $\mathbb{Z}$. Comme $\mathbb{F}_2$ est stable par addition, cela permet de faire des opérations de type pivot de Gauss. Dans le cas de $\{0, 1\}$, essayer de réaliser ce genre d'opérations donnerait un vecteur dont les éléments sont dans $\mathbb{Z}$, et qu'on ne saurait pas ramener à $\{0, 1\}$.

— Pour le problème de décodage, on se donne une contrainte additionnelle, sur le poids de la solution.

Remarque 3.1. Dans la littérature on trouvera souvent une convention différente, avec une contrainte énoncée $\mathbf{eH}^T = \mathbf{s}$, ou de façon équivalente $\mathbf{He}^T = \mathbf{s}^T$. Ici et dans toute la suite, nous utiliserons des vecteurs colonnes et n'auront pas recours aux transposées.

Remarque 3.2. Dans la définition du décodage de syndrome, nous avons pris pour convention $W = \lceil \omega n \rceil$. On aurait pu choisir $\lfloor \omega n \rfloor$, ou encore l'arrondi à l'entier le plus proche, cela ne change en rien le comportement asymptotique.

Remarque 3.3 (Symétrie petit/gros poids). On note $\mathbb{1}_n$ le vecteur de taille n qui ne contient que des 1. On remarque qu'un vecteur $\mathbf{e}$ est une solution au problème de décodage pour les entrées $(\mathbf{H}, \mathbf{s}, \omega)$ si et seulement si $\mathbf{e} + \mathbb{1}_n$ est une solution pour les entrées $(\mathbf{H}, \mathbf{s} + \mathbf{H}\mathbb{1}_n, 1 - \omega)$, avec la convention $W = \lfloor \omega n \rfloor$. Puisque le choix de convention est sans importance (remarque précédente), la complexité du problème de décodage est la même pour les paramètres ω et $1 - \omega$. Pour cette raison, on se restreindra à $\omega \in [0, 1/2]$.

3.1.1 Distance de Gilbert-Varshamov

En fonction des paramètres R et ω, différents cas de figure peuvent se présenter. Il peut y avoir un nombre exponentiel de solutions, ou au contraire l'existence d'une solution peut être très improbable. Plus précisément, nous avons :

Lemme 3.1. *À ω et R fixés, si $\mathbf{H}$ et $\mathbf{s}$ sont tirés aléatoirement, l'espérance du nombre de solutions au problème de décodage est :*

$$\tilde{\mathcal{O}}\left(2^{nh(\omega)-(n-k)}\right)$$

Démonstration. Il faut trouver un vecteur $\mathbf{e}$ de taille n et de poids $\lceil \omega n \rceil$. Cela offre donc $\binom{n}{\lceil \omega n \rceil} \sim 2^{nh(\omega)}$ choix possibles pour $\mathbf{e}$. Or il faut également vérifier la condition $\mathbf{He} = \mathbf{s}$. Comme $\mathbf{s}$ est de taille $n-k$, seulement une fraction $2^{-(n-k)}$ des choix possibles pour $\mathbf{e}$ va donner le bon $\mathbf{s}$. □

On distingue alors plusieurs cas, selon les valeurs relatives de $nh(\omega)$ et $(n-k)$:

- $nh(\omega) > n-k$: le nombre de solutions est exponentiel, et trouver une seule de ces solutions n'est pas si difficile.
- $nh(\omega) < n-k$: l'espérance du nombre de solutions est exponentiellement faible. Cependant, s'il y en a une, cette solution est tellement contrainte qu'elle est facile à trouver (toutes les contraintes agissent comme autant d'indices qui aident à trouver la solution).
- $nh(\omega) = n-k$: il s'agit du cas limite. L'espérance du nombre de solutions est polynomial en n. C'est le cas le plus difficile. Le poids ω_{GV} tel que $nh(\omega_G V) = n-k$ est le *poids de Gilbert-Varshamov*[1] associée à R.

3.1.2 Le problème de décodage en cryptographie

Le problème de décodage, défini précédemment, n'admet pas nécessairement de solution. Le problème de décodage utilisé en cryptographie est légèrement différent :

Problème : Décodage de syndrome binaire (version cryptographique)

Entrée :	$\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$ de rang $(n-k)$	matrice de parité
	$\mathbf{s} \in \mathbb{F}_2^{n-k}$	syndrome
	$\omega \in [0, 1/2]$	contrainte de poids

où $\mathbf{s} = \mathbf{He_0}$ pour un $\mathbf{e_0}$ de poids $W := \lceil \omega n \rceil$, gardé secret

Sortie : $\mathbf{e} \in \mathbb{F}_2^n$ tel que $\begin{cases} \mathbf{He} = \mathbf{s} \\ |\mathbf{e}| = W \text{ où } W := \lceil \omega n \rceil \end{cases}$

1. Ce poids (ou distance) de Gilbert-Varshamov est bien connu en théorie des codes. Au delà de cette distance, l'espérance du nombre de messages correspondant à un syndrome est supérieure à 1. Retrouver le message original est alors impossible.

Dans le problème tel que nous l'avons défini, nous nous donnons la garantie qu'il existe toujours une solution : le $\mathbf{e_0}$ utilisé pour générer le syndrome. Il peut y avoir d'autres solutions (si on se situe au delà du poids de Gilbert-Varshamov), il n'est alors pas demandé de retrouver ce $\mathbf{e_0}$ spécifiquement.

3.1.3 Complexité du décodage

Nous allons étudier la complexité asymptotique du problème de décodage cryptographique, en fonction de la contrainte de poids ω et du ratio $R := k/n$. Plus précisément, nous considérons la complexité en moyenne sur tous les choix possibles pour $\mathbf{H}$ et $\mathbf{e_0}$. La complexité d'un algorithme de résolution du problème de décodage peut alors se mettre sous la forme $\tilde{\mathcal{O}}(2^{\alpha_{R,\omega} n})$.

Il est à noter que, contrairement au problème de décodage général, il n'a pas été montré que le problème de décodage en moyenne soit NP-complet. Cependant, il n'existe pas actuellement d'algorithme efficace (même quantique) pour résoudre ce problème.

3.1.4 Décodage dans un corps $\mathbb{F}_q$

Si le problème de décodage dans le cas binaire est de loin le plus courant, il est possible de le transcrire pour des corps finis de caractéristique quelconque.

Problème : Décodage q-aire

Entrée :	$\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$	matrice de parité
	$\mathbf{s} \in \mathbb{F}_q^{n-k}$	syndrome
	$\omega \in [0,1]$	contrainte de poids

où $\mathbf{s} = \mathbf{He_0}$ pour un $\mathbf{e_0}$ de poids $W := \lceil \omega n \rceil$, gardé secret

Sortie : $\mathbf{e} \in \mathbb{F}_q^n$ tel que $\begin{cases} \mathbf{He} = \mathbf{s} \\ |\mathbf{e}| = W \end{cases}$

La notion de poids est généralisée de la façon suivante : il s'agit du nombre de symboles non nuls dans un vecteur. Par exemple, dans le cas ternaire, les symboles 2 ne comptent pas plus que les 1 : le poids d'un vecteur $\mathbf{e}$ est la somme du nombre de 1 et du nombre de 2 dans ce vecteur.

Par la suite nous nous concentrerons particulièrement sur le cas ternaire ($q = 3$), mais nous pouvons commencer par des résultats plus généraux. Dans un premier temps, déterminons l'espérance du nombre de solutions :

Lemme 3.2. *À ω et k fixés, si $\mathbf{H}$ et $\mathbf{s}$ sont tirés aléatoirement, l'espérance du nombre de solutions au problème de décodage dans $\mathbb{F}_q$ est :*

$$\tilde{\mathcal{O}}\left(2^{\omega n \log_2(q-1) + nh(\omega) - (n-k)\log_2(q)}\right)$$

Démonstration. Il s'agit d'une généralisation du lemme 3.1 au cas q-aire. Nous devons trouver un vecteur $\mathbf{e}$ de taille n et de poids ωn. Il y a $(q-1)^{\omega n}\binom{n}{\omega n}$ tels vecteurs (il y a $\binom{n}{\omega n}$ choix pour la position des symboles non nuls, puis $q-1$ choix pour la valeur de chacun de ces symboles). Cependant, pour être une solution, il faut également vérifier $\mathbf{He} = \mathbf{s}$. Or seule une fraction $q^{-(n-k)}$ des vecteurs satisfont cette condition. □

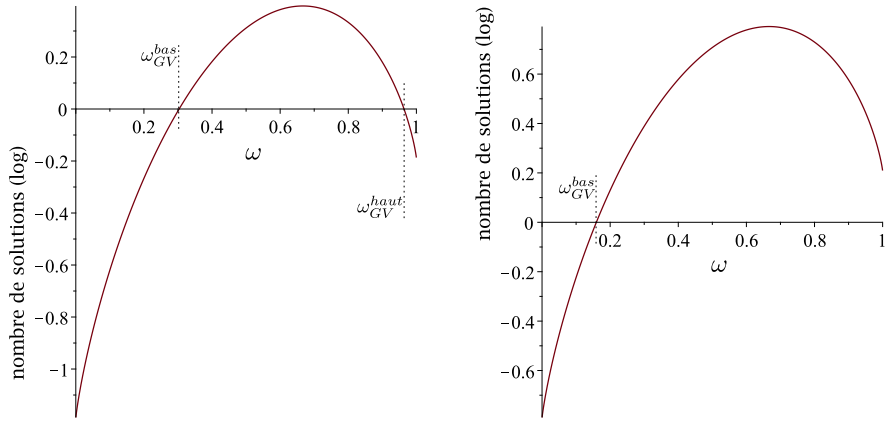

FIGURE 3.1 – À gauche : espérance du nombre de solutions (normalisé par n) en fonction du poids ω, pour $q = 3$ et $R = k/n = 0.25$ – À droite, avec $R = 0.5$.

De même que dans le cas binaire, pour des très petites valeurs de ω, il n'y aurait probablement aucune solution sans la promesse. Pour des valeurs plus grandes de ω, l'espérance du nombre de solutions augmente et devient exponentiel en n. Entre deux, il existe une valeur limite pour laquelle l'espérance du nombre de solutions est polynomial : quand ω est égal à la distance de Gilbert-Varshamov, ω_{GV}^{bas}. Enfin, pour de très grandes valeurs de ω, le nombre de solutions diminue à nouveau. On constate alors l'existence de deux régimes. Soit l'espérance du nombre de solutions repasse en dessous de 1, il y a dans ce cas une autre valeur critique ω_{GV}^{haut} (par exemple figure 3.1 gauche). Sinon, l'espérance du nombre de solutions peut diminuer sans atteindre 1, et ω_{GV}^{haut} n'existe pas (figure 3.1 droite).

Remarque 3.4. Dans le cas binaire, ω_{GV}^{haut} existe toujours et vaut $\omega_{GV}^{haut} = 1 - \omega_{GV}^{bas}$. Cependant, cette valeur est très peu intéressante. En effet, le cas binaire est symétrique, comme nous l'avons remarqué dans 3.3, et le comportement pour des grandes valeurs de ω est identique aux petites valeurs. Pour $q > 2$, ces comportements deviennent distincts, et ω_{GV}^{haut} prend de l'importance.

3.2 Décodage par ensemble d'information

Le décodage par ensemble d'information provient de l'algorithme proposé par Prange dans [Pra62]. Cette technique a inspiré de nombreux algorithmes ([Ste88, LB88, Dum91, MMT11, BJMM12]). Le décodage par ensemble d'information forme ainsi une large majorité des algorithmes de décodage.

3.2.1 Ensemble d'information

Nous commencerons par décrire l'algorithme de Prange dans le cas binaire.

Algorithme de Prange (résumé)

1. Choisir aléatoirement un ensemble d'information
2. Faire une élimination gaussienne
3. Regarder le poids du résultat
   (a) si ce poids est W, il s'agit d'une solution
   (b) sinon, recommencer à l'étape 1

Si on oublie (temporairement) la contrainte de poids, le problème de décodage revient à trouver un antécédent au syndrome par la matrice de parité, c'est-à-dire résoudre un système linéaire. Puisque le syndrome est un vecteur de taille $n-k$, alors que le vecteur d'erreur à trouver est de taille n, on dispose de k degrés de liberté excédentaires. On peut donc fixer arbitrairement k bits du vecteur $\mathbf{e}$, puis résoudre un système linéaire à $n-k$ variables et $n-k$ inconnues. Un tel choix de k coordonnées est appelé un *ensemble d'information*.

Remarque 3.5. Une fois fixé l'ensemble d'information, on obtient un système linéaire à $n-k$ variables et autant d'inconnues. Ce système n'est pas nécessairement inversible. S'il ne l'est pas, on recommence avec un autre ensemble d'information. La probabilité d'obtenir un système inversible étant minorée par $\prod_{i=1}^{\infty}\left(1-\frac{1}{2^i}\right) > \frac{1}{4}$ indépendamment de n, cela n'ajoute qu'un facteur constant à la complexité. On supposera donc par la suite que les systèmes linéaires obtenus sont inversibles.

Il faut maintenant s'occuper de la contrainte de poids ω. C'est très simple : on espère qu'elle est vérifiée. Si la contrainte de poids n'est pas satisfaite, on recommence en choisissant un autre ensemble d'information.

Par symétrie de problème du décodage binaire (3.3), on se contentera des poids faibles ($\omega \leqslant 0.5$). On distinguera deux cas :

— $\omega n \leqslant \frac{n-k}{2}$: on cherche un vecteur de poids très faible. Dans ce cas, le mieux est de remplir l'ensemble d'information avec des 0.

— $\frac{n-k}{2} < \omega n \leqslant n/2$: on cherche un vecteur de poids modérément faible. Dans ce cas, on met dans l'ensemble d'information un poids $\omega n - \frac{n-k}{2}$.

Proposition 3.3. *La complexité de l'algorithme de Prange est :*

$$\begin{cases} \tilde{\mathcal{O}}\left(2^{\min(nh(\omega),n-k)-(n-k)h(\omega n/(n-k))}\right) & si\ \omega n \leqslant \frac{n-k}{2} \\ polynomiale & si\ \omega n \geqslant \frac{n-k}{2} \end{cases}$$

Démonstration. Une fois qu'un ensemble d'information $\mathcal{I}$ est fixé, toutes les opérations de permutation, de pivot de Gauss, de résolution de système linéaire et de calcul du poids ont un coût polynomial en n. L'exposant de la complexité est donc uniquement donné par l'espérance du nombre d'ensembles d'information à essayer. On distingue trois cas :

— $\omega \leqslant \omega_{GV}$: dans ce cas, il n'y a très probablement qu'une solution (le $\mathbf{e_0}$ qui a servi à générer le syndrome). On ne place que des 0 dans les k coordonnées de l'ensemble d'information. Un ensemble d'information va alors convenir si et seulement si on a choisi ses k éléments parmi les $(1-\omega)n$ coordonnées qui contiennent un 0 dans la solution $\mathbf{e_0}$. Il y a alors $\binom{(1-\omega)n}{k}$ ensembles d'information valides sur $\binom{n}{k}$ ensembles d'information possibles. La probabilité de choisir un bon ensemble d'information est donc $\binom{(1-\omega)n}{k} / \binom{n}{k}$, ce qui peut se réécrire $\binom{n-k}{\omega n} / \binom{n}{\omega n}$. Asymptotiquement, cela donne $\tilde{\mathcal{O}}\left(2^{(n-k)h(\omega n/(n-k))-nh(\omega)}\right)$.

— $\omega_{GV} \leqslant \omega < \frac{1-R}{2}$: il y a dans ce cas de nombreuses solutions. Les k coordonnées de l'ensemble d'information contiennent des 0, les $n-k$ autres (obtenues par la résolution du système linéaire) se comportent comme des variables aléatoires indépendantes. On espère donc que sur ces $n-k$ variables, ωn exactement soient des 1. Cela arrive avec probabilité $\binom{n-k}{\omega n} / 2^{n-k}$. La probabilité de choisir un bon ensemble d'information est dans ce cas $\tilde{\mathcal{O}}\left(2^{(n-k)h(\omega n/(n-k))-(n-k)}\right)$.

— $\frac{1-R}{2} \leqslant \omega \leqslant 0.5$: c'est le cas des poids modérément faibles. On a placé un poids $\omega n - \frac{n-k}{2}$. On espère donc, quand on résout le système linéaire, que $\frac{n-k}{2}$ variables soient des 1. Cela arrive avec une probabilité inverse polynomiale, et le nombre d'ensembles d'information à tester est polynomial.

L'espérance du nombre d'ensembles d'information à essayer avant de trouver une solution est :

$$\begin{cases} \tilde{\mathcal{O}}\left(2^{nh(\omega)-(n-k)h(\omega n/(n-k))}\right) & \text{si } \omega \leqslant \omega_{GV} \\ \tilde{\mathcal{O}}\left(2^{(n-k)-(n-k)h(\omega n/(n-k))}\right) & \text{si } \omega_{GV} \leqslant \omega < \frac{1-R}{2} \end{cases}$$

(Note : les deux expressions coïncident si $\omega = \omega_{GV}$.) On constate que l'expression générale $\tilde{\mathcal{O}}\left(2^{\min(nh(\omega),n-k)-(n-k)h(\omega n/(n-k))}\right)$ fonctionne dans ces deux cas. □

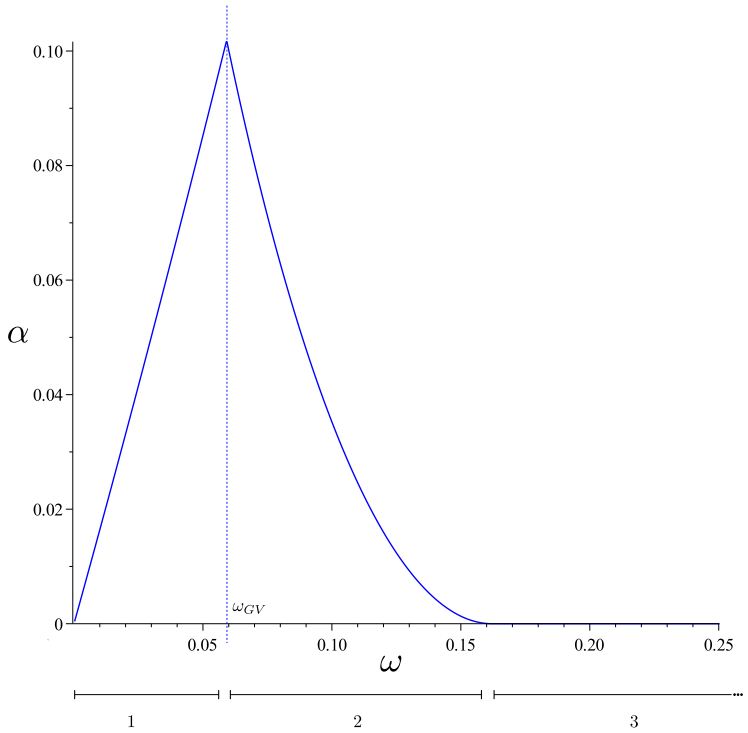

FIGURE 3.2 – Exposant de la complexité de l'algorithme de Prange pour résoudre le problème de décodage avec $R = 0.676$ (*i.e.* la complexité est $\tilde{\mathcal{O}}\left(2^{\alpha n}\right)$.)

La figure 3.2 montre les différents régimes qui peuvent se présenter :

1. $\omega \leqslant \omega_{GV}$, plus ω est petit, plus la solution (dont l'existence est promise) est contrainte, et plus elle est facile à trouver.
2. $\omega_{GV} \leqslant \omega \leqslant \frac{1-R}{2}$: plus ω est grand, plus il y a de solutions, et plus en trouver une est facile
3. $\frac{1-R}{2} \leqslant \omega$: il y a tellement de solutions qu'en trouver une ne demande qu'un temps polynomial.

Les paramètres les plus difficiles pour l'algorithme de Prange sont $k = 0.454n$ et $\omega = 0.12587$ (le ω_{GV} correspondant à k). L'exposant de complexité est dans ce cas $\alpha = 0.1207$.

3.2.2 Algorithme de Prange dans le cas q-aire

Dans le cas q-aire, l'algorithme de Prange fonctionne comme dans le cas binaire, il suffit juste d'adapter les calculs de dénombrement. En particulier, le problème de décodage ne présentant plus de symétrie entre les petits et les gros poids, il faut traiter les gros poids indépendamment. Il existe cinq régimes différents, illustrés sur la figure 3.3 :

1. $\omega \leqslant \omega_{GV}^{bas}$. Le poids est tellement faible qu'il n'y aurait très probablement aucune solution si $\mathbf{s}$ était choisi aléatoirement. Puisque le poids est très faible, on ne place que des 0 dans l'ensemble d'information. On retrouve alors la solution si on a choisi les k éléments de l'ensemble d'information parmi les $(1-\omega)n$ coordonnées de la solutions qui contiennent un 0. L'espérance du nombre de répétitions est $\tilde{\mathcal{O}}\left(2^{nh(\omega)-(n-k)h(\frac{\omega n}{n-k})}\right)$.
2. $\omega_{GV}^{bas} \leqslant \omega \leqslant \frac{(q-1)(1-R)}{q}$. Le problème admet beaucoup de solutions, mais le poids est encore très faible. On ne place encore que des 0 dans l'ensemble d'informations. On espère ensuite que la résolution du système linéaire donne un vecteur de bon poids. Dans ce cas, le nombre de répétitions est $\tilde{\mathcal{O}}\left(2^{(n-k)\log_2(q)-(n-k)h(\frac{\omega n}{n-k})-\omega n\log_2(q-1)}\right)$.
3. $\frac{(q-1)(1-R)}{q} \leqslant \omega \leqslant \frac{(q+R-1)}{q}$. C'est le cas facile : on place un poids $\omega - \frac{(q-1)(1-R)}{q}$ dans l'ensemble d'information. On espère que la résolution du système linéaire produise un vecteur équilibré. Cela ne demande qu'un nombre polynomial d'essais.
4. $\frac{(q+R-1)}{q} \leqslant \omega \leqslant \omega_{GV}^{haut}$. Ce cas est similaire au cas 2, si ce n'est qu'on place un poids maximal dans l'ensemble d'information. Il faut dans ce cas $\tilde{\mathcal{O}}\left(2^{(n-k)\log_2(q)-(n-k)h(\frac{\omega n-k}{n-k})-(\omega n-k)\log_2(q-1)}\right)$ répétitions.
5. $\omega_{GV}^{haut} \leqslant \omega$. Ce cas est similaire au cas 1 : on espère que les k éléments de l'ensemble d'information sont choisies parmi les coordonnées non nulles de la solutions. On espère aussi n'avoir fait aucune faute dans l'ensemble d'information. La complexité en temps est ici $\tilde{\mathcal{O}}\left(2^{k\log_2(q-1)+nh(k)-\omega nh(\frac{k}{\omega n})}\right)$.

Sur la figure 3.3, on constate que, à paramètres similaires, le cas $q = 3$ semble bien plus complexe que le cas $q = 2$, tout particulièrement pour ω proche de 1. Cependant, cette comparaison est un peu trompeuse, pour deux raisons.

D'une part, pour les mêmes valeurs de ω, k et n, le problème ternaire est fondamentalement plus gros que le cas binaire : chaque case de $\mathbf{H}$ ou de $\mathbf{s}$ et

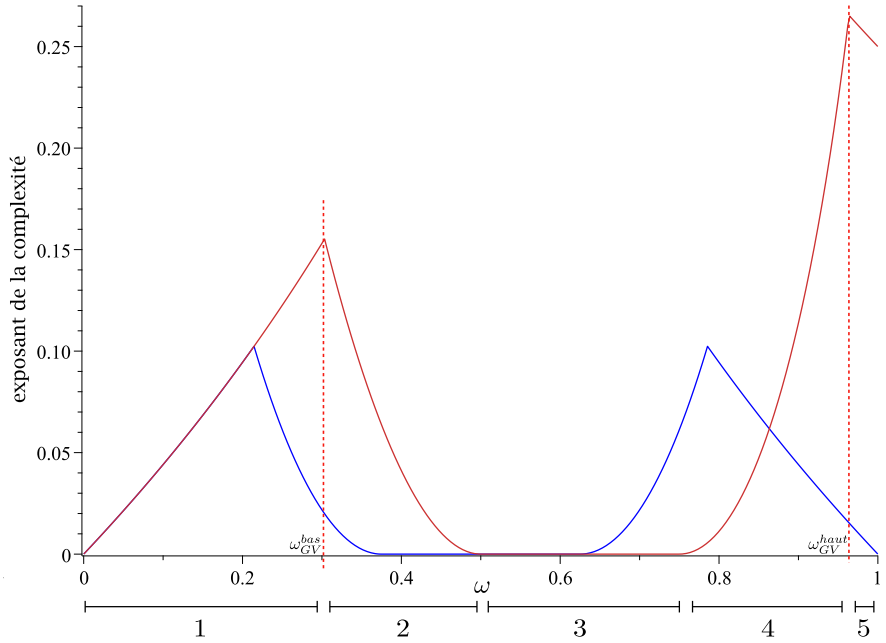

FIGURE 3.3 – exposant de la complexité de l'algorithme de Prange pour résoudre le problème de décodage avec $R = 0.25$, pour $q = 2$ (en bleu) et $q = 3$ (en rouge)

un élément de $\mathbb{F}_3$ au lieu de $\mathbb{F}_2$. Le problème est donc plus gros d'un facteur $\log_2(3)$.

D'autre part, les cas binaire et ternaire sont ici comparés sur l'algorithme de Prange. Si cet algorithme est raisonnable pour résoudre le décodage binaire, il est mal adapté pour de cas non-binaires : pour des gros poids, on sait qu'il faut remplir l'ensemble d'information avec des valeurs non nulles mais on ne sait pas lesquelles. Nous allons donc étudier des algorithmes plus complexes et plus efficaces.

3.2.3 Réduction au problème de sac à dos

Nous venons de décrire l'algorithme de Prange. Cet algorithme, grâce à l'utilisation de l'élimination de Gauss, est relativement efficace pour résoudre le problème du décodage. Ce type d'algorithmes était impossible à utiliser dans le cas du problème, similaire, du sac à dos. Cependant, rien n'interdit d'utiliser les techniques de remplissage de sac à dos dans le décodage de syndrome. Au contraire, la plupart des algorithmes de décodage combinent une élimination Gaussienne partielle et les techniques plus générales de remplissage de sac à dos. Nous appellerons cette structure PGE+SS (Partial Gaussian Elimination + Subset Sum), pour reprendre les conventions de [BCDL19]. Il ne s'agit pas d'un algorithme spécifique, mais d'un

schéma servant à construire différents algorithmes :

Schéma PGE+SS

1. Choisir aléatoirement un ensemble d'information
2. Faire une élimination gaussienne partielle (PGE)
3. Résoudre un sous-problème à l'aide des techniques de remplissage de sac à dos (SS)
4. Tester le poids des solutions au problème de sac à dos
   (a) si l'un de ces vecteurs est de poids W, c'est une solution
   (b) sinon, on recommence à l'étape 1

Nous allons maintenant décrire toutes ces étapes plus en détail, cela fait intervenir des paramètres libres, ℓ et ω'', qu'il faudra optimiser. Au départ, la matrice $\mathbf{H}$ donnée en entrée ne présente pas de structure particulière (figure 3.4).

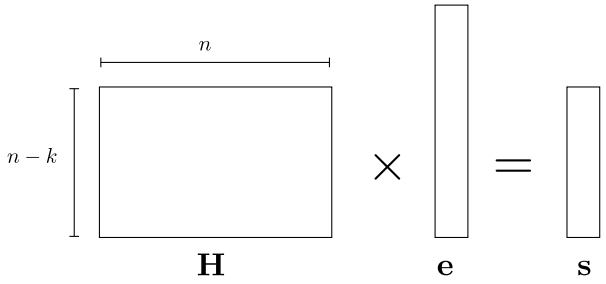

FIGURE 3.4 – Le problème à résoudre au départ.

Étape 1 : On choisit un ensemble d'information. Celui-ci est légèrement plus gros que dans l'algorithme de Prange : il comprend $k+\ell$ coordonnées au lieu de k. On multiplie $\mathbf{H}$ à droite par une matrice de permutation P pour déplacer les vecteurs de l'ensemble d'information vers la droite de $\mathbf{H}$.

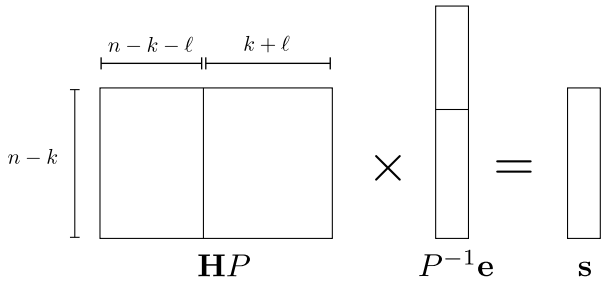

FIGURE 3.5 – Matrice $\mathbf{H}$ après le choix de l'ensemble d'information et le regroupement des coordonnées.

Étape 2 (PGE) : On applique une élimination gaussienne partielle sur les autres coordonnées pour faire apparaître un bloc identité et un bloc de 0, comme indiqué sur la figure 3.6. Cette opération peut se faire en ne se servant que d'opérations sur les lignes : cela revient à multiplier $\mathbf{H}$ à gauche par une matrice Q inversible. La contrainte matricielle à vérifier est $\mathbf{He} = \mathbf{s}$, ce qui est équivalent à :

$$\underbrace{Q\mathbf{H}P}_{\overline{\mathbf{H}}}\underbrace{P^{-1}\mathbf{e}}_{\overline{\mathbf{e}}} = \underbrace{Q\mathbf{s}}_{\overline{\mathbf{s}}}$$

Pour coïncider avec les blocs de $\overline{\mathbf{H}}$, on découpe $\overline{\mathbf{e}}$ en $(\overline{\mathbf{e}}', \overline{\mathbf{e}}'')$ et $\overline{\mathbf{s}}$ en $(\overline{\mathbf{s}}', \overline{\mathbf{s}}'')$. Grâce au calcul matriciel par bloc, l'égalité $\overline{\mathbf{H}}\overline{\mathbf{e}} = \overline{\mathbf{s}}$ se décompose en le système :

$$\begin{cases} \overline{\mathbf{e}}' + \overline{\mathbf{H}}'\overline{\mathbf{e}}'' = \overline{\mathbf{s}}' \\ \overline{\mathbf{H}}''\overline{\mathbf{e}}'' = \overline{\mathbf{s}}'' \end{cases}$$

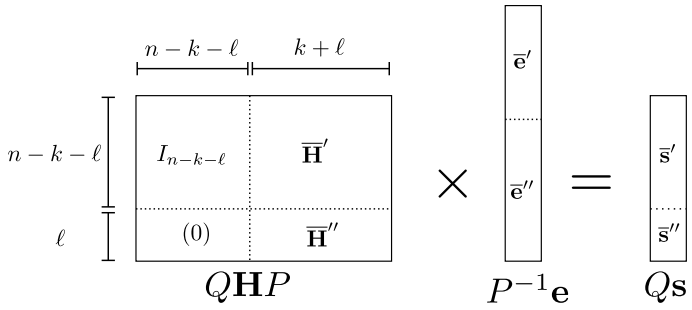

FIGURE 3.6 – Matrice $\mathbf{H}$ après l'élimination Gaussienne partielle.

Étape 3 (SS) : On donne $\overline{\mathbf{H}}''$ et $\overline{\mathbf{s}}''$ à un algorithme de remplissage de sac à dos en demandant des solutions de poids $\lceil \omega'' n \rceil$. En général, pour de bons choix des paramètres ℓ et ω'', ce sous-problème admet beaucoup (un nombre exponentiel) de solutions. L'algorithme de remplissage de sac à dos peut donc renvoyer beaucoup de solutions $\overline{\mathbf{e}}''$ à ce sous-problème.

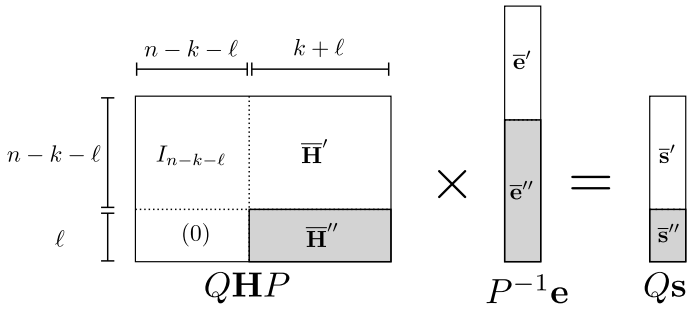

FIGURE 3.7 – En grisé : le sous-problème de remplissage de sac à dos.

Étape 4 : Pour chaque solution $\overline{\mathbf{e}}''$ au sous-problème, on calcule $\overline{\mathbf{e}}' = \overline{\mathbf{s}}' - \overline{\mathbf{H}}'\overline{\mathbf{e}}''$, et on espère que le poids $\omega' n$ de $\overline{\mathbf{e}}'$ vérifie $|\overline{\mathbf{e}}'| + |\overline{\mathbf{e}}''| = W$. On a alors trouvé une solution. Si aucun de ces candidats ne convient, on recommence à l'étape 1 avec un autre ensemble d'information.

Remarque 3.6. Il s'agit d'une généralisation de l'algorithme de Prange, celui-ci correspondant au cas particulier $\ell = 0$.

Pour obtenir un algorithme de décodage efficace, il "suffit" maintenant de bien choisir les paramètres pour ℓ et ω'', et de construire un bon algorithme de remplissage de sac à dos pour résoudre le sous-problème correspondant à $\overline{\mathbf{H}}''$, $\overline{\mathbf{s}}''$ et ω'', c'est-à-dire le problème suivant :

Problème : Remplissage de sac à dos (multiples solutions)
Entrée : $\overline{\mathbf{H}}'' \in \mathbb{F}_q^{\ell \times (k+\ell)}$ de rang ℓ
$\overline{\mathbf{s}}'' \in \mathbb{F}_q^{\ell}$
$\omega'' \in [0, 1]$

Sortie : $L_0 \subset \mathbb{F}_q^{k+\ell}$ tel que :
$\forall \overline{\mathbf{e}}'' \in L_0, \begin{cases} \overline{\mathbf{H}}''\overline{\mathbf{e}}'' = \overline{\mathbf{s}}'' \\ |\overline{\mathbf{e}}''| = \lceil \omega'' n \rceil \end{cases}$

3.2.4 Probabilité de succès

Comme dans le cas de l'algorithme de Prange, nous n'avons aucun contrôle sur le poids de $\overline{\mathbf{e}}'$, et nous ne pouvons qu'espérer que ce poids $\omega' n$ soit celui de la solution attendue. Dans les cas qui nous intéressent, la probabilité que le poids ω' convienne est exponentiellement faible, ce qui implique qu'il faut essayer en moyenne un nombre exponentiel de candidats avant de trouver une solution.

Proposition 3.4. *On appelle probabilité de succès, $\mathbb{P}_{succ}$, la probabilité qu'un vecteur $\overline{\mathbf{e}} = (\overline{\mathbf{e}}', \overline{\mathbf{e}}'')$ soit une solution au problème de décodage sachant que $\overline{\mathbf{e}}''$*

est une solution au sous-problème de sac à dos résolu dans l'étape 3. Cette probabilité est exponentiellement faible, et s'écrit $\mathbb{P}_{succ} = \tilde{\mathcal{O}}\left(2^{p_{succ}n}\right)$*, où :*

$$p_{succ} = (n-k-\ell)h\left(\frac{\omega' n}{n-k-\ell}\right) + \ell \log_2(q) + \max\left(\omega' n \log_2(q-1) - (n-k)\log_2(q)\,; -nh(\omega) - \omega'' n \log_2(q-1)\right)$$

Démonstration. On distingue le cas avec beaucoup de solutions, et le cas dans lequel il n'y a très probablement qu'une solution : le $\mathbf{e_0}$ qui a servi à générer le syndrome $\mathbf{S}$ donné en entrée.

Dans le cas où le problème admet de nombreuses solutions, l'existence de $\mathbf{e_0}$ n'est d'aucune utilité. Quand un prend un vecteur $\overline{\mathbf{e}}''$ de poids $\lceil \omega'' n \rceil$, le vecteur associé $\overline{\mathbf{e}}'$ se comporte comme un vecteur de $n-k-\ell$ coordonnées aléatoires et indépendantes. La probabilité que celui-ci soit de poids ω' est donnée par :

$$\frac{\binom{n-k-\ell}{\omega' n}(q-1)^{\omega' n}}{q^{n-k-\ell}}$$

C'est le nombre de vecteurs à $n-k-\ell$ coordonnées de poids $\omega' n$, divisé par le nombre total de vecteurs de taille $n-k-\ell$.

Dans le cas où $\mathbf{e_0}$ est la seule solution, on calcule la probabilité que $\overline{\mathbf{e}}''$ soit bien celui qui correspond à $\mathbf{e_0}$. Pour cela, la première étape est d'avoir choisi un ensemble d'information compatible avec $\mathbf{e_0}$, c'est-à-dire un ensemble d'information qui contient $\lceil \omega'' n \rceil$ coordonnées non nulles de $\mathbf{e_0}$. Le probabilité d'avoir choisi un bon ensemble d'information est :

$$\frac{\binom{\omega n}{\omega'' n}\binom{n-\omega n}{k+\ell-\omega'' n}}{\binom{n}{k+\ell}}$$

Ensuite, il ne faut faire aucune erreur dans la façon de remplir l'ensemble d'information : seul un vecteur convient sur les $\binom{k+\ell}{\omega'' n}(q-1)^{\omega'' n}$ vecteurs de poids $\lceil \omega'' n \rceil$. Cependant, $\overline{\mathbf{e}}''$ n'est pas quelconque : il fait partie des vecteurs privilégiés qui satisfont $\overline{\mathbf{H}}''\overline{\mathbf{e}}'' = \overline{\mathbf{s}}''$. Seule une fraction $q^{-\ell}$ des vecteurs sont dans ce cas. La probabilité que $\overline{\mathbf{e}}''$ corresponde à $\mathbf{e_0}$ est donc :

$$\frac{\binom{\omega n}{\omega'' n}\binom{n-\omega n}{k+\ell-\omega'' n} q^{\ell}}{\binom{n}{k+\ell}\binom{k+\ell}{\omega'' n}(q-1)^{\omega'' n}}$$

ce qui peut se simplifier en :

$$\frac{\binom{n-k-\ell}{\omega' n} q^\ell}{\binom{n}{\omega n} (q-1)^{\omega'' n}}$$

En prenant le maximum, on obtient une expression qui convient pour les deux cas :

$$\max \left(\frac{\binom{n-k-\ell}{\omega' n} (q-1)^{\omega' n}}{q^{n-k-\ell}}, \frac{\binom{n-k-\ell}{\omega' n} q^\ell}{\binom{n}{\omega n} (q-1)^{\omega'' n}} \right)$$

On obtient l'expression attendue en factorisant ce qui peut l'être, puis en prenant le logarithme. □

3.2.5 Nombre de répétitions

Dans le cas de l'algorithme de Prange, le nombre de répétitions (le nombre de retours à l'étape 1) est directement donné par la probabilité de succès. Ce n'est plus le cas dans le schéma PGE+SS. En effet, l'algorithme de remplissage de sac à dos (dont le temps d'exécution est $\tilde{\mathcal{O}}\left(2^{tn}\right)$, exponentiel en n) fournit un nombre exponentiel de candidats $\tilde{\mathcal{O}}\left(2^{\ell_0 n}\right)$. Ainsi, le nombre de répétitions est plus faible :

Lemme 3.5. *L'espérance de nombre d'ensembles d'informations à essayer, c'est-à-dire le nombre d'appels à la sous-routine de remplissage de sac à dos est :*

$$\tilde{\mathcal{O}}\left(2^{\max(0,-p_{succ}-\ell_0)n}\right)$$

Démonstration. Il faut essayer $\tilde{\mathcal{O}}\left(2^{-p_{succ}n}\right)$ candidats. Or un appel à la sous-routine fournit $\tilde{\mathcal{O}}\left(2^{\ell_0 n}\right)$ candidats. Avec $\tilde{\mathcal{O}}\left(2^{(-p_{succ}-\ell_0)n}\right)$ ensembles d'informations (et donc autant d'appels à la sous-routine), on obtient alors suffisamment de candidats. Cependant, si la sous-routine trouve trop de candidats, il faut quand même l'exécuter un nombre polynomial de fois, d'où le max avec $\tilde{\mathcal{O}}(1)$. □

3.2.6 Complexité d'un algorithme de type PGE+SS

Les opérations de choix d'un ensemble d'information et d'élimination gaussienne partielle prennent un temps polynomial. L'exposant d'un algorithme construit selon le schéma PGE+SS dépend donc de deux éléments :

— le coût d'un appel à la sous-routine de remplissage de sac à dos (*i.e.* le coût de l'étape 3)

— le nombre d'appels à cette sous-routine (*i.e.* le nombre de fois où l'étape 4 n'a pas trouvé de solution et retourne à l'étape 1)

Théorème 3.6. *La complexité d'un tel algorithme de type PGE+SS est :*

$$\tilde{\mathcal{O}}\left(2^{(t+\max(0,-\ell_0-p_{succ}))n}\right)$$

où :

$\tilde{\mathcal{O}}\left(2^{tn}\right)$ *correspond au coût de l'étape 3 (la sous-routine)*
$\tilde{\mathcal{O}}\left(2^{\ell_0 n}\right)$ *est le nombre de candidats trouvés lors de l'étape 3*
$\tilde{\mathcal{O}}\left(2^{-p_{succ}n}\right)$ *est l'espérance du nombre de candidats à essayer*

Bien évidemment, t, ℓ_0 et p_{succ} dépendent tous les trois de ℓ et de ω'', ce qui complique le choix de ces paramètres. Les relations entre ℓ_0 et t dépendent également de l'algorithme de remplissage de sac à dos.

Remarque 3.7. Le problème de décodage consiste à trouver un vecteur $\mathbf{e}$ qui vérifie $\mathbf{He} = \mathbf{s}$ et $|\mathbf{e}| = \lceil \omega n \rceil$. Nous venons d'effectuer une réduction dans laquelle nous nous ramenons à rechercher un vecteur $\overline{\mathbf{e}}''$ vérifiant $\overline{\mathbf{H}}''\overline{\mathbf{e}}'' = \overline{\mathbf{s}}''$ et $|\overline{\mathbf{e}}''| = \lceil \omega'' n \rceil$, cette fois à l'aide d'un algorithme de sac à dos. Alors qu'il s'agit d'un autre problème de décodage, similaire au problème de départ !

Cette différence de traitement vient du fait que ces deux problèmes de décodage ne se situent pas dans les mêmes zones de paramètres : dans le problème de départ, les dimensions n et $n-k$ de $\mathbf{H}$ sont du même ordre de grandeur et une élimination gaussienne donne de bons résultats. En revanche, dans le problème réduit, la matrice $\overline{\mathbf{H}}''$ est très aplatie. Dans ce cas, les techniques de remplissage de sac à dos sont très efficaces pour trouver d'un seul coup de nombreuses solutions.

Si k est très proche de n et que $\mathbf{H}$ est très plate, les algorithmes de décodage présentant la structure de 3.2.3 dégénèrent en $\ell = n - k$: il ne reste plus que le remplissage de sac à dos.

3.3 L'algorithme de Wagner dans le décodage

L'étape 3 d'un algorithme de type PGE+SS consiste en l'appel d'une sous-routine qui résout un problème de remplissage de sac à dos, et doit renvoyer de nombreuses solutions. Si ce problème admet suffisamment de solutions, on peut utiliser l'algorithme de Wagner. Plus précisément, on utilise dans ce cas l'algorithme de Wagner lissé 1.2.6, modifié pour renvoyer beaucoup de solutions.

3.3.1 Dans le cas classique

Lemme 3.7. *Nous disposons d'un espace de recherche de taille 2^Ω, et une contrainte sur c bits nous est imposée. Si $c < \Omega/2$, l'algorithme de Wagner avec $a+1$ niveaux peut trouver 2^λ solutions en temps $\tilde{\mathcal{O}}\left(2^\lambda\right)$, où :*

$$\lambda = \frac{c - \Omega/2^{a-1}}{a-2}$$

et le nombre d'étages $a+1$ est donné par le a qui vérifie :

$$\frac{a\Omega}{2^a} \leqslant c < \frac{(a-1)\Omega}{2^{a-1}}$$

Démonstration. On considère une structure d'arbre de $a+1$ niveaux, et on utilise une décomposition gauche-droite à tous les étages. Au niveau a (le dernier niveau), on construit les listes en énumérant entièrement l'espace disponible. Ces listes sont donc de taille $\mu = \frac{\Omega}{2^a}$. Pour passer au niveau $a-1$, on choisit une contrainte modulaire $2\mu - \lambda$. De cette façon, les listes du niveau $a-1$ sont de taille λ. Ensuite, toutes les contraintes modulaires ajoutées et toutes les listes sont de taille λ.

En particulier, la liste du niveau 0 contient 2^λ éléments, qui sont autant de solutions.

Pour que cet algorithme puisse exister, il faut que la contrainte modulaire $2\mu - \lambda$ soit un nombre positif. Cela correspond exactement à la contrainte $c < \frac{(a-1)\Omega}{2^{a-1}}$.

Par ailleurs, pour la correction, il faut que la contrainte modulaire totale soit une contrainte sur c bits. On peut constater que la valeur de λ a été choisie exprès pour satisfaire $(a-1)\lambda + (2\mu - \lambda) = c$. Les listes du niveau 0 sont donc bien des solutions.

Enfin, pour que l'algorithme s'exécute en temps $\tilde{\mathcal{O}}\left(2^\lambda\right)$, il faut que toutes les listes soient de taille au plus λ. Le seul cas potentiellement problématique est le cas des listes du niveau a. Pour cela, il faut vérifier $\mu \leqslant \lambda$, ce qui revient exactement à $\frac{a\Omega}{2^a} \leqslant c$. □

On veut maintenant appliquer ce résultat pour l'étape 3 d'un algorithme de type PGE+SS. Dans ce cas, l'espace de recherche est l'ensemble des vecteurs de taille $(k+\ell)$ et de poids $\lceil \omega'' n \rceil$. Les solutions doivent vérifier une condition $\overline{\mathbf{H}}''\overline{\mathbf{e}}'' = \overline{\mathbf{s}}''$, c'est-à-dire une condition sur ℓ coordonnées q-aires. On a donc :

$$\begin{cases} \Omega &= (k+\ell) h\left(\frac{\lceil \omega'' n \rceil}{k+\ell}\right) + \lceil \omega'' n \rceil \log_2(q-1) \\ c &= \ell \log_2(q) \end{cases}$$

De façon générale, si on cherche des vecteur de distribution $D^{k+\ell}$, la taille de l'espace de recherche 2^Ω est donnée par l'entropie de la distribution $\Omega = (k+\ell)\tilde{h}(D)$.

Le nombre de solutions trouvées, 2^λ, ainsi que le nombre de candidats à essayer $1/\mathbb{P}_{succ}$ dépendent tous les deux de ℓ ainsi que de la distribution ciblée $D^{k+\ell}$. On cherchera souvent à équilibrer ces paramètres pour avoir $1/\mathbb{P}_{succ} = 2^\lambda$ d'une part, puis pour minimiser cette quantité, qui correspond à la complexité du décodage par l'algorithme de Wagner.

Remarque 3.8. Bien entendu, il est possible (et même recommandé) d'ajouter des représentations pour améliorer cet algorithme. Cependant, il devient alors difficile d'établir des résultats généraux et tout se fait au cas par cas.

3.3.2 Dans le cas quantique

Dans le cas quantique, on peut profiter des marches quantiques pour tester des candidats en temps racine. Le lemme 3.7 est légèrement modifié et devient :

Lemme 3.8. *Nous disposons d'un espace de recherche de taille 2^Ω, et une contrainte sur c bits nous est imposée. Si $c < 2\Omega/5$, une variante quantique de l'algorithme de Wagner avec $a+1$ niveaux peut tester $2^{2\lambda}$ solutions en temps $\tilde{\mathcal{O}}\left(2^\lambda\right)$, où :*

$$\lambda = \frac{c - \Omega/2^{a-1}}{a - 2 - 1/2^{a-1}}$$

et le nombre d'étages $a+1$ est donné par le a qui vérifie :

$$\frac{a\Omega}{2^a+1} \leqslant c < \frac{(a-1)\Omega}{2^{a-1}+1}$$

Démonstration. On combine l'algorithme de Wagner avec une marche quantique, la structure est illustrée dans la figure 3.8.

Comme dans 2.3.2, on distingue au dernier niveau des listes gauches et des listes droites. Les listes gauches sont de tailles λ et sont obtenues en échantillonnant un espace de taille $\lambda\left(1 + \frac{1}{2^{a-1}}\right)$. Ce sont les listes sur lesquelles on marche. Les listes droites sont de taille $\mu = \frac{(a-1)\Omega - (2^{a-1}+1)c}{2^{a-1}(a-2)+1}$. Ces listes sont fixes. Par ailleurs, cela exploite exactement tout l'espace disponible, puisqu'on a $\Omega = 2^{a-1}\lambda\left(1 + \frac{1}{2^{a-1}}\right) + 2^{a-1}\mu$.

Pour passer au niveau $a-1$, on prend une contrainte modulaire μ. Les listes du niveau $a-1$ sont alors de taille λ. Pour les étages supérieurs, toutes les contraintes modulaires ajoutées et toutes les listes sont de taille λ.

En particulier, la liste du niveau 0 est de taille λ, mais explore lors de la marche un espace virtuel de taille 2λ. De plus, cette liste contient des solutions, les paramètres λ et μ ayant été spécifiquement choisis tels que $(a-1)\lambda + \mu = c$.

De façon similaire au cas classique, la valeur de la dernière contrainte de modulo est positive, ce qui est assuré par l'hypothèse $c < \frac{(a-1)\Omega}{2^{a-1}+1}$.

Pour que le coût de l'étape de SETUP soit $\mathsf{S} = \tilde{\mathcal{O}}\left(2^{\lambda n}\right)$, il faut que les listes droites du dernier étage ne soient pas le facteur limitant, c'est-à-dire qu'il

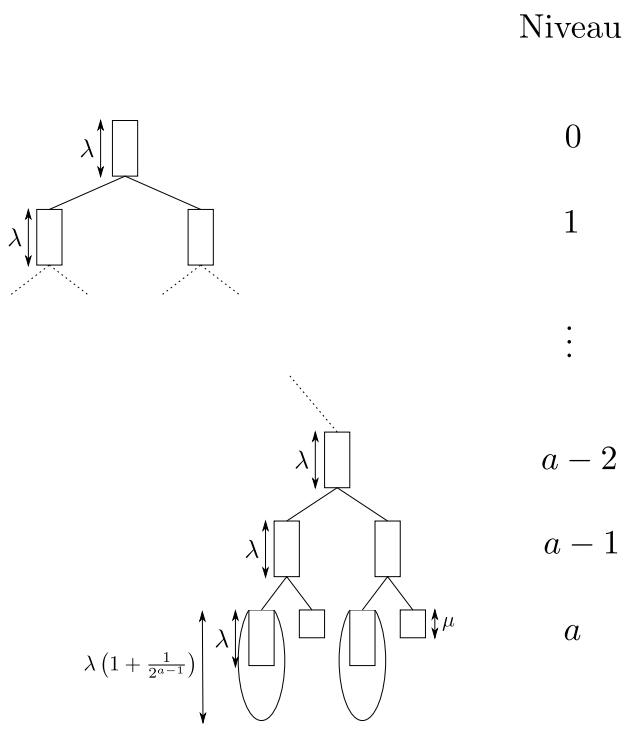

FIGURE 3.8 – Structure de l'algorithme utilisé pour résoudre le sous-problème de remplissage de sac à dos dans le cas quantique

faut $\mu \leqslant \lambda$. Cela est équivalent à l'hypothèse $\frac{a\Omega}{2^a+1} \leqslant c$. Le coût du CHECK est $\mathsf{C} = \tilde{\mathcal{O}}\left(2^{\lambda/2}\right)$: on recherche par amplification d'amplitude un bon élément dans une liste de taille λ. Enfin, puisqu'on n'utilise pas de représentations et que toutes les listes ont la même taille, le coût de l'UPDATE est $\mathsf{U} = \tilde{\mathcal{O}}(1)$.

Par ailleurs, la proportion des nœuds marqués est $\varepsilon = 1/\lambda$ (on cherche un élément dans un espace de taille λ^2, et la liste du niveau 0 contient déjà λ éléments). Enfin, $\delta = 1/\lambda$ provient directement de la taille λ des listes sur lesquelles on marche, à savoir les listes gauches du niveau a.

Le coût de cet algorithme de remplissage de sac à dos est alors :

$$\mathsf{S} + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C}\right) = \tilde{\mathcal{O}}\left(2^{\lambda}\right)$$

□

Pour utiliser ce lemme dans un algorithme PGE+SS, les paramètres Ω et c sont calculés exactement comme dans le cas classique.

Cependant, l'équilibrage sera différent. En effet, cet algorithme permet de tester $2^{2\lambda}$ candidats, contre seulement 2^{λ} dans le cas classique. Cette fois, on choisira ℓ et la distribution $D^{k+\ell}$ de façon à ce que le nombre de candidats

à tester vérifie $1/\mathbb{P}_{succ} = 2^{2\lambda}$, et ce pour un λ le plus petit possible. De cette façon, on peut résoudre le problème de décodage en temps $\tilde{\mathcal{O}}\left(2^{\lambda}\right)$.

3.4 Le cas binaire

Nous faisons une petite digression en nous attardant sur le cas binaire. L'objectif est de noter les points qui diffèrent par rapport au remplissage de sac à dos étudié dans la première partie, et en particulier les conséquences du fait de se placer dans un corps fini.

Dans le cas binaire, le sous-problème à résoudre est le suivant :

Problème : Sous-problème réduit
Entrée : $\overline{\mathbf{H}}'' \in \mathbb{F}_2^{\ell \times (k+\ell)}$
$\overline{\mathbf{s}}'' \in \mathbb{F}_2^{\ell}$
$\omega'' \in [0,1]$

Sortie : $\overline{\mathbf{e}}'' \in \mathbb{F}_2^{k+\ell}$ tel que $\begin{cases} \overline{\mathbf{H}}''\overline{\mathbf{e}}'' = \overline{\mathbf{s}}'' \\ |\overline{\mathbf{e}}''| = W'' := \lceil \omega'' n \rceil \end{cases}$

La matrice $\overline{\mathbf{H}}''$ est très plate et ce problème admet beaucoup de solutions. L'objectif est de trouver le plus de solutions possible, et le plus rapidement possible. Une très bonne façon pour cela est de se servir des techniques de remplissage de sac à dos. Par conséquent, l'évolution des algorithmes de décodage de syndrome est fortement corrélée au développement du remplissage de sac à dos. Par exemple, l'algorithme de Dumer ([Dum91]) peut être vu comme un algorithme de type PGE+SS pour lequel la partie sac à dos est résolue avec l'algorithme de [HS74]. Par la suite, les algorithmes de [MMT11] et [BJMM12] utilisent l'idée des représentations introduites dans [HJ10].

3.4.1 Calcul du nombre de représentations

Nous allons nous attarder quelque peu sur ces derniers algorithmes, en raison d'une subtilité concernant les représentations.

May, Meurer et Thomae ont présenté dans [MMT11] un algorithme qu'ils ont construit en adaptant les représentations de [HJ10]. Leur algorithme a pour complexité $\tilde{\mathcal{O}}\left(2^{0.1116n}\right)$ dans le pire cas.

Cependant, en faisant cela, les représentations ne sont pas utilisées à leur plein potentiel. Une différence fondamentale entre les problèmes de décodage et de sac à dos décrits précédemment est que le vecteur à renvoyer est dans $\mathbb{F}_2$ et non dans $\{0,1\}$. $\mathbb{F}_2$ présente l'immense avantage d'être stable par addition, là où dans $\{0,1\}$ on devait supprimer tous les vecteurs contenant un 2.

Ceci est exploité dans [BJMM12] : lors de l'utilisation de représentations, on ne cherche plus un vecteur de poids ωn comme la somme de deux vecteurs

de poids $\omega n/2$, mais comme la somme de deux vecteurs de poids $\omega n/2+\varepsilon n$, où ε est un paramètre à optimiser. Cela permet d'exploiter des représentations de bien meilleure qualité, comme on peut le voir dans :

Proposition 3.9. *Soit $D^n[\omega]$ une distribution et $\varepsilon \in [0, 1-\omega]$, la proportion des éléments bien formés lors de l'utilisation de $(D^n[\omega], D^n[\omega/2+\varepsilon], D^n[\omega/2+\varepsilon])-$représentations est :*

$$\tilde{\mathcal{O}}\left(2^{n(\tilde{h}(\omega/2,\omega/2,\varepsilon)-2\tilde{h}(\omega/2+\varepsilon))}\right)$$

Démonstration. Dans un vecteur de distribution $D^n[\omega]$ obtenu comme la somme de deux vecteurs de distribution $D^n[\omega/2+\varepsilon]$, on peut trier les coordonnées de ce vecteur en quatre catégories :

— les 1 obtenus comme $0+1$, il y en a $\omega n/2$

— les 1 obtenus comme $1+0$, il y en a également $\omega n/2$

— les 0 obtenus comme $1+1$, il y en a εn

— les 0 obtenus comme $0+0$: $n-\omega n-\varepsilon n$.

Le nombre de façons d'obtenir un vecteur de distribution $D^n[\omega]$ en additionnant deux vecteurs de distribution $D^n[\omega/2+\varepsilon]$ est déterminé par la façon dont ces quatre catégories se répartissent les coordonnées : il y en a $\tilde{\mathcal{O}}\left(2^{\tilde{h}(\omega/2,\omega/2,\varepsilon)n}\right)$.

Par ailleurs, il y a $\tilde{\mathcal{O}}\left(2^{2\tilde{h}(\omega/2+\varepsilon)n}\right)$ façons de choisir un couple de vecteurs de $D^n[\omega/2+\varepsilon]$. La probabilité que la somme de deux vecteurs de $D^n[\omega/2+\varepsilon]$ donne un vecteur de $D^n[\omega]$ est donc :

$$\tilde{\mathcal{O}}\left(2^{n(\tilde{h}(\omega/2,\omega/2,\varepsilon)-2\tilde{h}(\omega/2+\varepsilon))}\right)$$

□

Si on choisit $\varepsilon = \frac{1-\omega+\sqrt{1-2\omega}}{2}$, on obtient $\tilde{h}(\omega/2, \omega/2, \varepsilon) = 2\tilde{h}(\omega/2+\varepsilon)$: les vecteurs bien formés constituent une fraction polynomiale des sommes obtenues. Puisque seuls les vecteurs bien formés sont utiles, mais qu'il faut payer le coût du calcul de tous les vecteurs, ce cas est fort sympathique puisque les éléments mal formés n'augmentent pas inutilement l'exposant de la complexité.

Remarque 3.9. L'algorithme de [MMT11] choisit systématiquement $\varepsilon = 0$, ce qui n'est pas optimal. Avec un ε plus grand, on peut diminuer la probabilité de créer des éléments mal formés. Par ailleurs, choisir ε plus grand permet également d'augmenter la taille de l'espace de recherche, puisqu'on cherche alors des vecteurs dans $D^n[\omega/2n+\varepsilon n]$, et non plus dans $D^n[\omega/2n]$. Ces deux considérations vont dans le même sens : il est beaucoup plus facile de trouver de nombreuses solutions avec $\varepsilon > 0$.

Cela dit, il peut être plus avantageux de choisir un ε encore plus grand que $\frac{1-\omega+\sqrt{1-2\omega}}{2}$, comme cela a été fait dans [BJMM12]. Il y a alors à nouveau des éléments mal formés, mais l'espace de recherche est aussi bien plus grand. Comme toujours, toute la difficulté consiste à trouver le bon équilibre. L'algorithme de [BJMM12] a, dans le pire cas, une complexité $\tilde{\mathcal{O}}\left(2^{0.1019n}\right)$.

3.4.2 Les recherches de voisins proches

Il est nécessaire de mentionner l'utilisation de techniques de recherche de plus proches voisins pour améliorer la complexité asymptotique du décodage. Cela a été initialement montré dans [MO15], puis amélioré ([BM17, BM18]). Cette technique permet des améliorations pour de nombreux jeux de paramètres, et peut être généralisée au cas q-aire ([Hir16, GKH17]).

Nous allons décrire grossièrement ce en quoi cela consiste. Une étude complète peut être trouvée dans [Car20].

L'objectif de cette technique est de limiter le coût des vecteurs mal formés lors de la fusion de listes avec des représentations. On rappelle que la fusion de listes consiste à prendre deux listes $L_1 = \{x_1, \ldots, x_m\}$ et $L_2 = \{y_1, \ldots, y_n\}$, une fonction linéaire $\mathcal{L}$, une condition de modulo t modulo M, et une distribution cible D. Il faut alors déterminer les $x_i + y_j$ qui vérifient :

$$\begin{cases} \mathcal{L}(x_i + y_j) \equiv t \ [M] \\ x_i + y_j \in D \end{cases}$$

Comme on l'a vu en 0.1.5, on obtient un coût $\frac{|L_1| \times |L_2|}{M}$ et énumérant les couples satisfaisant $\mathcal{L}(x_i + y_j) \equiv t \ [M]$, et en ne retenant que ceux qui vérifient également $x_i + y_j \in D$.

Pour fusionner des listes avec la technique des plus proches voisins, on se donne une fonction de hachage floue : deux vecteurs présentant des caractéristiques similaires ont plus de chances d'être triés dans la même catégorie. Cette fonction de hachage permet de décomposer L_1 en plusieurs sous-ensembles $\mathcal{B}_1, \ldots, \mathcal{B}_k$, et L_2 en $\mathcal{C}_1, \ldots, \mathcal{C}_k$. On cherche alors des collisions entre $\mathcal{B}_1$ et $\mathcal{C}_1$, entre $\mathcal{B}_2$ et $\mathcal{C}_2$, ..., et entre $\mathcal{B}_k$ et $\mathcal{C}_k$.

Grâce aux propriétés locales de la fonction de hachage, les collisions entre $\mathcal{B}_i$ et $\mathcal{C}_i$ auront plus de chance de vérifier la condition $x_i + y_j \in D$ que les collisions entre L_1 et L_2.

En faisant cela, en rate une partie des solutions. On recommence alors avec d'autres fonctions de hachages, pour obtenir des ensembles $\mathcal{B}_i$ et $\mathcal{C}_i$ différents, et trouver d'autres solutions.

Pour obtenir une fusion de listes efficace, il faut alors trouver de bonnes fonctions de hachages, et optimiser le nombre d'itérations ainsi que les tailles des ensembles $\mathcal{B}_i$ et $\mathcal{C}_i$.

Cependant, nous n'allons pas étudier plus en détail cette méthode. D'une part, on ne pensait pas possible, jusque récemment, de s'affranchir des facteur

super-polynomiaux engendrés. D'autre part, on ne sait pas encore ce qu'il est possible de faire comme adaptation de cette technique pour $\mathbb{F}_q$.

3.4.3 Complexité du décodage binaire

Pour conclure ce survol de l'état de l'art dans le cas binaire, nous indiquons dans la table 3.9 la complexité de différents algorithmes de décodage de syndrome.

[Pra62]	$\tilde{\mathcal{O}}\left(2^{0.1207n}\right)$
[MMT11]	$\tilde{\mathcal{O}}\left(2^{0.1116n}\right)$
[BJMM12]	$\tilde{\mathcal{O}}\left(2^{0.1019n}\right)$
[MO15]	$2^{o(n)}2^{0.0967n}$
[BM18]	$2^{o(n)}2^{0.0885n}$
[Car20]	$\tilde{\mathcal{O}}\left(2^{0.0882n}\right)$

FIGURE 3.9 – Complexité de différents algorithmes de décodage de syndrome dans le cas binaire

Chapitre 4

Décodage ternaire à gros poids

Dans ce chapitre, nous étudierons particulièrement le cas du décodage dans $\mathbb{F}_3$, pour des poids proches de 1.

Ce type de paramètres n'a que très peu été étudié, ce qui s'explique par plusieurs facteurs. D'une part, le problème du décodage de syndrome provient de la théorie de la communication et des canaux bruités. Le choix d'une base différente du binaire est donc moins naturel. Par ailleurs, un canal de communication est censé avoir peu de bruit, et donner un message contenant peu d'erreurs. Reconstituer un message dont le poids de l'erreur est proche de 1, c'est-à-dire un message dont presque toutes les coordonnées ont été modifiées, est un problème complètement artificiel. Enfin, le cas des grands poids est fondamentalement identique au cas des poids faibles dans le cas binaire. L'étude du décodage de syndrome pour des grands poids est donc une spécificité pour $q \geqslant 3$, bien moins étudié que le cas binaire.

Cependant, comme nous pouvons le constater sur la figure 3.3, le décodage ternaire à gros poids semble très difficile, du moins bien plus que le cas binaire. Or, du point de vue de la cryptographie, un problème difficile est un bon candidat pour construire un protocole résistant. La publication du schéma de signature Wave [DST19] par Debris-Alazard, Sendrier et Tillich constitue une excellente motivation à la recherche d'algorithmes efficaces pour résoudre le problème de décodage de syndrome ternaire à gros poids.

Pour fixer les idées, définissons ce que signifie le décodage à gros poids :

Définition 4.1 (Gros poids)**.** *Par gros poids, on entend qu'on considère la zone de paramètres* $\frac{q+R-1}{q} \leqslant \omega$ *(c'est-à-dire* $\frac{R+2}{3} \leqslant \omega$ *dans le cas ternaire). Cela correspond aux poids pour lesquels la complexité du décodage est à nouveau exponentielle. Sur la figure 3.3, cela recouvre la zone* 4*, ainsi que la zone* 5 *si celle-ci existe.*

Pour résoudre le problème du décodage dans ce cas, nous utiliserons le schéma PGE+SS décrit dans chapitre précédent :

Schéma PGE+SS (rappel)

1. Choisir un ensemble d'information
2. Faire une élimination gaussienne partielle (PGE)
3. Résoudre un sous-problème à l'aide des techniques de remplissage de sac à dos (SS)
4. Tester le poids des solutions au problème de sac à dos
   (a) si l'un de ces vecteurs est de poids W, c'est une solution
   (b) sinon, on recommence à l'étape 1

Ce chapitre sera consacré à la recherche d'algorithmes efficaces pour exécuter l'étape 3, à savoir :

Problème : Remplissage de sac à dos (multiples solutions)
Entrée : $\overline{\mathbf{H}}'' \in \mathbb{F}_3^{\ell \times (k+\ell)}$ de rang ℓ
$\overline{\mathbf{s}}'' \in \mathbb{F}_3^{\ell}$
$\omega'' \in [0, 1]$

Sortie : $L_0 \subset \mathbb{F}_3^{k+\ell}$ tel que :
$\forall \overline{\mathbf{e}}'' \in L_0, \begin{cases} \overline{\mathbf{H}}''\overline{\mathbf{e}}'' = \overline{\mathbf{s}}'' \\ |\overline{\mathbf{e}}''| = \lceil \omega'' n \rceil \end{cases}$

Nous rappelons le théorème 3.6, qui donne la complexité du problème de décodage en fonction des performances de l'algorithme de remplissage de sac à dos :

Théorème. *La complexité d'un tel algorithme de type PGE+SS est :*

$$\tilde{\mathcal{O}}\left(2^{(t+\max(0,-\ell_0-p_{succ}))n}\right)$$

où :

$\tilde{\mathcal{O}}\left(2^{tn}\right)$ *correspond au coût de l'étape 3 (la sous-routine)*
$\tilde{\mathcal{O}}\left(2^{\ell_0 n}\right)$ *est le nombre de candidats trouvés lors de l'étape 3*
$\tilde{\mathcal{O}}\left(2^{-p_{succ} n}\right)$ *est l'espérance du nombre de candidats à essayer*

4.1 Réduction

La recherche d'un bon algorithme de remplissage peut être très laborieuse. Pour faciliter cette recherche, nous allons commencer pour énoncer

quelques hypothèses simplificatrices. Ces restrictions sont fondées sur des observations expérimentales, et il n'est pas certain que les algorithme engendrés soient optimaux.

4.1.1 Utiliser un poids plein

On rappelle qu'on n'a aucun contrôle sur ce que fait l'élimination gaussienne. Statistiquement, lors de la résolution du système linéaire, il y aura autant de 0, de 1 et de 2. Plus on cherche à obtenir un vecteur déséquilibré, plus il faudra essayer de candidats.

Le cas présent est déséquilibré en faveur des gros poids : le problème exige qu'on lui trouve une solution de gros poids. Pour faciliter la recherche d'une telle solution, on veut placer beaucoup de poids dans le sous-problème de sac à dos.

Ainsi, plus on choisit $\lceil \omega'' n \rceil$ proche de $k + \ell$, plus p_{succ} est grand : il faut essayer moins de candidats avant d'en trouver un qui convienne. D'un autre côté, si on se donne la possibilité d'avoir quelques 0 dans le problème de sac à dos, cela assouplit les contraintes qui pèsent sur ce problème, et trouver beaucoup de solutions est alors plus facile. On peut alors obtenir un meilleur algorithme de remplissage de sac à dos.

Empiriquement, cela n'est pas rentable : j'ai systématiquement obtenu de moins bons résultats en essayant de choisir un $\lceil \omega'' n \rceil < k + \ell$. Dans la suite, on ne se posera pas de questions et choisira donc toujours :

Résultat empirique 4.1 (Répartition du poids)**.** *Dans la réduction du problème de décodage au problème de sac à dos, on choisit* $\lceil \omega'' n \rceil = k + \ell$*. Autrement dit, on cherche des solutions au problème de sac à dos qui ne contiennent que des* 1 *et des* 2*, mais aucun* 0*.*

4.1.2 Retour aux solutions dans $\{0, 1\}$

Comme indiqué par l'heuristique 4.1, nous cherchons des vecteurs $\overline{\mathbf{e}}''$ de poids plein, *i.e.* $k + \ell$. Ces vecteurs doivent également vérifier $\overline{\mathbf{H}}''\overline{\mathbf{e}}'' = \overline{\mathbf{s}}''$. Or, nous pouvons constater, de façon similaire à ce que nous avions fait en 3.3, que l'ensemble de conditions :

$$\begin{cases} \overline{\mathbf{H}}''\overline{\mathbf{e}}'' = \overline{\mathbf{s}}'' \\ \overline{\mathbf{e}}'' \text{ contient uniquement des 1 et des 2} \end{cases}$$

est équivalent à :

$$\begin{cases} \overline{\mathbf{H}}''(\overline{\mathbf{e}}'' - \mathbb{1}) = \overline{\mathbf{s}}'' - \overline{\mathbf{H}}''\mathbb{1} \\ \overline{\mathbf{e}}'' \text{ contient uniquement des 1 et des 2} \end{cases}$$

soit :

$$\begin{cases} \overline{\mathbf{H}}''\overline{\mathbf{e}}^{(\mathbf{3})} = \overline{\mathbf{s}}^{(\mathbf{3})} \\ \overline{\mathbf{e}}^{(\mathbf{3})} \text{ contient uniquement des 0 et des 1} \end{cases}$$

Ainsi, nous pouvons nous ramener au cas de la recherche de vecteurs dans $\mathbb{F}_3$, qui ne contiennent que des 0 et des 1.

Remarque 4.1. Le fait de rechercher des vecteurs contenant uniquement des 0 et des 1, plutôt que des 1 et des 2, ne présente pas d'avantage fondamental. Cette réduction n'est effectuée que par souci de confort, pour travailler sur un problème plus proche des cas connus. Cela permet de transposer plus naturellement les techniques présentées précédemment.

En prenant en compte cette réduction, le problème de remplissage de sac à dos se ramène à :

Problème : Remplissage de sac à dos (multiples solutions)
Entrée : $\overline{\mathbf{H}}'' \in \mathbb{F}_3^{\ell \times (k+\ell)}$
$\overline{\mathbf{s}}^{(\mathbf{3})} \in \mathbb{F}_3^{\ell}$

Sortie : $L_0 \subset \mathbb{F}_3^{k+\ell}$ tel que $\forall \overline{\mathbf{e}}^{(\mathbf{3})} \in L_0, \begin{cases} \overline{\mathbf{H}}''\overline{\mathbf{e}}^{(\mathbf{3})} = \overline{\mathbf{s}}^{(\mathbf{3})} \\ \overline{\mathbf{e}}^{(\mathbf{3})} = (\mathbb{F}_3 \setminus \{2\})^{k+\ell} \end{cases}$

4.1.3 Résultats empiriques

En plus de chercher des vecteurs de poids plein (qu'on ramène à des vecteurs dans $\{0, 1\}$), on utilisera les deux simplifications suivantes :

Résultat empirique 4.2 (Trouver le nombre exact de candidats)**.** *On ajuste ℓ pour avoir $\ell_0 = -p_{succ}$: l'algorithme de sac à dos fournit le nombre de candidats requis pour trouver une solution. Par conséquent, la probabilité que l'étape 4 échoue et renvoie à l'étape 1 n'est qu'inverse-polynomiale.*

Le cas $\ell_0 < -p_{succ}$ présente un intérêt que nous nous contentons de mentionner ici. Dans un algorithme de type PGE+SS, la sous-routine de remplissage de sac à dos est la partie la plus gourmande en mémoire. Dans le cadre de compromis temps-mémoire, on aurait sûrement envie de choisir $\ell_0 < -p_{succ}$. Cela diminuerait le coût en mémoire de l'étape 3. En contrepartie la probabilité de succès à l'étape 4 deviendrait exponentiellement faible, ce qui augmenterait le temps global du décodage.

Résultat empirique 4.3 (Temps amorti 1)**.** *Si le problème possède suffisamment de solutions, il est préférable que l'algorithme résolvant le problème réduit trouve les solutions en temps amorti* 1*. C'est-à-dire qu'il doit vérifier $t = \ell_0$.*

Encore une fois, il s'agit d'observations empiriques, qui fonctionnent très bien dans les cas étudiés, mais n'ont pas prétention à être universelles.

4.2 Représentations dans $\mathbb{F}_3$

Comme nous l'avons constaté dans le premier chapitre, les représentations sont un outil puissant pour accélérer les algorithmes de remplissage de sac à dos. Globalement, l'utilisation des représentations dans $\mathbb{F}_3$ est très similaire à ce qui a été fait précédemment. Les calculs de cas typique et de dénombrement de vecteurs bien formés sont cependant différents. Pour étudier cela, nous commençons par introduire :

Définition 4.2 (Distribution dans $\mathbb{F}_3$)**.** *On notera $D_3^n[\alpha, \beta]$ l'ensemble des vecteurs de $\mathbb{F}_3^n$ dont $\lfloor \alpha n \rfloor$ coordonnées sont des 1, $\lfloor \beta n \rfloor$ sont des 2, et le reste, $n - \lfloor \alpha n \rfloor - \lfloor \beta n \rfloor$, sont des 0.*

Remarque 4.2. Dans le cas du sac à dos dans $\mathbb{Z}$, on paramétrait les distributions par le poids total du vecteurs plutôt que par le nombre de 1. En effet, le poids est, dans $\mathbb{Z}$, un paramètre additif, et l'utiliser permet d'économiser une équation. Dans le cas modulaire, le poids n'a plus aucun sens, et on utilise donc le paramétrage naturel par le nombre de 1.

4.2.1 Éléments bien formés

Proposition 4.1. *Lors d'une fusion de listes utilisant des représentations de paramètres $(D_3^n[\alpha, \beta], D_3^n[\alpha_1, \beta_1], D_3^n[\alpha_1, \beta_1])$, la proportion des éléments bien formés parmi les collisions est :*

$$\tilde{\mathcal{O}}\left(2^{(\tilde{h}(x_{00},x_{01},x_{02},x_{10},x_{11},x_{12},x_{20},x_{21},x_{22}) - 2\tilde{h}(\alpha_1,\beta_1))n}\right)$$

où :

$$\begin{array}{rcccc}
x_{01} = x_{10} & = & \frac{2\alpha + \beta - \alpha_1 - 2\beta_1}{3} & + & z \\
x_{02} = x_{20} & = & \frac{\alpha + 2\beta - 2\alpha_1 - \beta_1}{3} & + & z \\
x_{12} = x_{21} & = & 0 & + & z \\
x_{00} & = & 1 - \alpha - \beta & - & 2z \\
x_{11} & = & \frac{-2\alpha - \beta + 4\alpha_1 + 2\beta_1}{3} & - & 2z \\
x_{22} & = & \frac{-\alpha - 2\beta + 2\alpha_1 + 4\beta_1}{3} & - & 2z.
\end{array}$$

et z est la racine réelle de $x_{01}x_{02}x_{12} = x_{00}x_{11}x_{22}$

Démonstration. Soient $\mathbf{e_1}$ et $\mathbf{e_2}$ deux vecteurs de $D_3^n[\alpha_1, \beta_1]$. On cherche à calculer la probabilité que $\mathbf{e} := \mathbf{e_1} + \mathbf{e_2}$ soit dans $D_3^n[\alpha, \beta]$.

Pour $i, j \in \mathbb{F}_3$, notons x_{ij} la proportion des coordonnées pour lesquelles $\mathbf{e_1}$ présente un i et $\mathbf{e_2}$ un j.

Il y a $\tilde{\mathcal{O}}\left(2^{\tilde{h}(x_{00},x_{01},x_{02},x_{10},x_{11},x_{12},x_{20},x_{21},x_{22})n}\right)$ façons de choisir $\mathbf{e_1}$ et $\mathbf{e_2}$ tels que : pour tous i, j, on additionne un i de $\mathbf{e_1}$ et un j de $\mathbf{e_2}$ sur $x_{ij}n$ coordonnées.

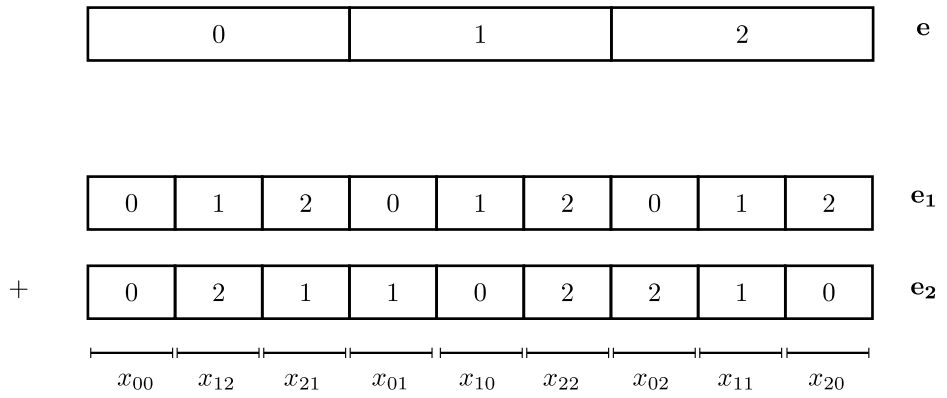

FIGURE 4.1 – Décomposition de $\mathbf{e}$ en $\mathbf{e_1} + \mathbf{e_2}$

Ces x_{ij} vérifient un certain nombre de contraintes, à savoir :

$$\left.\begin{array}{rcl} x_{00}+x_{01}+x_{02} &=& 1-\alpha_1-\beta_1 \\ x_{10}+x_{11}+x_{12} &=& \alpha_1 \\ x_{20}+x_{21}+x_{22} &=& \beta_1 \end{array}\right\} \mathbf{e_1} \in D_3^n[\alpha_1,\beta_1]$$

$$\left.\begin{array}{rcl} x_{00}+x_{10}+x_{20} &=& 1-\alpha_1-\beta_1 \\ x_{01}+x_{11}+x_{21} &=& \alpha_1 \\ x_{02}+x_{12}+x_{22} &=& \beta_1 \end{array}\right\} \mathbf{e_2} \in D_3^n[\alpha_1,\beta_1]$$

$$\left.\begin{array}{rcl} x_{00}+x_{12}+x_{21} &=& 1-\alpha-\beta \\ x_{01}+x_{10}+x_{22} &=& \alpha \\ x_{02}+x_{11}+x_{20} &=& \beta \end{array}\right\} \mathbf{e_1}+\mathbf{e_2} \in D_3^n[\alpha,\beta]$$

Cependant, ces équations ne sont pas linéairement indépendantes, et ce système possède deux degrés de liberté. Toutes les solutions peuvent être

décrites à l'aide de deux paramètres w et z :

$$\begin{array}{lcccccc}
x_{00} & = & 1-\alpha-\beta & - & 2z & & \\
x_{01} & = & \frac{2\alpha+\beta-\alpha_1-2\beta_1}{3} & + & z & + & w \\
x_{02} & = & \frac{\alpha+2\beta-2\alpha_1-\beta_1}{3} & + & z & - & w \\
x_{10} & = & \frac{2\alpha+\beta-\alpha_1-2\beta_1}{3} & + & z & - & w \\
x_{11} & = & \frac{-2\alpha-\beta+4\alpha_1+2\beta_1}{3} & - & 2z & & \\
x_{12} & = & 0 & + & z & + & w \\
x_{20} & = & \frac{\alpha+2\beta-2\alpha_1-\beta_1}{3} & + & z & + & w \\
x_{21} & = & 0 & + & z & - & w \\
x_{22} & = & \frac{-\alpha-2\beta+2\alpha_1+4\beta_1}{3} & - & 2z. & &
\end{array}$$

On cherche à déterminer les paramètres w et z du cas typique, c'est-à-dire les paramètres qui maximisent $\tilde{h}(x_{00}, x_{01}, x_{02}, x_{10}, x_{11}, x_{12}, x_{20}, x_{21}, x_{22})$.

Puisque les vecteurs $\mathbf{e_1}$ et $\mathbf{e_2}$ ont la même distribution, nous somme dans le cas d'application du théorème 2.2, qui nous donne la symétrie du cas typique. On doit donc avoir $x_{01} = x_{10}$, $x_{02} = x_{20}$ et $x_{12} = x_{21}$. Ainsi, dans le cas typique, $w = 0$. Il reste à déterminer z.

Les x_{ij} correspondent à des nombres de coordonnées, et doivent donc être positifs. Ainsi, z doit être compris entre $z_{\min}$ et $z_{\max}$, où

$$z_{\min} = \max\left(0, \frac{\alpha_1+2\beta_1-2\alpha-\beta}{3}, \frac{2\alpha_1+\beta_1-\alpha-2\beta}{3}\right)$$

$$z_{\max} = \min\left(\frac{1-\alpha-\beta}{2}, \frac{-2\alpha-\beta+4\alpha_1+2\beta_1}{6}, \frac{-\alpha-2\beta+2\alpha_1+4\beta_1}{6}\right)$$

On rappelle que les paramètres du cas typique maximisent l'expression $\tilde{h}(x_{00}, x_{01}, x_{02}, x_{10}, x_{11}, x_{12}, x_{20}, x_{21}, x_{22})$. Par définition de $\tilde{h}$, cette expression peut s'écrire $-\sum_{i,j} x_{ij} \log_2(x_{ij})$. La dérivée de cette expression par rapport à z est :

$$2\log_2\left(\frac{x_{00}x_{11}x_{22}}{x_{01}x_{02}x_{12}}\right)$$

ce qui est égal à 0 si et seulement si :

$$x_{01}x_{02}x_{12} = x_{00}x_{11}x_{22}$$

soit :

$$\frac{(2\alpha+\beta-\alpha_1-2\beta_1+3z)(\alpha+2\beta-2\alpha_1-\beta_1+3z)z}{(1-\alpha-\beta-2z)(-2\alpha-\beta+4\alpha_1+2\beta_1-6z)(-\alpha-2\beta+2\alpha_1+4\beta_1-6z)} = 1$$

L'expression $-\sum_{i,j} x_{ij} \log_2(x_{ij})$ étant une fonction strictement concave de z, elle admet un unique maximum. Celui-ci est atteint soit au bord de l'intervalle de définition (en $z_{\min}$ ou en $z_{\max}$), soit en un point d'annulation de la dérivée.

Par ailleurs, si $z = z_{\min}$, alors $z \in \left\{0, \frac{\alpha_1+2\beta_1-2\alpha-\beta}{3}, \frac{2\alpha_1+\beta_1-\alpha-2\beta}{3}\right\}$, et donc x_{01}, x_{02} ou x_{12} est nul. En particulier, $x_{01}x_{02}x_{12} = 0$ et la dérivée de $-\sum_{i,j} x_{ij} \log_2(x_{ij})$ est $+\infty$. D'un autre côté, si $z = z_{\max}$, c'est alors au tour de $x_{00}x_{11}x_{22}$ de s'annuler, et la dérivée de $-\sum_{i,j} x_{ij} \log_2(x_{ij})$ est $-\infty$. Le maximum ne peut donc pas être atteint sur un bord.

Pour obtenir une probabilité, il ne reste plus qu'à diviser le nombre de couples de vecteurs de $D_3^n[\alpha_1, \beta_1]$ qui donnent un vecteur de $D_3^n[\alpha, \beta]$, par le nombre total de couples de vecteurs de $D_3^n[\alpha_1, \beta_1]$. Il y a $\tilde{\mathcal{O}}\left(2^{2\tilde{h}(\alpha_1,\beta_1)n}\right)$ tels couples. □

4.2.2 Représentations partielles

Quand un algorithme utilise la structure d'arbre de 1.2.4, une liste peut être construite de deux façons :

— soit par fusion de deux listes du niveau inférieur avec un découpage gauche-droite. Chaque vecteur admet une seule décomposition, il n'y a alors aucun risque de création de doublons.

— soit par fusion de deux listes du niveau inférieur en utilisant des représentations. Chaque vecteur admet plusieurs représentations. Cela permet de meilleurs algorithmes en augmentant artificiellement le nombre de solutions.

Pour obtenir une bonne complexité, nous aimerions utiliser au maximum les représentations. Cependant, la création de doublons est catastrophique. Nous proposons donc d'utiliser une solution intermédiaire, qui permet une interpolation entre les découpages gauche-droite et les représentations. Cette interpolation sera appelée *représentations partielles*.

Dans le cas d'un découpage gauche-droite, on recherche un vecteur comme la concaténation de sa partie gauche et de sa partie droite. Dans de cas des représentations, on recherche un vecteur comme la somme de deux autres vecteurs. Dans le cas des représentations partielles, on utilise ces deux techniques : on décompose une partie du vecteur à l'aide d'un découpage gauche-droite, et l'autre à l'aide de représentations, comme illustré sur la figure 4.2. La taille relative de ces deux parties est un paramètre libre, qui permet de contrôler le nombre de représentations (et ainsi les nombres d'éléments mal formés et de doublons). En général, on cherchera à maximiser la taille de la partie sur laquelle on utilise des représentations, dans la mesure où cela ne

produit pas de doublons.

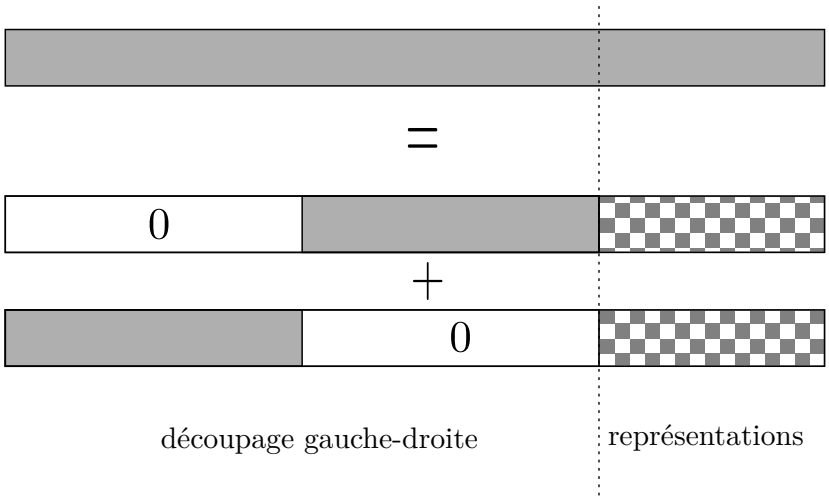

FIGURE 4.2 – Décomposition d'un vecteur en utilisant des représentations partielles

4.3 Décodage dans le pire cas

Nous disposons maintenant de tous les outils nécessaires pour résoudre le problème de décodage ternaire à gros poids. Pour cela, nous commençons par déterminer les paramètres pour lesquels le problème est, à priori, le plus difficile.

Dans le cas binaire, on sait que le cas difficile se trouve à la distance de Gilbert-Varshamov. Dans le cas ternaire à gros poids, ce ω_{GV}^{haut} n'existe que si $k \leqslant \frac{\log_2(3)-1}{\log_2(3)}n$. Cependant, on constate empiriquement que :

— si $k \leqslant \frac{\log_2(3)-1}{\log_2(3)}n$, le poids qui maximise la difficulté du décodage est $\omega = \omega_{GV}^{haut}$. Par ailleurs, le décodage est d'autant plus difficile que k est grand.

— si $k > \frac{\log_2(3)-1}{\log_2(3)}n$, le cas le plus difficile est $\omega = 1$. Dans ce cas, plus k est grand, plus le problème est facile.

Le choix des paramètres pour lesquels le problème de décodage ternaire à gros poids est le plus difficile est donc :

$$\begin{cases} k = \dfrac{\log_2(3) - 1}{\log_2(3)} n \simeq 0.369n \\ \omega = 1 \end{cases}$$

Remarque 4.3. Dans le cas binaire, le pire choix de paramètres dépend de l'algorithme considéré. Dans le cas ternaire à gros poids, de nombreux algorithmes partagent le même pire cas. Cela est dû à l'apparition d'un cas limite pour l'existence de ω_{GV}^{haut}.

Paramètres de l'élimination gaussienne partielle (PGE)

On choisit $\ell = 0.4096n$. On veut alors, dans le sous-problème, trouver des vecteurs de taille $k + \ell = 0.7787n$ et de distribution $D^{k+\ell}[1/2, 0]$.

Par ailleurs, selon la proposition 3.4, la probabilité pour l'un de ces vecteurs d'être une solution au problème de décodage de départ est $\tilde{\mathcal{O}}\left(2^{p_{succ}n}\right)$, où $p_{succ} = -0.1293$. On veut donc trouver $\tilde{\mathcal{O}}\left(2^{\ell_0 n}\right)$ tels vecteurs, pour $\ell_0 = 0.1293$.

Paramètres du sous-problème (SS)

Pour résoudre le sous-problème et trouver ces $\tilde{\mathcal{O}}\left(2^{\ell_0 n}\right)$ candidats, on utilise une structure d'arbre à quatre niveaux. Cette structure est illustrée sur la figure 4.3. On utilise des représentations partielles entre les niveaux 0 à 2, et un découpage gauche/droite entre les niveaux 2 et 3. Plus précisément :

— sur une fraction $r_1 = 0.0233$ du vecteur, on utilise un découpage gauche/droite entre les niveaux 0 et 1, puis des représentations entre les niveaux 1 et 2. On se sert ici de $(D_0, D_1, D_1)-$représentations, où $D_0 = D_3[1/2, 0]$ et $D_1 = D_3[0.2511, 0.0012]$.

— sur le reste, *i.e.* une fraction $r_2 = 1 - r_1 = 0.9767$, on utilise des représentations sur le deux étages, depuis le niveau 0 jusqu'au niveau 2. Pour cela, on passe successivement par les distributions $D_0 = D_3[1/2, 0]$ au niveau 0, puis $D_{2,1} = D_3[0.2539, 0.0043]$ au niveau 1, et enfin $D_{2,2} = D_3[0.1312, 0]$ au niveau 2.

Taille des listes et des contraintes modulaires

Au dernier niveau, le niveau 3, on énumère tous les vecteurs possibles. Une partie de ce vecteur est choisie dans $D_1^{r_1(k+\ell)/4}$, une autre est choisie dans $D_{2,2}^{r_2(k+\ell)/2}$, et le reste est constitué de 0. Une liste du niveau 3 est donc de taille :

$$\ell_3 = (k+\ell)\left(\frac{r_1\tilde{h}(D_1)}{4} + \frac{r_2\tilde{h}(D_{2,2})}{2}\right) = 0.2170$$

Pour obtenir les listes du niveau 2, on fusionne les listes du niveau 3 avec une contrainte modulaire $c_2 = 0.1978$, on obtient alors des listes de taille $\ell_2 = 0.2362$.

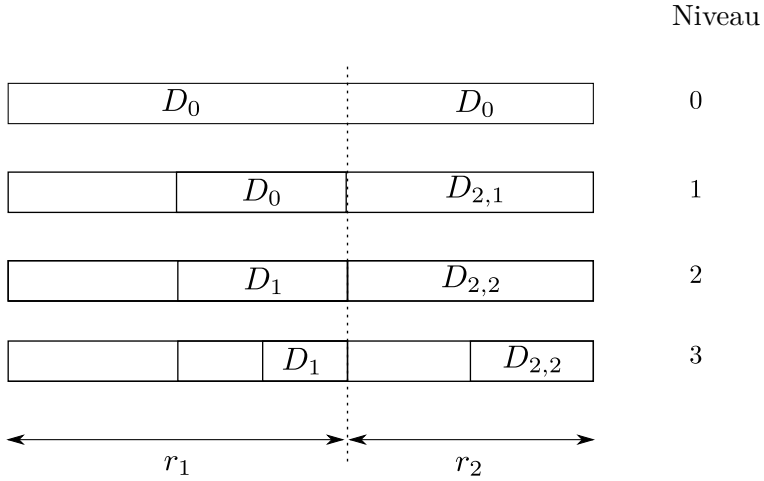

FIGURE 4.3 – Décomposition d'un vecteur en suivant la stratégie pour l'algorithme utilisé dans le pire cas

Pour construire les listes du niveau 1, on ajoute une contrainte modulaire $c_1 - c_2 = 0.2362$. Cependant, parmi les 0.2362 collisions, il y a de nombreux éléments qui ne vérifient pas les distributions imposées. Après avoir éliminé les vecteurs indésirables, il reste des listes de taille $\ell_1 = 0.2257$. En effet, selon la proposition 4.1, la probabilité d'être bien formé pour un vecteur obtenu grâce à des $(D_0, D_1, D_1)-$représentations est $p_{D_0,D_1} = -0.1275$. Selon cette même proposition, on a une probabilité $p_{D_{2,1},D_{2,2}} = -0.0122$ pour les $(D_{2,1}, D_{2,2}, D_{2,2})-$représentations. Ainsi, la probabilité d'être bien formé pour un vecteur obtenu à l'aide des représentations partielles du niveau 4 est :

$$(k+\ell)\left(\frac{r_1 p_{D_0,D_1}}{2} + r_2 p_{D_{2,1},D_{2,2}}\right) = -0.0104$$

puisqu'on utilise des $(D_0, D_1, D_1)-$représentations sur une partie de taille $r_1(k+\ell)/2$, et des $(D_{2,1}, D_{2,2}, D_{2,2})-$représentations sur une partie de taille $r_2(k+\ell)$.

Pour construire les listes du niveau 0, ajoute la contrainte modulaire restante $c_0 - c_1 = 0.2153$ (nécessaire puisqu'on veut $c_0 = \ell \log_2(3)$). Il y a ici encore des éléments mal formés, et on obtient une liste de taille :

$$\ell_0 = 2\ell_1 - (c_0 - c_1) + r_2(k+\ell)p_{D_0,D_{2,1}} = 0.1293$$

Contrôle des doublons

On utilise des représentations pour construire les listes des niveaux 1 et 0. Il faut donc s'assurer qu'on ne crée pas de doublons en calculant, par exemple,

plusieurs copies du même élément du niveau 1, chaque copie provenant d'une représentation différente. On vérifie cela à l'aide du théorème 2.1 :

Un vecteur du niveau 1 est composé de la distribution D_0 sur une partie de taille $r_1(k+\ell)/2$, de la distribution $D_{2,1}$ sur une partie de taille $r_2(k+\ell)$, et de 0 partout ailleurs. L'entropie d'une telle distribution est :

$$\tilde{h}_1 = \frac{k+\ell}{2}\left(r_1\tilde{h}(D_0) + 2r_2\tilde{h}(D_{2,1})\right) = 0.6598$$

Comme par ailleurs les listes de ce niveau sont de taille $\ell_1 = 0.2257$, et que ses vecteurs vérifient une contrainte modulaire $c_1 = 0.4340$, on constate que la contrainte :

$$\ell_1 \leqslant \tilde{h}_1 - c_1$$

est satisfaite. On a même ici une égalité, ce qui signifie que la contrainte modulaire et le nombre de représentations se compensent exactement.

Un vecteur du niveau 0 est également obtenu à l'aide de représentations partielles, et il faut donc vérifier qu'on n'y crée pas de doublons. Un vecteur de niveau 0 est de taille $k+\ell$ et de distribution D_0. L'entropie de cette distribution est : $\tilde{h}_0 = 0.7787$. Comme on a par ailleurs $\ell_0 = 0.1293$ et $c_0 = 0.6492$, on constate que l'inégalité :

$$\ell_0 \leqslant \tilde{h}_0 - c_0$$

est vérifiée. C'est ici aussi une égalité. C'est d'ailleurs pour cela qu'on utilise seulement des représentations partielles : on n'a pas le droit de mettre des représentations sur toute la longueur du vecteur.

Complexité

Au niveau de la réduction, nous avons calculé qu'il faut trouver $-p_{succ} = 0.1293$ solutions au sous-problème pour décoder le syndrome. Puisque le sous-problème nous fournit une liste de $\ell_0 = 0.1293$ vecteurs, l'espérance du nombre d'appels au sous-problème est polynomial.

Dans le problème réduit, nous avons équilibré les listes et les contraintes modulaires de façon à ce que toutes les fusions prennent un temps $\tilde{\mathcal{O}}\left(2^{0.2362n}\right)$. Lors de la construction de certaines listes à l'aide de représentations, on crée des éléments mal formés. Cela est pris en compte, $\tilde{\mathcal{O}}\left(2^{0.2362n}\right)$ étant le temps requis pour produire tous les éléments, bien formés ou non. La construction des listes du niveau 3 par l'énumération de tous les vecteurs possibles prend un temps plus court.

La complexité en temps de cet algorithme est donc $\tilde{\mathcal{O}}\left(2^{0.2362n}\right)$.

Comparaison des pires cas binaire et ternaire

La table 4.4 compare la difficulté du décodage dans le cas binaire et dans le cas ternaire, pour des algorithmes similaires. Dans le cas binaire, il s'agit

du décodage à $\omega = \omega_{GV}$, et le pire ratio de code $R = k/n$ est indiqué. Dans le cas ternaire, le pire cas est le même pour tous les algorithmes présentés ($R = \frac{\log_2(3)-1}{\log_2(3)}, \omega = 1$).

Algorithme	$q = 2$	$q = 3$ et $\omega \geqslant \frac{q+R-1}{q}$
Prange	0.121 ($R = 0.454$)	0.369
Dumer/Wagner	0.116 ($R = 0.447$)	0.269
[BJMM12]/4.3	0.102 ($R = 0.427$)	0.2362

FIGURE 4.4 – Exposants de complexité de différents algorithmes dans le pire cas, pour $q = 2$ et $q = 3$

4.4 Décodage à gros poids sous ω_{GV}^{haut}

Nous cherchons à présent à créer de bons algorithmes pour résoudre le problème de décodage dans des zones de paramètres similaires à ceux de Wave. Pour ce type de paramètres, le poids est en dessous de ω_{GV}^{haut} : il y a donc beaucoup de solutions.

Comme dans le cas de l'algorithme de Wagner, le nombre optimal de niveaux dépend de la densité des solutions : plus la densité de solutions est élevée, plus on peut se permettre d'ajouter des niveaux.

Concernant les représentations, il est préférable, dans le cas présent, de n'en utiliser que dans les niveaux les plus profonds. En effet, les éléments mal formés sont trop coûteux si on utilise des représentations plus haut dans l'arbre. Pour un algorithme comportant $a + 1$ niveaux (numérotés de 0 à a), on utilisera éventuellement des représentations entre les niveaux $a - 3$ et $a - 2$, et entre les niveaux $a - 2$ et $a - 1$. Pour les niveaux 0 à $a - 3$, le coût des éléments mal formés est prohibitif, tandis que des représentations entre les niveaux a et $a - 1$ produiraient des doublons.

Il y a un certain nombre de façons de choisir où et comment utiliser des représentations. Nous allons présenter quatre stratégies possibles. En fonction des paramètres, chacune de ces stratégies peut être soit meilleure que les trois autres, soit sous-optimale, ou même impossible. La figure 4.5 illustre les complexités de ces différentes stratégies pour des paramètres particuliers. Nous allons décrire les stratégies en partant du point limite A, et en diminuant progressivement la contrainte de poids ω (et donc en augmentant le nombre de solutions).

4.4.1 Cas limite A

Dans ce cas particulier, les listes du niveau a font la moitié de la taille des listes du niveau $a - 1$. La contrainte de modulo pour passer du niveau

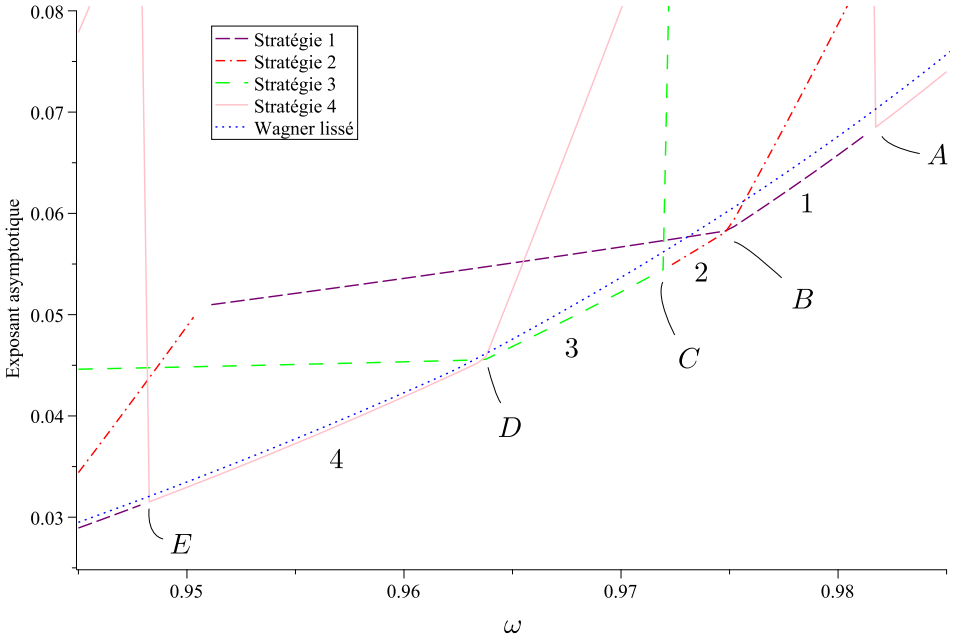

FIGURE 4.5 – Comparaison des exposants de complexité asymptotique des différentes stratégies et de l'algorithme de Wagner lissé, en fonction du poids ω, pour un ratio $R = k/n = 0.6$.

a au niveau $a-1$ est donc $c_{a-1} = 0$. Puisqu'il n'y a aucun filtrage entre les niveaux a et $a-1$, nous ne pouvons pas mettre de représentations entre les niveaux $a-1$ et $a-2$.

En revanche, il y a un filtrage non trivial entre les niveaux $a-1$ et $a-2$, ce qui permet d'utiliser des représentations entre les niveaux $a-2$ et $a-3$. Cependant, la condition de filtrage c_{a-2} n'étant pas suffisante pour utiliser des représentations sur la totalité du vecteur, nous nous contentons d'en mettre autant que possible à l'aide de représentations partielles.

4.4.2 Stratégie 1

Si nous diminuons ω, nous augmentons le nombre de solutions. Les listes du dernier niveau (le niveau a) sont alors de plus en plus grandes. Par conséquent, les contraintes de filtrage sont également plus grandes, ce qui permet d'utiliser plus de représentations.

Pour cette stratégie, un vecteur du niveau $a-3$ est décomposé en trois parties, à savoir :

— une partie sur laquelle on utilise des représentations depuis le niveau $a-3$ jusqu'au niveau $a-1$. Pour maximiser le nombre de représentations, nous voulons cette partie la plus grande possible. Cependant, elle est

limitée par la condition de filtrage c_{a-1}, assez faible, entre les niveaux a et $a-1$.

— une partie sur laquelle on utilise des représentations entre les niveaux $a-3$ et $a-2$, et un découpage gauche-droite entre les niveaux $a-2$ et $a-1$. Pour avoir un maximum de représentations, nous en ajoutons jusqu'à saturer également la contrainte de filtrage c_{a-2} entre les niveaux $a-1$ et $a-2$.

— une partie sur laquelle on ne fait que du découpage gauche-droite.

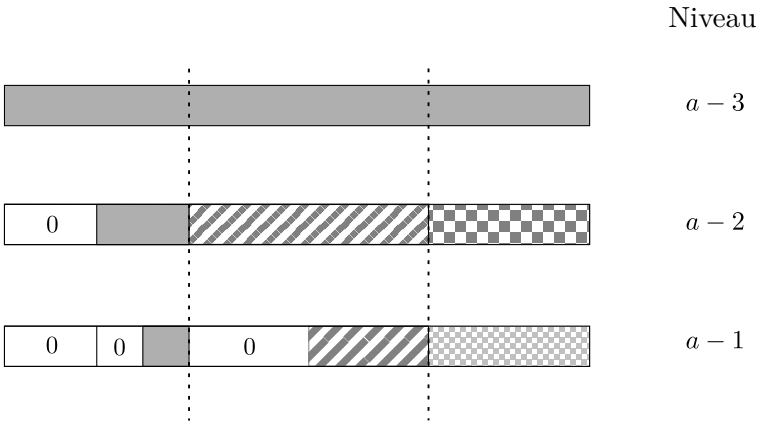

FIGURE 4.6 – Décomposition d'un vecteur en suivant la stratégie 1

Plus nous diminuons ω, plus la partie sans représentations est petite, jusqu'à disparaître complètement. C'est le cas limite B. À partir de ce point, c_{a-2} est suffisamment grand pour ne plus être contraignant : il y a des représentations sur toute la longueur du vecteur entre les niveaux $a-2$ et $a-3$.

4.4.3 Stratégie 2

Nous continuons à diminuer ω. Les contraintes de modulo sont de plus en plus grandes et autorisent de plus en plus de représentations entre les niveaux $a-1$ et $a-2$. Nous augmentons donc progressivement la fraction du vecteur sur laquelle il y a deux étages de représentations.

Cela faisant, la partie sur laquelle il n'y a de représentations qu'entre les étages $a-2$ et $a-3$ est de plus en plus petite. Quand elle disparaît entièrement, et qu'il y a des représentations sur tout le vecteur entre les étages $a-1$ et $a-3$, nous avons atteint le cas limite C.

FIGURE 4.7 – Décomposition d'un vecteur en suivant la stratégie 2

4.4.4 Stratégie 3

Nous diminuons encore ω, et nous ajoutons encore des solutions. Nous avons tellement de solutions que nous pouvons nous passer d'une partie des représentations, maintenant superflues : nous devrions payer le coût des éléments mal formés pour des représentations qui ne nous profitent plus. Au fur et à mesure que nous disposons de plus de solutions, nous diminuons progressivement le nombre de représentations, en enlevant en priorité celles qui se trouvent entre les niveaux $a-2$ et $a-3$, plus coûteuses en éléments mal formés.

En suivant cette stratégie, un vecteur du niveau $a-3$ est décomposé en deux parties :

— une partie sur laquelle on n'utilise de représentations qu'entre les niveaux $a-1$ et $a-2$.

— une partie sur laquelle on utilise des représentations depuis le niveau $a-3$ jusqu'au niveau $a-1$. Cette partie crée plus de représentations (et d'éléments mal formés) que l'autre partie. Elle doit être suffisamment grande pour nous fournir assez de représentations. Elle ne doit cependant pas être plus grande que nécessaire, car les représentations superflues créent des éléments mal formés, coûteux et inutiles.

Dans un vecteur de niveau $a-2$, il n'y a pas la même densité de 0, 1 et 2 suivant si on a utilisé des représentations ou un découpage gauche-droite entre les niveaux $a-2$ et $a-3$. En particulier, si on a utilisé un découpage gauche-droite, on aura plus de représentations entre les niveaux $a-1$ et $a-2$ que si on avait déjà utilisé des représentations entre les niveaux $a-2$ et $a-3$.

Par conséquent, quand nous remplaçons progressivement les représentations sur deux étages par des représentations uniquement entre les niveaux $a-1$ et $a-2$, il y a globalement moins de représentations *mais* il y a plus

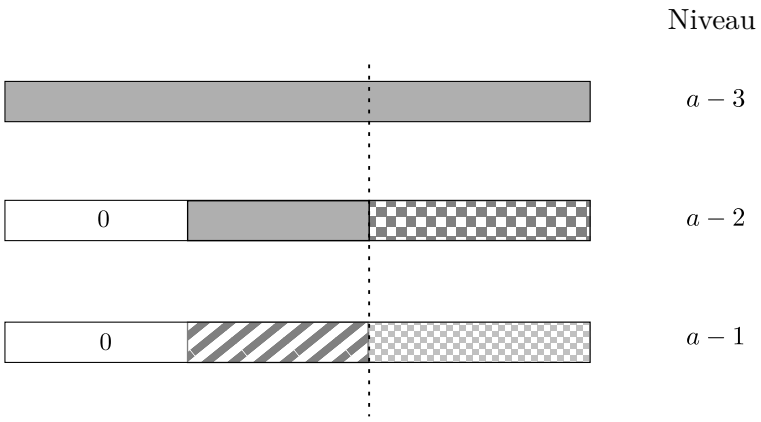

FIGURE 4.8 – Décomposition d'un vecteur en suivant la stratégie 3

de représentations entre les niveaux $a-1$ et $a-2$. Or, pour ne pas faire apparaître de doublons, nous ne pouvons pas mettre plus de représentations que la contrainte de modulo c_{a-1}.

Quand le nombre de représentations entre les niveaux $a-1$ et $a-2$ atteint c_{a-1}, c'est le cas limite D.

4.4.5 Stratégie 4

Pour continuer à diminuer le nombre d'éléments mal formés (et donc de représentations), on ne peut plus se contenter d'enlever des représentations entre les niveaux $a-2$ et $a-3$ (qui sont les représentations les plus coûteuses en éléments mal formés) : on doit aussi en enlever entre les niveaux $a-1$ et $a-2$.

Un vecteur du niveau $a-3$ est donc découpé en trois parties :

— une partie sur laquelle on fait du découpage gauche-droite entre les niveaux $a-3$ et $a-2$, et des représentations entre les niveaux $a-2$ et $a-1$. Il faut une certaine quantité de représentations. Nous aimerions en mettre un maximum dans cette partie, puisque les éléments mal formés sont moins gênants qu'au dessus. Cependant, cette partie ne peut pas non plus être trop grande, le nombre de représentations entre les niveaux $a-1$ et $a-2$ étant majoré par c_{a-1}.

— une partie sur laquelle on n'utilise que des représentations entre les niveaux $a-3$ et $a-1$. Cette partie est nécessaire pour avoir un nombre suffisant de représentations.

— une partie sur laquelle on ne fait que du découpage gauche-droite. Nous avons assez de représentations avec les deux autres parties, on ne fait que du découpage gauche-droite pour ne pas créer d'éléments mal formés inutiles.

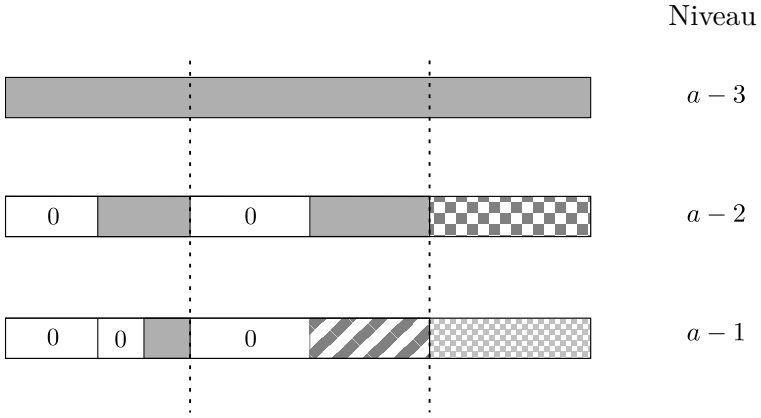

FIGURE 4.9 – Décomposition d'un vecteur en suivant la stratégie 4

Quand il n'y a plus de représentations sur deux étages, nous avons atteint le cas limite E. Dans ce cas, nous utilisons du découpage gauche-droite partout, sauf entre les niveaux $a-1$ et $a-2$ où il y a des représentations partielles.

Appliquons ici une transformation à priori inutile. De façon similaire à l'algorithme de Schroeppel et Shamir, nous ajoutons un étage en faisant un découpage gauche-droite sur les listes du niveau a (qui devient alors l'avant-dernier niveau).

Nous constatons que la structure obtenue correspond exactement à celle du cas limite A, mais avec un étage de plus. Si nous continuons à diminuer ω, nous pouvons donc reprendre la stratégie 1 avec un étage de plus, en ainsi de suite.

4.5 Complexité dans le cas de Wave

La principale motivation de l'étude du problème du décodage de syndrome dans le cas ternaire, pour des grands poids, est le schéma de signature Wave, introduit par [DST19]. Une façon d'attaquer ce schéma est de résoudre un problème de décodage de syndrome ternaire, pour un ratio de code $R = k/n = 0.6089$ et un poids $\omega = 0.9261$. Pour ces paramètres, il y a beaucoup de solutions. Aussi, les algorithmes de Prange ou de Dumer sont assez peu efficaces. En revanche, en utilisant l'algorithme de Wagner, on peut profiter du nombre élevé de solutions en construisant un arbre très profond. Comme on peut le constater sur la figure 4.10, l'ajout de représentations partielles permet d'obtenir une amélioration, mais celle-ci est très faible.

Pour les paramètres de Wave, le meilleur algorithme obtenu utilise la stratégie 4 avec 7 niveaux. Sa complexité est $\tilde{\mathcal{O}}\left(2^{0.0155n}\right)$. Dans ce qui suit, nous allons détailler les paramètres de cet algorithme.

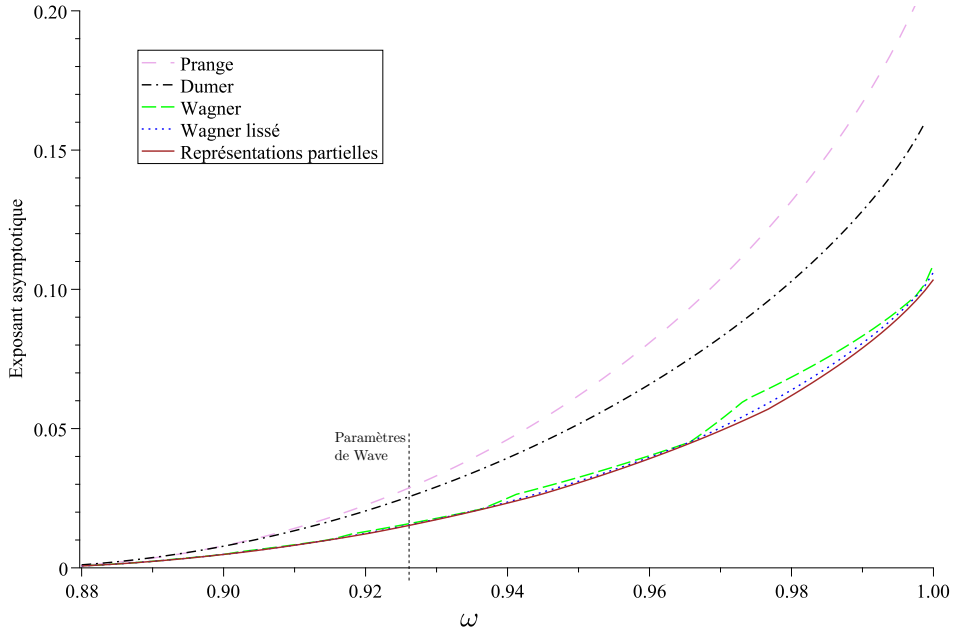

FIGURE 4.10 – Comparaison de l'exposant de complexité asymptotique de différents algorithmes en fonction de ω, pour $R = k/n = 0.676$.

Paramètres de l'élimination gaussienne partielle (PGE)

On choisit $\ell = 0.0533n$. On veut alors, dans le sous-problème, trouver des vecteurs de taille $k + \ell = 6622n$, et de distribution $D^{k+\ell}[1/2, 0]$. Par ailleurs, la probabilité pour l'un de ces vecteurs d'être une solution au problème de décodage de départ est $p_{succ} = -0.0155$. On veut donc trouver $\ell_0 = 0.0155$ tels vecteurs.

Paramètres du sous-problème (SS)

Comme expliqué précédemment dans la description de la stratégie 4, les fusions de listes se font de la façon suivante :

— entre les niveaux 6 et 5 : décomposition gauche-droite (comme toujours au dernier niveau)

— entre les niveaux 5 et 3 : représentations partielles, comme illustré sur la figure 4.11

— entre les niveau 3 et 0 : décomposition gauche-droite.

Au niveau 0, on sait qu'il faut trouver des vecteurs de distribution : $D_0^{k+\ell} = D_3^{k+\ell}[1/2, 0]$. Ensuite, puisqu'on utilise uniquement des décompositions gauche-droite jusqu'au niveau 3, les vecteurs de ce niveau sont composés d'une partie à $D_0^{(k+\ell)/8}$, et de 0 partout ailleurs. Arrivé au niveau 3, un

vecteur de $D_0^{(k+\ell)/8}$ est décomposé à l'aide de représentations partielles :

— sur une fraction $r_1 = 0.6661$ du vecteur, on utilise des représentations uniquement entre les niveaux 5 et 4. Plus précisément on utilise des $(D_0, D_1, D_1)-$représentations, où $D_0 = D_3[1/2, 0]$ et $D_1 = D_3[0.2511, 0.0012]$.

— sur une fraction $r_2 = 0.0493$ du vecteur, on utilise des représentations du niveau 3 au niveau 5. Par cela, on passe successivement par les distributions $D_0 = D_3[1/2, 0]$ au niveau, puis $D_{2,1} = D_3[0.2539, 0.0043]$ au niveau 4, et enfin $D_{2,2} = D_3[0.1312, 0]$ au niveau 5.

— sur le reste, on n'utilise de représentations nulle part.

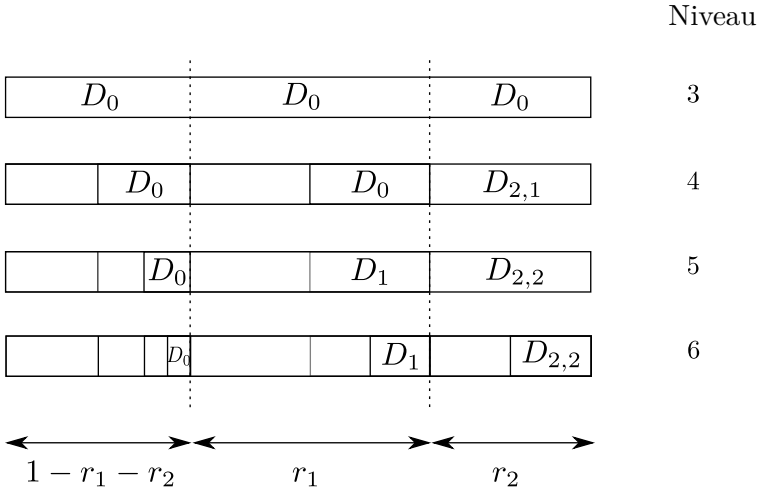

FIGURE 4.11 – Répartition des représentations partielles en suivant la stratégie adoptée pour les paramètres de Wave.

Taille des listes et des contraintes modulaires

Au dernier niveau, le niveau 6, on énumère tous les vecteurs possibles. Une partie de ce vecteur est choisie dans $D_0^{(1-r_1-r_2)(k+\ell)/64}$, une autre partie est choisie dans $D_1^{r_1(k+\ell)/32}$, une troisième partie provient de $D_{2,2}^{r_2(k+\ell)/16}$, et le reste est constitué de 0. Une liste du niveau 6 est donc de taille :

$$\ell_6 = (k+\ell)\left(\frac{(1-r_1-r_2)\tilde{h}(D_0)}{64} + \frac{r_1\tilde{h}(D_1)}{32} + \frac{r_2\tilde{h}(D_{2,2})}{16}\right) = 0.0155$$

Pour obtenir les listes du niveau 5, on fusionne les listes du niveau 6 avec une contrainte modulaire $c_5 = 0.0155$, on obtient alors des listes de taille $\ell_5 = 0.0155$.

Pour construire les listes du niveau 4, on ajoute une nouvelle contrainte de modulo $c_4 - c_5 = 0.0155$. Cependant, parmi les $\tilde{\ell}_4 = 0.0155$ collisions obtenues, il y a de nombreux éléments qui ne vérifient pas les distributions imposées. La probabilité pour une de ces collisions d'être bien formée est :

$$p_4 = (k+\ell)\left(\frac{r_1 p_{D_0,D_1}}{16} + \frac{r_2 p_{D_{2,1},D_{2,2}}}{8}\right) = -0.0036$$

Cela donne des listes de taille :

$$\ell_4 = \tilde{\ell}_4 + p_4 = 0.0119$$

Pour construire les listes du niveau 3, on choisit une contrainte de modulo $c_3 - c_4 = 0.00834$. Cette valeur est prise de façon à ce que les listes du niveau 3 soient à nouveau de taille 0.0155. Cependant, il y a ici aussi des éléments mal formés, et on obtient seulement des listes de taille :

$$\ell_3 = 2\ell_4 - (c_3 - c_4) + (k+\ell)\frac{r_2 p_{D_0,D_{2,1}}}{8} = 0.0149$$

Pour construire les listes du niveau 2, on prend pour contrainte modulaire : $c_2 - c_3 = 0.0142$. Ainsi, les listes du niveau 2 sont de taille $\ell_2 = 0.0155$. Après cela, il n'y a plus de représentations. On choisit donc des contraintes modulaires additionnelles 0.0155, ce qui assure que les listes sont également de taille $\ell_1 = \ell_0 = 0.0155$.

Contrôle des doublons

On utilise des représentations pour construire les listes des niveaux 4 et 3. On fait appel au théorème 2.1 pour s'assurer qu'on ne crée pas de doublons.

Au niveau 4, on a :

$$\begin{aligned} \tilde{h}_4 &= (k+\ell)\left(\tfrac{(1-r_2)\tilde{h}(D_0)}{16} + \tfrac{r_2\tilde{h}(D_{2,1})}{8}\right) = 0.0428 \\ \ell_4 &= 0.0119 \\ c_4 &= 0.0309 \end{aligned}$$

L'inégalité $\ell_4 \leqslant \tilde{h}_4 - c_4$ est bien satisfaite (on a même une égalité.)

Au niveau 3 :

$$\begin{aligned} \tilde{h}_3 &= \tfrac{(k+\ell)\tilde{h}(D_0)}{8} = 0.0828 \\ \ell_3 &= 0.0149 \\ c_3 &= 0.0393 \end{aligned}$$

La contrainte $\ell_3 \leqslant \tilde{h}_3 - c_3$ est vérifiée. L'inégalité est même très large, ce qui était parfaitement attendu, puisqu'on n'utilise des représentations que sur une partie r_2 très faible du vecteur.

Cohérence des paramètres

Les contraintes modulaires ont été choisies pour permettre à toutes les fusions de listes de s'effectuer en temps $\tilde{\mathcal{O}}\left(2^{0.0155n}\right)$. Cependant, nous avons une autre contrainte : le sous-problème doit trouver des vecteurs $\overline{\mathbf{e}}^{(\mathbf{3})}$ satisfaisant une condition de la forme $\overline{\mathbf{H}}''\overline{\mathbf{e}}^{(\mathbf{3})} = \overline{\mathbf{s}}^{(\mathbf{3})}$, c'est-à-dire une condition sur $\ell = 0.0533$ coordonnées.

La condition modulaire au niveau 0 doit donc correspondre à une condition sur ℓ coordonnées. En additionnant les contraintes ajoutées à chaque étage, on constate que les contraintes de modulo totales sont, de c_5 à c_0 :

$$0.0155, 0.0309, 0.0393, 0.0536, 0.0691, 0.0845$$

Ainsi, c_0 est bien une contrainte sur 0.0845 bits, c'est-à-dire une contrainte sur $0.0533 = \ell$ coordonnées ternaires.

Complexité

Selon le théorème 3.6, la complexité est $\tilde{\mathcal{O}}\left(2^{(t+\max(0,-\ell_0-p_{succ}))n}\right)$. On sait déjà que $p_{succ} = -0.0155$ et que $\ell_0 = 0.0155$. Par ailleurs, le calcul de chaque liste prend un temps $\tilde{\mathcal{O}}\left(2^{0.0155n}\right)$, que ce soit par énumération (au niveau 6) ou par fusion (aux autres niveaux). La complexité de la sous-routine est alors $\tilde{\mathcal{O}}\left(2^{tn}\right) = \tilde{\mathcal{O}}\left(2^{0.0155n}\right)$.

Finalement, la complexité en temps de cet algorithme est donc :

$$\tilde{\mathcal{O}}\left(2^{0.0155n}\right)$$

Remarque 4.4. Cet algorithme de décodage de syndrome sur les paramètres de Wave ne correspond pas à ce qui peut être trouvé dans [BCDL19]. En effet, les paramètres de Wave ont été modifiés depuis. J'ai ici utilisé les paramètres définitifs (c'est-à-dire ceux de [DST19]).

4.6 Décodage par un algorithme quantique

Lorsque nous avons étudié le problème de remplissage de sac à dos, nous avons considéré les accélérations possibles pour quelqu'un disposant de ressources quantiques. Naturellement, puisque nous réduisons le problème du décodage de syndrome à un problème de remplissage de sac à dos, il est également possible d'accélérer le décodage de syndrome en utilisant des algorithmes quantiques. Nous allons donc reprendre les deux cas précédents (le pire cas et les paramètres de Wave), et proposer des algorithme quantiques pour les résoudre.

4.6.1 Décodage quantique dans le pire cas

Nous voulons résoudre le problème de décodage de syndrome à l'aide d'un algorithme quantique pour les paramètres du pire cas :

$$\begin{cases} k = \dfrac{\log_2(3) - 1}{\log_2(3)} n \\ \omega = 1 \end{cases}$$

Pour la réduction à un problème de remplissage de sac à dos, on choisit $\ell = 0.2185n$. Par conséquent, il faut tester $\mathbb{P}_{succ} = \tilde{\mathcal{O}}\left(2^{p_{succ}n}\right)$ candidats pour trouver une solution, où $p_{succ} = -0.2413$. Il reste maintenant à décrire l'algorithme de remplissage de sac à dos. Pour cette étape, on utilise des représentations partielles sur deux niveaux comme représenté sur la figure 4.12.

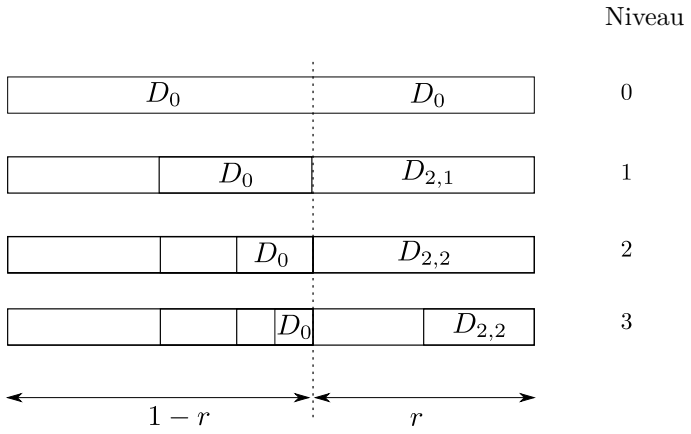

FIGURE 4.12 – Décomposition d'un vecteur pour l'algorithme quantique dans le pire cas

Dans le cas classique, à cause des représentations entre les niveaux 1 et 0, on ne pouvait pas tester les candidats en temps amorti $\tilde{\mathcal{O}}(1)$. Dans le cas quantique, de la même façon, on ne va pas pouvoir tester des solutions en temps racine, l'algorithme de remplissage de sac à dos va tester ces $2^{0.2413n}$ candidats en temps $\tilde{\mathcal{O}}\left(2^{0.1368n}\right)$.

Les densités $D_0, D_{2,1}$ et $D_{2,2}$ sont les mêmes que celles utilisées dans le cas classique. On utilise des représentations sur deux niveaux sur une proportion $r = 0.7212$ du vecteur.

— au niveau 3, on crée des listes gauches de taille $\ell_3^l = 0.1368$. Ces listes sont obtenues en échantillonnant un espace de taille 0.1710. Les listes droites sont de taille $\ell_3^r = 0.1076$ et sont construites par énumération d'un espace de taille 0.1076.

— au niveau 2, à l'aide d'une contrainte modulaire $c_2 = 0.1076$, on obtient des listes de taille $\ell_2 = 0.1368$.

— au niveau 1, on ajoute une contrainte $c_1 - c_2 = 0.1342$. À cause des représentations, les listes de ce niveau ne sont que de taille $\ell_1 = 0.1342$.

— au niveau 0, on complète par une condition modulaire $c_0 - c_1 = \ell \log_2(3) - c_1 = 0.1045$. On se retrouve alors avec une liste de taille $\ell_0 = 0.1045$.

Le temps de l'étape de SETUP est $\mathsf{S} = \tilde{\mathcal{O}}\left(2^{0.1368}\right)$, ce qui correspond au temps de construction des listes des niveau 1, 2 et des listes gauches du niveau 3. Pour le niveau 1, il faut utiliser le filtrage quantique décrit en 2.3.1. Le temps demandé par la liste du niveau 0 et les listes droites du niveau 3 sont négligeable devant le temps requis par les autres listes.

On a choisi les tailles des listes et des contraintes modulaires de façon à ce que l'UPDATE s'exécute en temps $\tilde{\mathcal{O}}(1)$. L'étape de CHECK consiste à chercher un élément dans la liste du niveau 0, qui est de taille $\ell_0 = 0.1045$, ce qui donne $\mathsf{C} = \tilde{\mathcal{O}}\left(2^{0.0522n}\right)$.

Par ailleurs, on a $\varepsilon = \tilde{\mathcal{O}}\left(2^{-0.1368n}\right)$ et $\delta = \tilde{\mathcal{O}}\left(2^{-0.1368n}\right)$, ce qui donne une complexité totale :

$$\mathsf{S} + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C}\right) = \tilde{\mathcal{O}}\left(2^{0.1368n}\right)$$

4.6.2 Décodage quantique avec beaucoup de solutions

Si on se place en dessous de ω_{GV}^{haut}, le problème de décodage admet de nombreuses solutions. C'est notamment le cas des paramètres utilisés pour le schéma de signature Wave. Comme expliqué dans la section 4.4, la meilleure façon connue pour résoudre le sous-problème de remplissage de sac à dos pour ce type de paramètres dans le cas classique utilise une structure d'arbre avec de nombreux niveaux, et des représentations partielles aux niveaux inférieurs.

Par ailleurs, nous avons vu, dans la section 3.3 et particulièrement le lemme 3.8 comment utiliser l'algorithme de Wagner dans le cas quantique. Malheureusement, il n'est pas possible d'ajouter facilement des représentations partielles à la variante quantique de l'algorithme de Wagner. En effet, dans les algorithmes de la section 4.4, certaines listes sont plus petites que les listes des niveau supérieurs. Par exemple, dans l'algorithme détaillé dans 4.5, on a $\ell_4 = 0.0119$ tandis que $\ell_3 = 0.0149$. Dans une marche quantique telle que celles utilisées dans 2.3, cela rend l'étape d'UPDATE beaucoup trop longue.

Aussi, pour échapper à cette difficulté, nous allons simplement ignorer les représentations dans le cas présent. Cela est également motivé par le fait que les représentations n'apportaient qu'une faible d'amélioration (voir figure 4.10).

Nous utilisons alors directement le lemme 3.8. Dans le cas présent, la taille de l'espace de recherche est $\Omega = k + \ell$ (on cherche des vecteurs de taille $k + \ell$, et pour chaque coordonnée on a le choix entre un 0 et un 1). La contrainte modulaire à vérifier est $c = \ell \log_2(3)$. L'algorithme de Wargner peut alors tester $2^{2\lambda}$ candidats en temps $\tilde{\mathcal{O}}\left(2^{\lambda}\right)$, pour le λ donné par le lemme 3.8.

Par ailleurs, pour espérer trouver une solution, il faut tester $1/\mathbb{P}_{succ}$ candidats, où $\log_2(\mathbb{P}_{succ}) = (n-k-\ell)\left(h\left(\frac{\omega n-k-\ell}{n-k-\ell}\right) - \log_2(3)\right) + \omega n - k - \ell$. On choisit alors ℓ de façon à ce que $\log_2(\mathbb{P}_{succ}) = -2\lambda$.

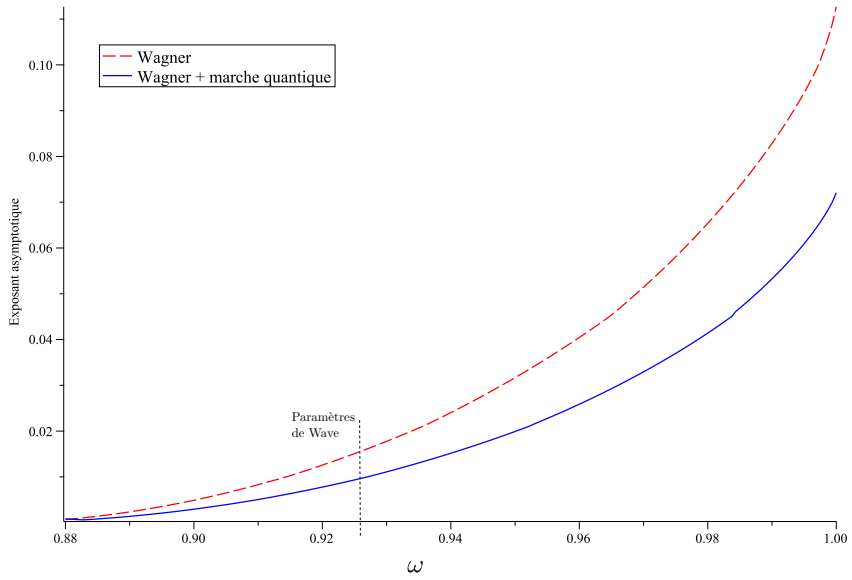

FIGURE 4.13 – Comparaison des exposants de complexité asymptotique classique et quantique

Dans la figure 4.13, nous comparons la complexité asymptotique de l'algorithme de Wagner dans le cas classique (en pointillés rouge), et du même algorithme combiné à une marche quantique (ligne pleine bleue).

Par exemple, dans le cas des paramètres de Wave, l'algorithme de Wagner donne une complexité $\tilde{\mathcal{O}}\left(2^{0.0157n}\right)$ (ou $\tilde{\mathcal{O}}\left(2^{0.0155}\right)$ avec des représentations partielles). Avec une marche quantique, on pourrait obtenir une complexité $\tilde{\mathcal{O}}\left(2^{0.0097n}\right)$.

Chapitre 5

DOOM

Pour certains protocoles de signature (par exemple Wave), réussir à forger une signature pour un message donné revient à résoudre une instance d'un problème de décodage de syndrome. Cependant, d'autres attaques existent contre ce type de protocole : au lieu d'essayer de forger une signature pour un message particulier, on peut chercher à forger une signature pour un message au choix dans une liste.

Ce problème revient à résoudre une variante du problème de décodage de syndrome : le problème DOOM (Decode One Out of Many).

5.1 Définition du problème

5.1.1 Nombre fini de syndromes

La différence entre le problème DOOM et le problème de décodage de syndrome est simple : au lieu de nous donner en entrée un syndrome à décoder, il nous est donné une liste de syndromes, et il faut réussir à décoder l'un d'entre eux.

Problème : Decoding One Out of Many (fini)

Entrée : $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$ matrice de parité
$\mathbf{s_1}, \ldots, \mathbf{s_z} \in \mathbb{F}_2^{n-k}$ syndromes
$\omega \in [0, 1/2]$ contrainte de poids

où $\forall i, \mathbf{s_i} = \mathbf{He_i}$ pour un certain $\mathbf{e_i}$ secret, de poids $W := \lceil \omega n \rceil$

Sortie : $\begin{cases} \mathbf{e} \in \mathbb{F}_2 \\ i \in [\![1, z]\!] \end{cases}$ tels que $\begin{cases} \mathbf{He} = \mathbf{s_i} \\ |\mathbf{e}| = \lceil \omega n \rceil \end{cases}$

Remarque 5.1. Ce problème est au moins aussi facile que le décodage de syndrome : on peut utiliser n'importe quel algorithme de décodage de syndrome pour décoder le premier élément de la liste.

5.1.2 Nombre infini de syndromes

Cependant, cette définition du problème DOOM n'est pas satisfaisante : le nombre de syndromes n'a pas de raison particulière d'être fixé par avance. En pratique, le nombre z de syndromes serait donné par le nombre de messages parmi lesquels on veut forger une signature lors d'une attaque. Par prudence, on veut disposer d'une sécurité contre n'importe quelle attaque de ce type, quel que soit le nombre de messages.

D'autre part, donner une très grande liste de messages en entrée est ambigu : si cette liste devient trop longue, la simple lecture de l'entrée peut devenir une borne sur la complexité de l'algorithme. On aurait pu pallier ce problème en supposant qu'il n'est pas obligatoire de lire la totalité de la liste de syndromes. On préférera la formulation suivante :

Problème : Decoding One Out of Many (infini)

Entrée : $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$ matrice de parité
$\mathbf{s} : \mathbb{N} \longrightarrow \mathbb{F}_2^{n-k}$ fonction aléatoire générant des syndromes
$\omega \in [0, 1/2]$ contrainte de poids

où $\forall i, \mathbf{s}(i) = \mathbf{He_i}$ pour un certain $\mathbf{e_i}$ secret, de poids $W := \lceil \omega n \rceil$

Sortie : $\begin{cases} \mathbf{e} \in \mathbb{F}_2 \\ i \in \mathbb{N} \end{cases}$ tels que $\begin{cases} \mathbf{He} = \mathbf{s}(i) \\ |\mathbf{e}| = \lceil \omega n \rceil \end{cases}$

On suppose que l'oracle $\mathbf{s}$ est une fonction aléatoire parfaite qui s'exécute en temps polynomial en n. Un algorithme de résolution du problème DOOM

peut donc demander une liste de syndromes aussi grande qu'il le veut, pour un coût en complexité directement lié à la taille de cette liste.

Les syndromes générés par cette fonction $\mathbf{s}$ sont aléatoires, tirés uniformément et indépendamment dans l'image par $\mathbf{H}$ de l'ensemble des vecteurs de poids ω. La liste des syndromes demandés par l'algorithme ne présente donc ni structure, ni corrélations.

5.1.3 Le problème en moyenne

De même que pour le remplissage de sac à dos et le décodage simple, nous étudierons la complexité de ce problème en moyenne. Plus précisément, nous fixons le ratio $R = k/n$ ainsi que le poids ciblé ω. Nous cherchons alors un algorithme qui résout efficacement le problème DOOM en moyenne sur tous les choix possibles pour $\mathbf{H}$ et pour les $\mathbf{e_i}$.

5.2 Algorithmes de résolution du problème DOOM

Nous disposons d'un nombre illimité de syndromes (en pratique limité par le temps requis pour les générer), et nous devons réussir à décoder n'importe lequel. Nous allons maintenant voir comment exploiter cette abondance.

5.2.1 Collisions naïves

Une façon simple de faire serait de rechercher naïvement des collisions entre une liste de vecteurs d'erreur et une liste de syndromes.

Considérons une paire $(\mathbf{e}, i)$, avec $\mathbf{e}$ choisi parmi les vecteurs de poids ω, et i choisi n'importe comment dans $\mathbb{N}$, et calculons la probabilité que $\mathbf{He} = \mathbf{s}(i)$.

À la fois $\mathbf{He}$ et $\mathbf{s}(i)$ appartiennent à l'image par $\mathbf{H}$ de l'ensemble des vecteurs de poids ω, lui-même inclus dans $\mathbb{F}_2^{n-k}$. $\mathbf{He}$ et $\mathbf{s}(i)$ appartiennent donc tous les deux à un ensemble dont le cardinal est borné par 2^{n-k} (le taille de $\mathbb{F}_2^{n-k}$), et par $\tilde{\mathcal{O}}(2^{nh(\omega)})$ (le nombre de vecteurs de poids ω). La probabilité d'avoir $\mathbf{He} = \mathbf{s}(i)$ est donc au moins $\tilde{\mathcal{O}}(2^{p_{succ}})$, où $p_{succ} = \max(-(n-k), -nh(\omega))$.

Ainsi, si on prend une liste de $2^{-p_{succ}/2}$ vecteurs $\mathbf{e}$ de poids ω, et une liste d'autant d'entiers i, on peut espérer obtenir une collision $\mathbf{He} = \mathbf{s}(i)$.

Cette façon de chercher des collisions exploite bien la possibilité d'accéder à beaucoup de syndromes. Elle présente néanmoins de nombreux défauts, parmi lesquels :

— Cela donne un algorithme moins performant que les algorithmes de décodage de syndrome simple.

— La liste de syndromes n'est pas nécessaire : on peut obtenir exactement la même complexité en recherchant des collisions entre partie gauche et partie droite (algorithme de Schroeppel et Shamir).

— On n'exploite plus la possibilité du pivot de Gauss (partiel)

— ...

Cependant, si cette idée seule n'est pas suffisante pour profiter du choix du syndrome à décoder, nous pouvons la combiner avec toutes les autres techniques pour améliorer les algorithmes précédents.

5.2.2 Fusion hybride

Dans le paragraphe précédent, nous avons vu qu'il est possible de chercher des collisions entre une liste de vecteurs d'erreur et une liste de syndromes. Par ailleurs, dans les algorithmes de décodage de syndrome et de remplissage de sac à dos, nous avions largement utilisé des collisions entre des listes de morceaux de vecteurs d'erreur. Nous allons maintenant combiner ces deux possibilités.

Considérons un algorithme construit selon une structure d'arbre (1.2.4). Nous remplaçons maintenant l'une des listes de départ (l'une des listes construites par énumération de tous les vecteurs possibles) par une liste de syndromes. Naturellement, il faut adapter la sous-routine de fusion de listes.

Dans le décodage de syndrome (et déjà dans le remplissage de sac à dos), la fusion de listes fonctionnait de la façon suivante (figure 5.1 gauche). À partir de deux listes de vecteurs, L_1 et L_2, on construit la liste des sommes d'un vecteur de L_1 et d'un vecteur de L_2, qui vérifient une certaine contrainte de modulo. Cette contrainte de modulo est la même pour toutes les sommes considérées, et est choisie en fonction du syndrome à décoder.

Dans le cas d'une fusion hybride, on peut avoir une liste constituée de couples [vecteur, syndrome], et non plus de vecteurs. Dans ce cas, c'est ce syndrome qui sert de condition de modulo (figure 5.1 droite). Ainsi, dans la liste finale, la liste du niveau 0, on obtient des couples [vecteur d'erreur, syndrome] compatibles.

Remarque 5.2. Dans un algorithme de résolution de DOOM, à chaque niveau, au plus une liste peut contenir des syndromes. En effet, on sait fusionner une liste de vecteurs et une liste de couples [vecteur, syndrome]. En revanche, on ne sait pas fusionner deux listes qui contiennent chacune des syndromes. Cela est dû à l'absence de structure de $\mathbf{s}$.

5.2.3 Algorithme pédagogique

Nous allons maintenant examiner un algorithme simple qui illustre les fusions hybrides. Cet algorithme est à visée purement pédagogique, ses performances étant très mauvaises. En particulier, il est plus lent que l'algorithme

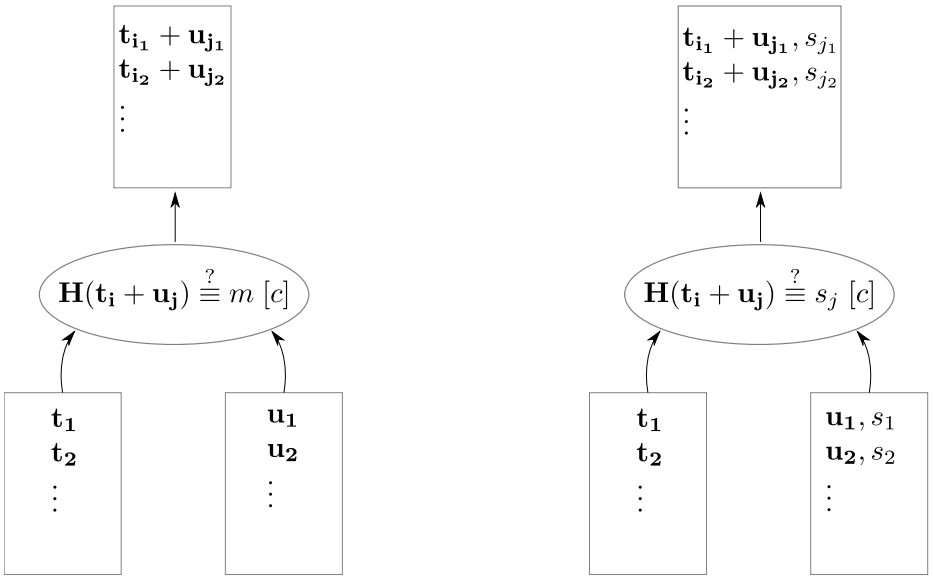

FIGURE 5.1 – À gauche : fusion usuelle entre deux listes, avec une condition de modulo. À droite : fusion de deux listes dans le cas où l'une d'elles contient des syndromes.

de [Sen11], et même que le décodage de syndrome. On se place dans le cas où le poids demandé est ω_{GV}, c'est-à-dire que le poids ω vérifie $nh(\omega) = n - k$.

Dans cet algorithme, on utilise (comme dans le cas de décodage de syndrome), une élimination gaussienne partielle. On se ramène alors à résoudre un problème de DOOM réduit, pour lequel on veut beaucoup de solutions.

Pour reprendre les notations du chapitre 3, la matrice de parité du problème réduit est de taille $\ell \times (k + \ell)$, et on demande un poids ω''. Ici, on choisit $\ell := \frac{2k(n-k)}{2k+n}$ et $\omega'' := \frac{(k+\ell)\omega}{n}$. Étant donné une solution $(\overline{\mathbf{e}''}, \overline{\mathbf{s}''}(i))$ au problème réduit, la probabilité que ce soit une solutions au problème de départ est alors $\tilde{\mathcal{O}}\left(2^{p_{succ}}\right)$, où $p_{succ} = -\ell/2$.

Pour trouver des solutions au problème réduit, on va cherche le vecteur d'erreur comme la concaténation de trois parties : la partie gauche, la partie centrale et la partie droite, chacune de longueur $(k + \ell)/3$. Pour cela, on utilise un arbre à trois niveaux (du niveau 0 au niveau 2) :

— au niveau 2 : on construit quatre listes : $L_{2,0}, L_{2,1}, L_{2,2}$ et $L_{2,3}$. Ces listes sont définies par :

$$\begin{array}{lcll} L_{2,0} & := & \{[\mathbf{e_0}, \mathbf{0}, \mathbf{0}] \mid |\mathbf{e_0}| = \omega''/3\} & \text{(les parties gauches)} \\ L_{2,1} & := & \{[\mathbf{0}, \mathbf{e_1}, \mathbf{0}] \mid |\mathbf{e_1}| = \omega''/3\} & \text{(les parties centrales)} \\ L_{2,2} & := & \{[\mathbf{0}, \mathbf{0}, \mathbf{e_2}] \mid |\mathbf{e_2}| = \omega''/3\} & \text{(les parties droites)} \\ L_{2,3} & := & \{\overline{\mathbf{s}''}(i) \mid i \in [\![1, 2^{\ell/2}]\!]\} & \text{(les syndromes)} \end{array}$$

Ces quatre listes sont de taille $2^{\ell/2}$. C'est évident pour $L_{2,3}$, et pour

les autres c'est une conséquence des choix de ℓ et ω''.

— au niveau 1, on a deux liste. $L_{1,0}$ est construite en fusionnant normalement $L_{2,0}$ et $L_{2,1}$, et $L_{1,1}$ et obtenue par une fusion hybride de $L_{2,2}$ et $L_{2,3}$. La contrainte de modulo est $c_1 = 2^{\ell/2}$. On a ainsi :

$$L_{1,0} = \left\{ [\mathbf{e_0}, \mathbf{e_1}, \mathbf{0}] \middle| \begin{array}{c} |\mathbf{e_0}| = \omega''/3, |\mathbf{e_1}| = \omega''/3 \\ \overline{\mathbf{H}''}[\mathbf{e_0}, \mathbf{e_1}, \mathbf{0}] \equiv 0 \ [2^{\ell/2}] \end{array} \right\}$$

$$L_{1,1} = \left\{ [\mathbf{0}, \mathbf{0}, \mathbf{e_2}], \overline{\mathbf{s}''}(i) \middle| \begin{array}{c} |\mathbf{e_2}| = \omega''/3, i \in [\![1, 2^{\ell/2}]\!] \\ \overline{\mathbf{H}''}[\mathbf{0}, \mathbf{0}, \mathbf{e_2}] \equiv \overline{\mathbf{s}''}(i) \ [2^{\ell/2}] \end{array} \right\}$$

L'espérance des tailles de ces deux listes est $2^{\ell/2}$.

— au niveau 0, on réalise une fusion hybride des deux listes du niveau 1, avec une condition de modulo totale $c_2 = 2^{\ell}$. On obtient :

$$L_0 = \left\{ [\mathbf{e_0}, \mathbf{e_1}, \mathbf{e_2}], \overline{\mathbf{s}''}(i) \middle| \begin{array}{c} |\mathbf{e_0}| = \omega''/3, |\mathbf{e_1}| = \omega''/3 \\ |\mathbf{e_2}| = \omega''/3, i \in [\![1, 2^{\ell/2}]\!] \\ \overline{\mathbf{H}''}[\mathbf{e_0}, \mathbf{e_1}, \mathbf{e_2}] \equiv \overline{\mathbf{s}''}(i) \ [2^{\ell}] \end{array} \right\}$$

C'est une liste de $2^{\ell/2}$ solutions au problème réduit.

On a $2^{\ell/2}$ solutions au problème réduit, chacune ayant une probabilité $2^{-\ell/2}$ d'être une solution au problème de départ : on a en moyenne une solution (aux facteur polynomiaux près).

5.2.4 Représentations asymétriques

L'algorithme précédent est très mauvais. La raison à cela est qu'il utilise un arbre trop profond compte tenu du nombre de solutions disponibles. Or nous savons bien comment ajouter artificiellement beaucoup de solutions : il suffit d'utiliser des représentations. Nous allons donc ajouter des représentations entre les niveaux 0 et 1.

Dans la liste L_0, on veut des couples (vecteur d'erreur, syndrome). On va chercher ce couples comme la combinaison d'un vecteur de distribution $D^{k+\ell}[\alpha_{1,0}]$ et d'un couple (vecteur de $D^{k+\ell}[\alpha_{1,1}]$, syndrome).

Dans le cas du décodage de syndrome, on choisissait toujours la même distribution pour la liste de gauche et la liste de droite. Dans le cas de DOOM, le fait d'avoir une liste de syndromes déséquilibre complètement l'arbre. Pour avoir une meilleure complexité, on essaie de rétablir l'équilibre en prenant des distributions différentes.

5.2.5 Paramètres dans le pire cas

Le pire cas est atteint pour $R = k/n \simeq 0.408$. Le poids de Gilbert-Varshamov associé est $\omega \simeq 0.143$. Dans ce cas, le meilleur exposant de complexité obtenu est 0.0822, à l'aide des paramètres suivants :

— on fait une élimination gaussienne partielle, avec les paramètres $\ell = 0.1644n$, et $\omega'' = 0.0337$. Une solution au problème réduit est alors une solution au problème de départ avec probabilité $p_{succ} \simeq -0.0770n$.
Il reste à trouver comment calculer $2^{0.0770n}$ solutions au problème réduit en temps $\tilde{\mathcal{O}}\left(2^{0.0822n}\right)$. Pour cela, on utilise une structure d'arbre à 3 niveaux.

— au niveau 2, on construit quatre listes. $L_{2,0}$ et $L_{2,1}$ sont des énumérations de moitiés gauches et droites de vecteurs contenant une certaine densité $\alpha_{1,0} \simeq 0.0502$ de 1. Ces listes sont de taille $\ell_{2,0} = \ell_{2,1} = 0.0822$.

$$\begin{aligned} L_{2,0} &:= D^{(k+\ell)/2}[\alpha_{1,0}] \times \mathbf{0}^{(k+\ell)/2} \\ L_{2,1} &:= \mathbf{0}^{(k+\ell)/2} \times D^{(k+\ell)/2}[\alpha_{1,0}] \end{aligned}$$

$L_{2,2}$ est l'énumération de tous les vecteurs contenant un certaine densité $\alpha_{1,1} \simeq 0.0204$ de 1, et sa taille est $\ell_{2,2} = 0.0822$:

$$L_{2,2} := D^{k+\ell}[\alpha_{1,1}]$$

Enfin, $L_{2,3}$ est une liste de syndromes de taille $\ell_{2,3} = 0.0822$:

$$L_{2,3} := \{\overline{\mathbf{s}''(i)} \mid i \in [\![1, 2^{0.0822n}]\!]\}$$

— au niveau 1, on construit deux listes par fusion des listes du niveau 2 avec une contrainte de modulo $c_1 = 0.0822n$.
$L_{1,0}$ est obtenue en faisant une fusion des listes $L_{2,0}$ et $L_{2,1}$. $L_{1,0}$ est une liste de taille $\ell_{1,0} = \ell_{2,0} + \ell_{2,1} - c_1 = 0.0822n$, et contient des vecteurs $\mathbf{e_1} \in D^{k+\ell}[\alpha_{1,0}]$ vérifiant $\mathbf{e_1} \equiv 0\ [c_1]$.
$L_{1,1}$ est obtenue par une fusion hybride des listes $L_{2,2}$ et $L_{2,3}$. Cette liste est de taille $\ell_{1,1} = \ell_{2,2} + \ell_{2,3} - c_1 = 0.0822n$. Elle est constituée de couples $(\mathbf{e_2}, \mathbf{s}(i))$ vérifiant $\mathbf{e_2} \equiv \mathbf{s}(i)\ [c_1]$.

— au niveau 0, on construit la liste L_0 en fusionnant les deux listes du niveau 1, avec une contrainte de modulo totale $c_0 = 0.0822n$. À cause des éléments mal formés, la liste L_0 est seulement de taille $\ell_0 = 0.0770$.

5.2.6 Complexité

La figure 5.3 illustre la complexité l'algorithme obtenus de cette façon. Pour chaque valeur du ratio $R = k/n$, on se place au poids de Gilbert-Varshamov associé.

Le cas le plus difficile correspond à $R = 0.408$, avec une complexité de $\tilde{\mathcal{O}}\left(2^{0.0822n}\right)$. Le tableau 5.2 compare la complexité de cet algorithme à d'autres algorithmes (y compris à des algorithmes de décodage simple).

[BJMM12]	$\tilde{\mathcal{O}}\left(2^{0.1019n}\right)$
[Car20]	$\tilde{\mathcal{O}}\left(2^{0.0882n}\right)$
[Sen11]	$\tilde{\mathcal{O}}\left(2^{0.0872n}\right)$
5.2.5	$\tilde{\mathcal{O}}\left(2^{0.0822n}\right)$

FIGURE 5.2 – Complexité de différents algorithmes pour résoudre le problème DOOM

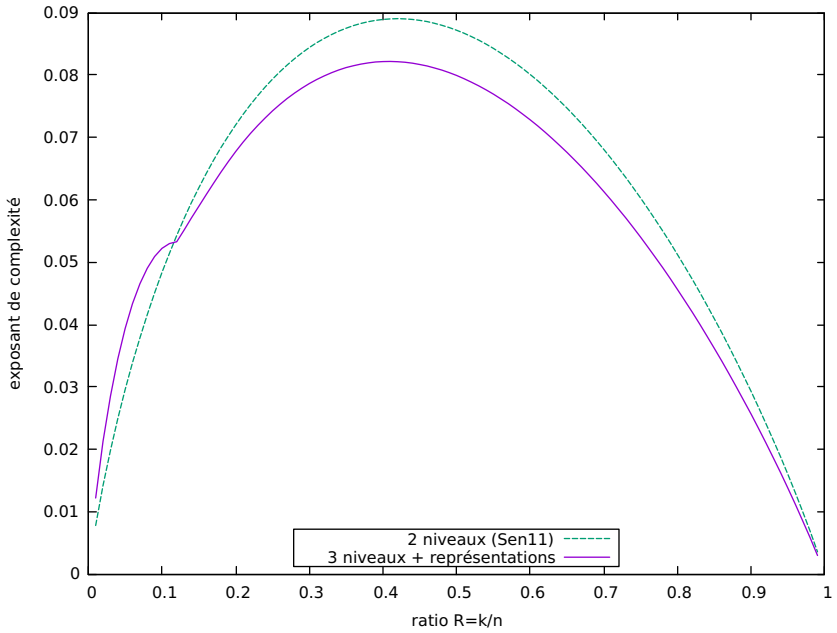

FIGURE 5.3 – Exposant asymptotique de complexité du problème DOOM à ω_{GV}, en utilisant une élimination de Gauss partielle, un arbre à 3 niveaux et des représentations.

5.3 Algorithme quantique

5.3.1 Énumération de listes à la volée

Nous allons introduire une asymétrie supplémentaire. À chaque niveau, une liste (par exemple la liste la plus à droite) sera énumérée au fur et à mesure, au lieu d'être gardée en mémoire. Nous nous concentrerons sur le cas d'un arbre à trois niveau (illustré dans 5.4), ce qui correspond au cas qui a donné les meilleurs résultats. Concrètement, la marche à suivre est la suivante :

Pré-calcul : on construit et enregistre en mémoire les listes comme on le faisait auparavant, à l'exception de la liste la plus à droite de chaque niveau.

Énumération : on considère un vecteur qui aurait dû servir à construire la liste la plus à droite du dernier niveau (la liste $L_{2,3}$ dans la figure 5.4). On calcule les collisions entre ce vecteur et la liste $L_{2,2}$. Chacune de ces collisions aurait été un élément de $L_{1,1}$, si on avait construit cette liste. Pour chacun de ces vecteurs, on recherche les collisions avec $L_{1,0}$. On obtient ainsi quelques vecteurs de L_0, c'est-à-dire quelques solutions au problème réduit. Si on trouve une solution au problème de départ parmi ces vecteur, on a fini. Sinon, on oublie les collisions obtenues et on recommence avec un autre vecteur de $L_{2,3}$.

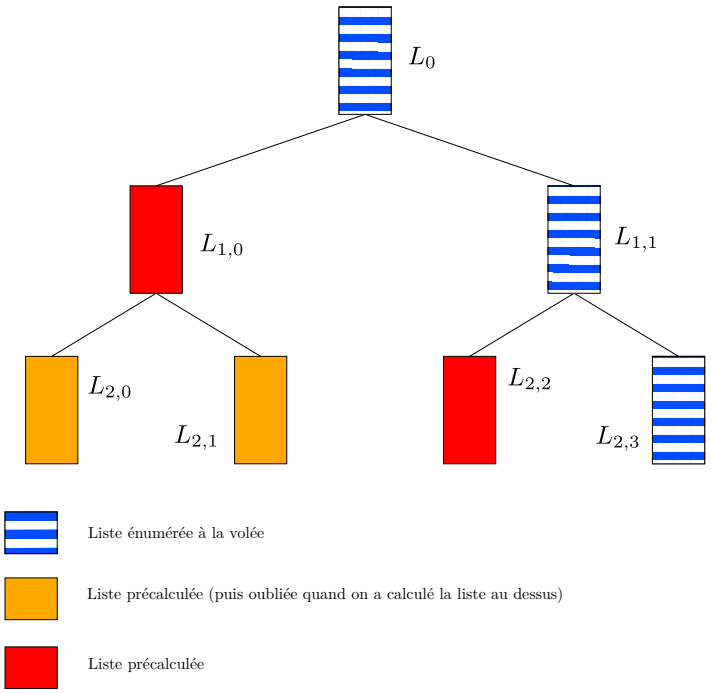

FIGURE 5.4 – Structure d'un arbre à trois niveaux dans lequel on énumère des listes à la volée au lieu de les mémoriser

5.3.2 Applications des énumérations

La complexité en mémoire d'un arbre tel que celui de 5.4 est donnée par le maximum des tailles des listes mémorisées. Si une liste est énumérée à la volée au lieu d'être conservée en mémoire, sa taille n'intervient pas dans la complexité en mémoire. Le cas intéressant est bien entendu le cas où les listes énumérées (les listes les plus à droite de l'arbre) sont beaucoup plus grandes que les autres listes : cela permet de diminuer la complexité en mémoire de l'arbre.

En général, il est plus intéressant d'avoir un arbre le plus équilibré possible. Cependant, il y a plusieurs exceptions à cela :

— Les compromis temps/mémoire : on déséquilibre un arbre qui était auparavant équilibré. Certaines listes deviennent alors plus petites tandis que d'autres grandissent. Le coût en temps de l'arbre déséquilibré est donné par la taille de la liste la plus grande, et va donc augmenter. En revanche, le coût en mémoire dépend de la taille de la plus grande liste *stockée dans la mémoire*. Si on parvient à énumérer les listes les plus grandes, la complexité en mémoire peut diminuer. Les arbres déséquilibrés et les énumérations de listes permettent donc de diminuer la mémoire requise par un algorithme, au prix d'une augmentation de la complexité en temps.

— Les amplifications d'amplitude quantiques. Dans la sous-section précédente, nous avons considéré une sous-routine qui, à partir d'un élément de $L_{2,3}$, détermine s'il permet d'obtenir une solution au problème. À l'aide d'une amplification d'amplitude, le temps requis pour trouver un élément de $L_{2,3}$ qui conduit à une solution n'est que la racine carré de sa taille. Pour un algorithme quantique, les listes énumérées peuvent contenir le carré du nombre d'éléments des autres listes, sans que cela ne se ressente sur la complexité.

5.3.3 Structure d'un algorithme quantique

Pour résoudre le problème DOOM à l'aide d'un algorithme quantique, nous allons supposer que nous avons un accès quantique à la fonction $\mathbf{s}$ qui génère les syndromes. Autrement dit, nous disposons d'un oracle vérifiant $\mathcal{O}_{\mathbf{s}} : \sum_i |i\rangle|0\rangle \mapsto \sum_i |i\rangle|\mathbf{s}(i)\rangle$.

Nous utiliserons un algorithme très similaire au cas classique. De la même façon, on fait intervenir un problème réduit, que nous résolvons grâce à un arbre à trois niveaux.

Au niveau 2, les listes $L_{2,0}$, $L_{2,1}$ et $L_{2,2}$ contiennent toujours le même nombre d'éléments. En revanche, la liste de syndromes, $L_{2,2}$, contient le carré de ce nombre d'éléments. Nous avons choisi la liste $L_{2,2}$ pour être la liste la plus grande car cette liste n'est pas contrainte par l'entropie. En effet, les tailles des listes $L_{2,0}$, $L_{2,1}$ et $L_{2,2}$ sont limitées par le nombre de vecteurs différents qui existent pour un poids donné, tandis que le nombre de syndromes différents disponibles n'est pas contraignant (du moins, pas à ω_{GV}).

La contrainte de modulo entre les niveaux 2 et 1, ainsi que la contrainte additionnelle entre les niveaux 1 et 0 sont choisies du même ordre de grandeur que les tailles de $L_{2,0}$, $L_{2,1}$ et $L_{2,2}$. De cette façon, la taille de la liste $L_{1,0}$ est également du même ordre de grandeur. Les listes virtuelles $L_{1,1}$ et L_0, quant à elles, ont une taille similaire à $L_{2,2}$.

Par ailleurs, les paramètres ℓ et ω'' sont choisis de façon à ce que $L_0 = -p_{succ}$: il y a en moyenne une solution au problème de départ dans la liste

virtuelle L_0.

La complexité d'un tel algorithme est donné dans la figure 5.5.

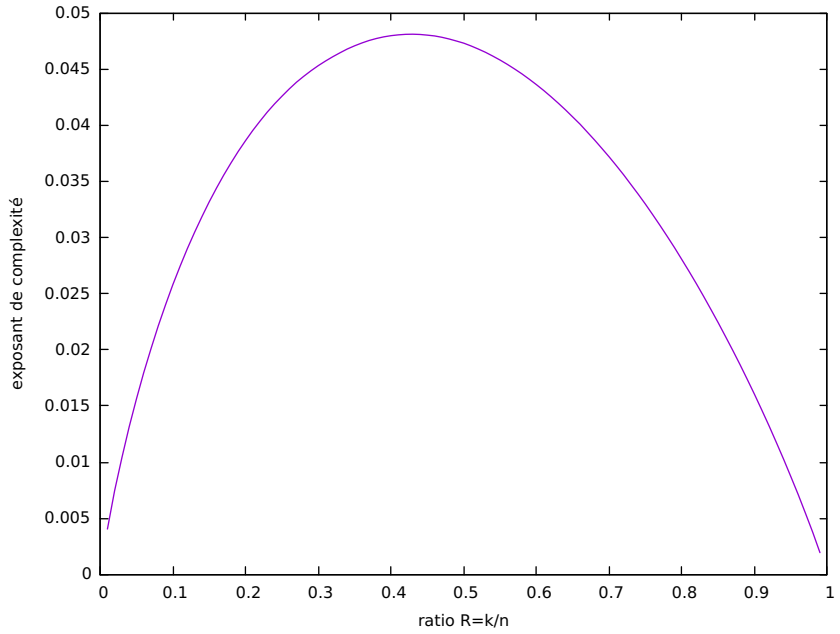

FIGURE 5.5 – Exposant asymptotique de complexité du problème DOOM à ω_{GV} pour un algorithme quantique.

5.3.4 Pire cas

Le pire cas est atteint pour $R = k/n \simeq 0.428$. La complexité est alors $\tilde{\mathcal{O}}\left(2^{0.0481n}\right)$. Cette complexité est obtenue en prenant comme paramètres $\ell = 0.0981$ et $\omega'' = 0.0204$ pour le problème réduit. La probabilité qu'une solution au problème réduit soit une solution au problème de départ est alors donnée par $p_{succ} = -0.0962$.

Pour le problème réduit, on utilise la structure de 5.4. La liste $L_{2,0}$ (resp. $L_{2,1}$) est constituée de tous les vecteurs dont la densité est $\alpha_{1,0} \simeq 0.0278$ sur la partie gauche (resp. droite), et 0 sur l'autre moitié. La liste $L_{2,2}$ est constituée de tous les vecteurs de densité $\alpha_{1,1} = 0.0117$. Ces trois listes sont de taille 0.0481.

Entre les niveaux 2 et 1, on utilise une condition de modulo $c_1 = 0.0481$. La liste $L_{1,0}$ a alors une taille similaire aux trois listes précédentes.

Les listes $L_{2,2}$ et $L_{1,0}$ ayant ainsi été pré-calculées, on réalise maintenant une amplification d'amplitude sur un ensemble de syndromes de taille 0.0962. Les listes virtuelles $L_{1,1}$ et L_0 sont de taille $0.0962 = -p_{succ}$.

La complexité globale est donc $\tilde{\mathcal{O}}\left(2^{0.0481n}\right)$.

Remarque 5.3. Nous avons supposé un accès quantique à $\mathbf{s}$. Cela est nécessaire pour pouvoir faire une amplification d'amplitude sur la liste contenant les syndromes, la liste $L_{2,3}$. Si on ne disposait pas d'un tel accès, on ne pourrait pas obtenir la même complexité. La meilleure complexité atteignable dans ce cas se situerait quelque part entre la complexité avec accès quantique à $\mathbf{s}$ et la complexité du décodage simple.

5.4 Et pour Wave ?

Le schéma de signature Wave peut être attaqué en utilisant le problème DOOM plutôt que le décodage de syndrome. Cependant, il y a déjà tellement de solutions que l'apport d'une liste de syndromes est très faible.

Cas classique

Le problème DOOM se combine très mal avec les représentations. En effet, le fait d'utiliser une liste de syndromes force un arbre déséquilibré. Cela fait exploser le nombre de paramètres nécessaires pour décrire toutes les densités utilisées par les représentations.

Si on oublie les représentations et qu'on se contente de l'algorithme de Wagner, il est beaucoup plus facile de remplacer l'une des listes du dernier étage par une liste de syndromes. Cela permet de passer d'une complexité de $\tilde{\mathcal{O}}\left(2^{0.01565n}\right)$ à $\tilde{\mathcal{O}}\left(2^{0.01561n}\right)$.

Dans le cas des représentations, il n'y a aucune raison d'espérer une amélioration plus avantageuse.

Cas quantique

Dans le cas des algorithmes quantiques, nous avions déjà abandonné les représentations pour des problèmes d'Update. Nous pouvons donc directement considérer l'avantage du problème DOOM : ce problème peut être résolu avec une complexité $\tilde{\mathcal{O}}\left(2^{0.00969n}\right)$ là où on proposait $\tilde{\mathcal{O}}\left(2^{0.00970n}\right)$ pour le décodage.

Dans les deux cas, on constate que le problème DOOM est strictement plus facile à résoudre que le problème de décodage, mais que la différence est insignifiante. Cela est dû au fait que, dans les paramètres utilisés par Wave, il y a déjà énormément de solutions. Le fait de pouvoir choisir le syndrome n'aide pas tellement.

Une autre façon de le voir est de constater que les algorithmes présentés pour la résolution de Wave utilisent des arbres avec 7 niveaux. Au dernier niveau, le niveau 6, on peut remplacer l'une des listes par une liste de syndromes. Ainsi, l'une des 2^6 listes du niveau 6 ne consomme pas de place dans

l'espace de recherche. Le fait de rendre gratuite une liste sur 64 n'apporte pas grand chose.

Conclusion

Dans cette thèse, nous avons étudié des techniques permettant d'accélérer le décodage de syndrome, aussi bien dans le cas d'algorithmes classiques que pour des algorithmes quantiques. Au passage, nous avons présenté quelques résultats concernant le problème, proche, du remplissage de sac à dos.

Dans une première partie, nous nous sommes concentrés sur le problème de remplissage de sac à dos. Nous y avons présenté un algorithme classique dont la complexité asymptotique améliore l'état de l'art. Si le gain apporté n'est pas significatif dans ce cas précis, l'apport théorique n'en est pas moins intéressant. Une meilleure compréhension des représentations, notamment, possède des applications génériques dans bon nombre de problèmes similaires (cas quantique, décodage, DOOM, ...)

Nous avons également proposé un meilleur algorithme quantique pour résoudre le problème de remplissage de sac à dos. En plus de la répercussion du travail réalisé dans le cas classique, nous avons montré comment utiliser plus subtilement les marches quantiques.

Dans une seconde partie, nous avons étudié le problème du décodage de syndrome. La principale contribution de ces travaux est une première étude des algorithmes de décodage dans le corps $\mathbb{F}_3$, lorsque le poids de l'erreur à trouver est grand. Nous avons proposé des algorithmes, classiques et quantiques, pour résoudre le problème dans le pire cas. Nous avons également construit des algorithmes fonctionnant dans le cas où il y a de nombreuses solutions, et tout particulièrement dans le cas des paramètres de Wave.

Enfin, nous avons étudié la variante DOOM du problème de décodage. Dans le cas binaire, nous avons créé un meilleur algorithme classique en incluant des représentations. Dans le cas quantique, nous avons présenté un premier algorithme de résolution du problème DOOM.

Prolongements : Malgré les résultats obtenus et publiés, il reste encore de nombreuses perspectives, dont :

— L'heuristique quantique 2 est très pratique, mais peu satisfaisante. Est-il possible de mieux la justifier ?

— Pour résoudre le problème de remplissage de sac à dos, il serait possible d'obtenir un meilleur algorithme théorique en incluant les résultats de [Oze16]. Son amélioration de la fusion de listes en présence d'éléments mal formés permet de diminuer de coût de ceux-ci. Il serait alors possible d'utiliser des représentations plus nombreuses. Le cas quantique, notamment, semble très délicat à étudier.

— Dans le cas du décodage, il serait intéressant de voir ce qu'on pourrait faire en incluant la technique de recherche de plus proches voisins de [Car20]. Comme dans le cas précédent, cela permettrait de diminuer le coût des éléments mal formés lors de l'utilisation de représentations.

— Le problème de décodage semble plus difficile dans $\mathbb{F}_3$ que dans $\mathbb{F}_2$. Est-ce réellement le cas, ou est-ce dû au fait que le cas binaire a été bien plus étudié ? Qu'en est-il des corps $\mathbb{F}_q$, pour $q \geqslant 4$?

Bibliographie

[ACP+17] Martin Albrecht, Carlos Cid, Kenneth G. Paterson, Cen Jung Tjhai, and Martin Tomlinson. NTS-KEM. first round submission to the NIST post-quantum cryptography call, December 2017. 75

[AMAB+17a] Carlos Aguilar Melchor, Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Shay Gueron, Tim Güneysu, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, Jean-Pierre Tillich, and Gilles Zémor. BIKE. first round submission to the NIST post-quantum cryptography call, November 2017. 75

[AMAB+17b] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, and Gilles Zémor. HQC, December 2017. NIST Round 1 submission for Post-Quantum Cryptography. 75

[BBC+19] Marco Baldi, Alessandro Barenghi, Franco Chiaraluce, Gerardo Pelosi, and Paolo Santini. LEDAcrypt. second round submission to the NIST post-quantum cryptography call, January 2019. 75

[BBSS20] Xavier Bonnetain, Rémi Bricout, André Schrottenloher, and Yixin Shen. Improved classical and quantum algorithms for subset-sum. In *ASIACRYPT*, Lecture Notes in Computer Science. Springer, 2020. 12, 55, 67, 71

[BCDL19] Rémi Bricout, André Chailloux, Thomas Debris-Alazard, and Matthieu Lequesne. Ternary syndrome decoding with large

weight. In *SAC*, volume 11959 of *Lecture Notes in Computer Science*, pages 437–466. Springer, 2019. 13, 84, 120

[BCJ11] Anja Becker, Jean-Sébastien Coron, and Antoine Joux. Improved generic algorithms for hard knapsacks. *IACR Cryptology ePrint Archive*, 2011 :474, 2011. 11, 12, 18, 48, 49, 50, 56, 57, 59, 60, 67

[BCL+17] Daniel J. Bernstein, Tung Chou, Tanja Lange, Ingo von Maurich, Ruben Niederhagen, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, and Wang Wen. Classic McEliece : conservative code-based cryptography. `https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/submissions/Classic_McEliece.zip`, November 2017. First round submission to the NIST post-quantum cryptography call. 75

[BHMT02] Gilles Brassard, Peter Hoyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305 :53–74, 2002. 24

[BJLM13] Daniel J. Bernstein, Stacey Jeffery, Tanja Lange, and Alexander Meurer. Quantum algorithms for the subset-sum problem. In *PQCrypto*, volume 7932 of *Lecture Notes in Computer Science*, pages 16–33. Springer, 2013. 11, 12, 50, 53, 71

[BJMM12] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in 2 n/20 : How 1 + 1 = 0 improves information set decoding. In *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 520–536. Springer, 2012. 80, 94, 96, 97, 111, 132

[BM17] Leif Both and Alexander May. Optimizing BJMM with Nearest Neighbors : Full Decoding in $2^{2/21n}$ and McEliece Security. In *WCC Workshop on Coding and Cryptography*, September 2017. on line proceedings, see `http://wcc2017.suai.ru/Proceedings_WCC2017.zip`. 96

[BM18] Leif Both and Alexander May. Decoding linear codes with high error rate and its impact for LPN security. In *PQCrypto*, volume 10786 of *Lecture Notes in Computer Science*, pages 25–46. Springer, 2018. 96, 97

[BMvT78] Elwyn R. Berlekamp, Robert J. McEliece, and Henk C. A. van Tilborg. On the inherent intractability of certain coding

problems (corresp.). *IEEE Trans. Inf. Theory*, 24(3) :384–386, 1978. 75

[Car20] Kévin Carrier. *Presque-Collisions et applications au Décodage Générique et à la Reconnaissance de Codes Correcteurs d'Érreurs*. Theses, Sorbonne Université, 2020. 96, 97, 132, 140

[CFS01] Nicolas T. Courtois, Matthieu Finiasz, and Nicolas Sendrier. How to achieve a mceliece-based digital signature scheme. In *ASIACRYPT*, volume 2248 of *Lecture Notes in Computer Science*, pages 157–174. Springer, 2001. 13

[CJL+92] Matthijs J. Coster, Antoine Joux, Brian A. LaMacchia, Andrew M. Odlyzko, Claus-Peter Schnorr, and Jacques Stern. Improved low-density subset sum algorithms. *Comput. Complex.*, 2 :111–128, 1992. 31

[Die82] D. Dieks. Communication by epr devices. *Physics Letters A*, 92(6) :271 – 272, 1982. 19

[DST19] Thomas Debris-Alazard, Nicolas Sendrier, and Jean-Pierre Tillich. Wave : A new family of trapdoor one-way preimage sampleable functions based on codes. In *ASIACRYPT (1)*, volume 11921 of *Lecture Notes in Computer Science*, pages 21–51. Springer, 2019. 13, 99, 116, 120

[Dum91] Ilya Dumer. On minimum distance decoding of linear codes. In *Proc. 5th Joint Soviet-Swedish Int. Workshop Inform. Theory*, pages 50–52, Moscow, 1991. 80, 94

[GJ79] M. R. Garey and David S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. 30

[GKH17] Cheikh Thiécoumba Gueye, Jean Belo Klamti, and Shoichi Hirose. Generalization of BJMM-ISD using may-ozerov nearest neighbor algorithm over an arbitrary finite field $\mathbb{F}_q$. In *C2SI*, volume 10194 of *Lecture Notes in Computer Science*, pages 96–109. Springer, 2017. 96

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *STOC*, pages 212–219. ACM, 1996. 11, 22

[Hir16] Shoichi Hirose. May-ozerov algorithm for nearest-neighbor problem over $\mathbb{F}_q$ and its application to information set deco-

ding. In *SECITC*, volume 10006 of *Lecture Notes in Computer Science*, pages 115–126, 2016. 96

[HJ10] Nick Howgrave-Graham and Antoine Joux. New generic algorithms for hard knapsacks. In *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 235–256. Springer, 2010. 11, 12, 18, 41, 45, 47, 48, 50, 56, 94

[HM18] Alexander Helm and Alexander May. Subset sum quantumly in 1.17^{n}. In *TQC*, volume 111 of *LIPIcs*, pages 5 :1–5 :15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. 11, 12, 48, 50, 53, 67, 71

[Hol73] Alexander S. Holevo. Bounds for the quantity of information transmitted by a quantum communication channel. *Probl. Peredachi Inf.*, 9 :177–183, 1973. 19

[HS74] Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *J. ACM*, 21(2) :277–292, 1974. 10, 32, 34, 94

[Kit97] A Yu Kitaev. Quantum computations : algorithms and error correction. *Russian Mathematical Surveys*, 52(6) :1191–1249, dec 1997. 21

[KT17] Ghazal Kachigar and Jean-Pierre Tillich. Quantum information set decoding algorithms. In *PQCrypto*, volume 10346 of *Lecture Notes in Computer Science*, pages 69–89. Springer, 2017. 25, 53

[LB88] Pil J. Lee and Ernest F. Brickell. An observation on the security of McEliece's public-key cryptosystem. In *'88*, volume 330, pages 275–280. Springer, 1988. 80

[LO85] J. C. Lagarias and Andrew M. Odlyzko. Solving low-density subset sum problems. *J. ACM*, 32(1) :229–246, 1985. 31

[LPS10] Vadim Lyubashevsky, Adriana Palacio, and Gil Segev. Public-key cryptographic primitives provably as secure as subset sum. In *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 382–400. Springer, 2010. 29

[McE78] Robert J. McEliece. *A Public-Key System Based on Algebraic Coding Theory*, pages 114–116. Jet Propulsion Lab, 1978. DSN Progress Report 44. 11, 75

[MH78] Ralph C. Merkle and Martin E. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Trans. Inf. Theory*, 24(5) :525–530, 1978. 29

[MMT11] Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in $\tilde{\mathcal{O}}(2^{0.054n})$. In *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 107–124. Springer, 2011. 80, 94, 95, 97

[MNRS11] Frédéric Magniez, Ashwin Nayak, Jérémie Roland, and Miklos Santha. Search via quantum walk. *SIAM J. Comput.*, 40(1) :142–164, 2011. 24, 25, 50

[MO15] Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In *EUROCRYPT (1)*, volume 9056 of *Lecture Notes in Computer Science*, pages 203–228. Springer, 2015. 96, 97

[NC00] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000. 20, 22

[Nie86] H. Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Problems Control Inform. Theory/Problemy Upravlen. Teor. Inform.*, 15(2) :159–166, 1986. 75

[Oze16] Ilya Ozerov. *Combinatorial algorithms for subset sum problems*. doctoralthesis, Ruhr-Universität Bochum, Universitätsbibliothek, 2016. 43, 140

[Pra62] Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Trans. Information Theory*, 8(5) :5–9, 1962. 80, 97

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2) :120–126, 1978. 10

[Sen11] Nicolas Sendrier. Decoding one out of many. In *PQCrypto*, volume 7071 of *Lecture Notes in Computer Science*, pages 51–67. Springer, 2011. 14, 129, 132

[Sha49] Claude E. Shannon. Communication theory of secrecy systems. *Bell Syst. Tech. J.*, 28(4) :656–715, 1949. 9

[Sho94] Peter W. Shor. Algorithms for quantum computation : Discrete logarithms and factoring. In *FOCS*, pages 124–134. IEEE Computer Society, 1994. 10

[SS81] Richard. Schroeppel and Adi. Shamir. A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain np-complete problems. *SIAM Journal on Computing*, 10(3) :456–464, 1981. 10, 12, 34, 56

[Ste88] Jacques Stern. A method for finding codewords of small weight. In G. D. Cohen and J. Wolfmann, editors, *Coding Theory and Applications*, volume 388, pages 106–113. Springer, 1988. 80

[Ste93] Jacques Stern. A new identification scheme based on syndrome decoding. In *CRYPTO*, volume 773 of *Lecture Notes in Computer Science*, pages 13–21. Springer, 1993. 75

[Sze04] Mario Szegedy. Quantum speed-up of markov chain based algorithms. In *FOCS*, pages 32–41. IEEE Computer Society, 2004. 24

[Tof80] Tommaso Toffoli. Reversible computing. In *ICALP*, volume 85 of *Lecture Notes in Computer Science*, pages 632–644. Springer, 1980. 21

[Wag02] David A. Wagner. A generalized birthday problem. In *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–303. Springer, 2002. 36

[WZ82] W. K. Wootters and W. H. Zurek. A single quantum cannot be cloned. *Nature*, 299(5886) :802–803, October 1982. 19