

Algorithmique de la réduction de réseaux et application à la recherche de pires cas pour l'arrondi defonctions mathématiques

Damien Stehlé

▶ **To cite this version:**

Damien Stehlé. Algorithmique de la réduction de réseaux et application à la recherche de pires cas pour l'arrondi defonctions mathématiques. Génie logiciel [cs.SE]. Université Henri Poincaré - Nancy I, 2005. Français. NNT : . tel-00011150

HAL Id: tel-00011150
https://theses.hal.science/tel-00011150

Submitted on 5 Dec 2005

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.

Département de formation doctorale en informatique **École doctorale IAEM Lorraine**
UFR STMIA

Algorithmique de la réduction de réseaux et application à la recherche de pires cas pour l'arrondi de fonctions mathématiques

THÈSE

présentée et soutenue publiquement le 2 décembre 2005

pour l'obtention du

Doctorat de l'Université Henri Poincaré – Nancy 1

(spécialité informatique)

par

Damien STEHLÉ

Composition du jury

Rapporteurs :	François MORAIN, Professeur, École Polytechnique, Palaiseau Gilles VILLARD, Chargé de recherche CNRS, Lyon
Examinateurs :	Peter MARKSTEIN, Principal research scientist, Hewlett-Packard, États-Unis Gérald TENENBAUM, Professeur, Université Henri Poincaré, Nancy Brigitte VALLÉE, Directrice de recherche CNRS, Caen Paul ZIMMERMANN, Directeur de recherche INRIA, Nancy (Directeur de thèse)

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thloria.

Remerciements

J'ai eu la chance d'avoir bénéficié du soutien et de l'aide de nombreuses personnes durant la préparation de cette thèse. J'aimerais leur exprimer ma profonde reconnaissance à travers ces quelques lignes.

En tout premier lieu, je souhaite souligner combien a été agréable et enrichissant l'encadrement de cette thèse par Paul Zimmermann. Il a su créer et entretenir un cadre de travail formidable. Sa patience, sa disponibilité et sa qualité d'écoute se sont avérées particulièrement précieux.

Je tiens aussi à remercier très profondément Phong Nguyễn, sans qui une très large partie de ce travail n'aurait été possible. Il m'a fait découvrir le domaine passionnant de la géométrie des nombres, dans lequel je prends énormément plaisir à travailler. Sa vivacité d'esprit et son sens critique redoutables ont développé mon enthousiasme. J'espère continuer cette collaboration scientifique, qui m'a déjà tant apporté.

Je remercie François Morain et Gilles Villard pour avoir relu ce manuscrit en profondeur. Leurs encouragements m'ont beaucoup touché, et leurs remarques, nombreuses et d'une grande richesse, m'ont aidé pour améliorer la rédaction. Je suis très heureux qu'ils aient accepté d'être mes rapporteurs.

Je suis honoré que Peter Markstein, Gérald Tenenbaum et Brigitte Vallée soient membres de mon jury. Je suis très sensible qu'ils aient fait le déplacement pour assister à la soutenance. J'en profite pour remercier chaleureusement Brigitte, qui a guidé mes premiers pas, à l'Université de Caen, dans la recherche en algorithmique. Ses travaux, dont plusieurs chapitres de cette thèse sont en partie inspirés, ont éveillé et entretenu mon intérêt pour ce domaine et stimulé de nombreuses interrogations. Elle m'a aussi mis en relation avec Paul, et encouragé à de multiples reprises. Je lui dois énormément et lui témoigne ma gratitude. Je n'oublie pas non plus Jacques Stern, Dan Boneh et Nigel Smart, qui m'ont accueilli dans leurs laboratoires et amené sur la voie de ma thématique de recherche.

Merci à tous les membres du projet Spaces, pour leur écoute, leur aide technique et scientifique, et leur soutien sans faille. Merci ainsi à Laurent, Pierrick, Guillaume, Vincent, Emmanuel. Je remercie tout particulièrement Patrick, avec lequel nous avons partagé le même bureau pendant deux ans. Il a su répondre avec patience à mes innombrables questions de programmation et m'a aidé à réparer mes étourderies diverses et variées.

Je suis également reconnaissant au LORIA et à sa directrice, pour m'avoir offert des conditions de travail remarquables. Je pense tout particulièrement à mes encadrants et tuteurs d'enseignement, auprès desquels j'ai beaucoup appris. Je remercie spécialement l'assistante de notre équipe Spaces, Céline, pour son énergie communicative et son efficacité à toute épreuve.

Je voudrais enfin remercier mes amis proches et ma famille. Leur soutien, à distance pour certains, a toujours été là, pendant les hauts et pendant les bas. J'espère sincèrement qu'ils continueront de m'accompagner.

Introduction.

Les réseaux euclidiens sont les arrangements réguliers de points dans l'espace, ou, plus précisément, les sous-groupes discrets de $\mathbb{R}^n$ pour un certain entier n. La théorie des réseaux euclidiens, aussi appelée géométrie des nombres, est née dans les années 1890, quand Minkowski a publié son ouvrage *Geometrie der Zahlen* [131]. Dans la suite des travaux de Hermite [77], il étudiait les minima de formes quadratiques définies positives. Il a montré que ce type de questions se traduisait géométriquement de manière extrêmement naturelle. Avec cette vision plus géométrique, de nombreux résultats sont devenus plus intuitifs et de ce fait beaucoup plus simples à établir. Ces théories des formes quadratiques définies positives et des réseaux euclidiens ont de très nombreuses applications en mathématiques, de l'étude des courbes algébriques aux corps de nombres, en passant par des résultats plus anecdotiques comme le théorème des quatre carrés de Lagrange (tout entier positif peut s'écrire comme somme de quatre carrés d'entiers).

Du point de vue algorithmique, la date charnière se situe au début des années 1980, lorsque Arjen Lenstra, Hendrik Lenstra et László Lovász mettent au point un algorithme de réduction de réseaux de complexité polynomiale [112]. Cet algorithme, que l'on appelle LLL du nom de ses auteurs, ou encore L^3, a constitué une véritable révolution. Tout d'abord, sa complexité est sans comparaison avec les algorithmes décrits jusqu'alors pour étudier les réseaux euclidiens, et, surtout, il a ouvert la voie à un nombre impressionnant d'applications. Les trois applications historiques sont la factorisation des polynômes à coefficients entiers ou rationnels [79, 89, 112, 156], les approximations rationnelles simultanées [97] et la programmation entière en dimension fixée [114]. Cet algorithme a aussi immédiatement reçu un succès immédiat dans le domaine de la cryptanalyse. Il a en particulier été utilisé par Adleman, Brickel, Lagarias et Odlyzko [3, 33, 34, 35, 98, 100] pour casser le cryptosystème de type « sac-à-dos » proposé par Merkle et Hellman [126] (une attaque proposée par Shamir [160] existait déjà, mais seulement pour la variante la plus simple du crytosystème). Cette attaque a ensuite été généralisée pour cryptanalyser d'autres schémas de type « sac-à-dos » (voir par exemple [142]). L'algorithme dû à Lenstra, Lenstra et Lovász est toujours un outil très utilisé en cryptanalyse de cryptosystèmes à clé publique, comme par exemple certaines variantes rapides de RSA [22, 26, 43, 44, 60, 123] et certaines variantes rapides du schéma de signatures DSA [82, 137]. L'algorithme LLL a aussi permis de montrer que trouver l'exposant privé d'une clé RSA est calculatoirement équivalent à factoriser le module [125] (il permet de construire une réduction polynomiale

déterministe). Il ne s'agit que de quelques exemples parmi beaucoup d'autres, le lecteur est renvoyé aux deux panoramas des applications cryptographiques de cet algorithme qui sont dressés dans [85, 141]. Un autre domaine d'application important de l'algorithme LLL est la théorie algorithmique des nombres : il a permis d'invalider la conjecture de Mertens [143], il sert aussi pour calculer des polynômes minimaux (par exemple de nombres algébriques), ou encore pour travailler dans les corps de nombres [40].

Depuis 1982, de nombreuses variantes et améliorations de l'algorithme LLL ont été proposées, que ce soit vis-à-vis du temps d'exécution [87, 94, 151, 156, 174], vis-à-vis de la qualité de la base renvoyée [150, 152], ou encore pour étendre le champ des entrées possibles [146].

Le contenu du présent mémoire se partage en deux types de sujets : d'une part nous présentons de nouveaux algorithmes de réductions de réseaux, de la dimension 1 (on a alors un problème de plus grand diviseur commun), à la dimension quelconque (comme l'algorithme LLL), en passant par la petite dimension (dimensions $2, 3$ et 4) ; d'autre part, nous décrivons de nouvelles applications de la réduction des réseaux, dans le domaine de l'arithmétique des ordinateurs. Cette discipline consiste entre autres à étudier l'implantation des nombres réels en machine. L'alternative la plus répandue pour cela est l'arithmétique flottante, qui est normalisée pour un certain nombre de précisions et pour les opérations arithmétiques de base (addition, soustraction, multiplication, division et racine carrée) par la norme IEEE-754 [2]. Pour certaines applications scientifiques et industrielles, il devient aujourd'hui important d'étendre cette norme à quelques fonctions mathématiques élémentaires, parmi lesquelles des fonctions exponentielles, logarithmiques et trigonométriques. Cependant, des inconnues demeurent pour déterminer la « bonne » méthode pour implanter ces fonctions. Nous utilisons la réduction de réseaux pour résoudre certaines difficultés liées à l'implantation des fonctions mathématiques, en particulier nous proposons un nouvel algorithme pour résoudre le « dilemme du fabricant de tables » [107]. Apporter une réponse à ce dilemme revient à déterminer à quel point l'image par la fonction donnée d'un nombre « représentable » peut être proche d'un nombre représentable ou du milieu de deux nombres représentables : de tels nombres ont des images très difficiles à arrondir puisqu'il faut beaucoup de bits pour décider du sens d'arrondi. La méthode présentée sert aussi à déterminer dans quelle mesure la suite des bits de l'image par la fonction donnée d'un nombre représentable est aléatoire. Nous améliorons aussi la méthode des tables de Gal [62], qui est une technique classique pour implanter certaines fonctions mathématiques. Le travail présenté ici donne ainsi des arguments supplémentaires aux partisans de la standardisation des fonctions élémentaires, parmi lesquels figurent les projets ARÉNAIRE et SPACES de l'INRIA. Parmi les travaux de ces deux équipes qui améliorent la compréhension et l'implantation des fonctions élémentaires, on peut citer [36, 37, 51, 52, 103, 109, 147, 177].

PLAN. Cette thèse comporte neuf chapitres partagés en trois parties.

Dans la première partie, nous rappelons certains résultats de base sur les réseaux euclidiens (chapitre I-1), et sur l'arithmétique flottante (chapitre I-2). Il s'agit de rappels nécessaires pour les autres chapitres, ils ne comportent aucun résultat nouveau. Dans les deux chapitres en question, nous avons aussi essayé de donner le contexte (difficulté des

problèmes sous-jacents, normalisation) des sujets abordés.

La partie II est consacrée à l'étude algorithmique des réseaux euclidiens. Dans le chapitre II-1, nous décrivons et étudions un nouvel algorithme de calcul de plus grand diviseur commun, problème qui peut être considéré comme une réduction de réseau en dimension 1. Au chapitre II-2, nous analysons un algorithme très naturel de réduction dont la sortie est en un certain sens optimale jusqu'en dimension 4. Ensuite, au chapitre II-3, nous décrivons un nouvel algorithme de réduction de réseaux utilisant de l'arithmétique flottante, et qui admet la meilleure borne de complexité actuelle vis-à-vis de la taille des vecteurs donnés en entrée. Cet algorithme a été implanté et nous avons étudié son comportement pratique : nous donnons des détails relatifs à ces expérimentations au chapitre II-4.

Dans la partie III, l'algorithmique de la géométrie des nombres devient un outil pour résoudre un certain nombre de questions liées à l'arithmétique des ordinateurs. Le chapitre III-1 est consacré à la description d'un nouvel algorithme et de son analyse pour trouver des mauvais cas pour l'arrondi des fonctions mathématiques élémentaires, c'est-à-dire le dilemme du fabricant de tables. Au chapitre III-2, nous adaptons la méthode ainsi développée au chapitre III-1 pour trouver des mauvais cas simultanés de deux fonctions, ce qui nous a servi à améliorer une méthode pour implanter les fonctions mathématiques à l'aide de tables. Enfin, dans le chapitre III-3, nous modifions la méthode développée au chapitre III-1 pour cryptanalyser un générateur de nombres pseudo-aléatoire qui avait été décrit par J. E. Littlewood dans les années 1950.

Pour la plupart des chapitres nous avons mis en ligne à l'URL suivante un certain nombre de documents ayant un lien direct avec les questions abordées :

`http://www.loria.fr/~stehle/these.html`

On y trouvera en particulier des programmes en langage C qui correspondent aux algorithmes décrits dans les chapitres II-3, II-4, III-1 et III-2.

Table des matières

Introduction. **iii**

Notations. **xiii**

Table des figures **xv**

Partie I Préliminaires. 1

Chapitre I-1 Éléments de géométrie des nombres. **3**

1.1 Définition et représentation. 4

1.2 Quelques invariants d'un réseau. 7

1.2.1 Minima d'un réseau euclidien. 7

1.2.2 Rayon de recouvrement. 9

1.2.3 Volume. 11

1.3 Théorèmes de Minkowski et réductions. 12

1.3.1 Les deux théorèmes de Minkowski. 12

1.3.2 Des réductions fortes, mais difficiles à obtenir. 13

1.3.3 Des réductions plus faibles, mais calculables efficacement. 15

Chapitre I-2 Éléments d'arithmétique flottante. **17**

2.1 Nombres flottants et norme IEEE-754. 18

2.1.1 Nombres flottants. 18

2.1.2 Opérations arithmétiques et modes d'arrondi. 19

2.2 Deux extensions de la norme IEEE-754. 19

2.2.1 Normalisation des fonctions élémentaires. 19

2.2.2 Calculs en multiprécision. 20

2.3 Sommation de nombres flottants. 21

Partie II Algorithmique de la réduction des réseaux. 23

Chapitre II-1 L'algorithme rapide de pgcd binaire. 25

1.1 L'algorithme de Knuth-Schönhage. 26
1.2 La division binaire généralisée. 27
1.2.1 La division binaire généralisée. 29
1.2.2 L'algorithme d'Euclide binaire généralisé. 32
1.2.3 Sur la taille réelle des restes binaires généralisés. 34
1.2.4 Calcul d'inverses modulaires. 37
1.3 L'algorithme binaire rapide. 37
1.4 Correction et analyse de complexité de l'algorithme binaire rapide. . . 40
1.4.1 Correction de l'algorithme **Pgcd-BG-rapide**. 40
1.4.2 Analyse de complexité de l'algorithme **Pgcd-BG-rapide**. . . . 42
1.5 Remarques sur l'implantation des algorithmes rapides de calcul de pgcd. 43

Chapitre II-2 La réduction de réseaux en petite dimension 47

2.1 Préliminaires. 50
2.1.1 Minima successifs. 51
2.1.2 Réduction au sens de Minkowski. 51
2.1.3 Cellule de Voronoï. 53
2.2 Deux analyses de l'algorithme de Lagrange. 56
2.2.1 L'analyse globale de l'algorithme de Lagrange. 57
2.2.2 L'analyse locale de l'algorithme de Lagrange. 58
2.3 Une généralisation gloutonne de l'algorithme de Lagrange. 59
2.3.1 L'algorithme glouton de réduction. 59
2.3.2 Une description itérative de l'algorithme glouton de réduction. . 61
2.3.3 G-réduction. 61
2.4 Le problème CVP en petite dimension. 63
2.5 L'approche globale. 65
2.5.1 Deux phases dans l'algorithme glouton récursif. 66
2.5.2 L'algorithme glouton avec une base plutôt orthogonale. 67
2.6 La preuve de la complexité quadratique. 68
2.7 L'approche locale. 70
2.7.1 Une analyse géométrique unifiée jusqu'en dimension 4. 71
2.7.2 Fin de l'analyse en dimension 4. 76

2.8 Résultats sur la géométrie des réseaux en petite dimension. 80
2.8.1 Les cellules de Voronoï pour des bases réduites au sens de Minkowski. 80
2.8.2 Les cellules de Voronoï pour des vecteurs ε-G-réduits. 84
2.8.3 Le lemme du vide. 88
2.9 Qu'en est-il en plus grande dimension ? 94

Chapitre II-3 L'algorithme LLL en arithmétique flottante 97
3.1 Rappels sur l'algorithme LLL. 102
3.1.1 Orthogonalisation et LLL-réduction. 102
3.1.2 L'algorithme LLL. 103
3.2 L'algorithme L^2. 108
3.2.1 Orthogonalisation en arithmétique flottante. 108
3.2.2 Un algorithme paresseux de proprification. 110
3.2.3 L'algorithme L^2. 111
3.3 Correction de l'algorithme L^2. 113
3.3.1 Acuité lors des calculs d'orthogonalisation. 113
3.3.2 Acuité des calculs de proprification. 117
3.3.3 Correction de l'algorithme L^2. 121
3.4 Analyse de complexité de l'algorithme L^2. 122
3.4.1 Une analyse très naïve de l'algorithme d'Euclide. 123
3.4.2 Une analyse amortie en dimension arbitraire. 124
3.5 Conclusion et problèmes ouverts. 126

Chapitre II-4 Résultats expérimentaux autour de l'algorithme LLL. 129
4.1 Implantation de l'algorithme LLL flottant. 131
4.1.1 La variante prouvée. 132
4.1.2 Les variantes heuristique et rapide. 134
4.1.3 Les bases aléatoires étudiées. 135
4.2 Étude expérimentale de la qualité de la sortie. 136
4.2.1 Étude du défaut d'orthogonalité. 137
4.2.2 Étude des bases locales. 138
4.2.3 L'insertion profonde de Schnorr-Euchner. 141
4.3 Remarques sur le temps d'exécution de l'algorithme L^2. 142
4.3.1 Mesure du temps d'exécution de l'algorithme L^2. 142

4.3.2 Cas des réseaux sac-à-dos. 145
4.3.3 Évolution du temps d'exécution avec les facteurs (δ, η). 147
4.4 Étude expérimentale de la précision numérique nécessaire. 147
4.4.1 Comportement le pire de l'algorithme LLL flottant. 148
4.4.2 Comportement « moyen » de l'algorithme LLL flottant. 148
4.4.3 À propos de l'orthogonalisation de Givens et Householder. . . . 150
4.5 Problèmes ouverts. 152

Partie III Pires cas pour l'arrondi de fonctions. 153

Chapitre III-1 Recherche des pires cas de fonctions mathématiques. 155
1.1 Méthode de Coppersmith. 160
1.1.1 Petites racines de polynômes modulaires univariés. 160
1.1.2 Petites racines de polynômes modulaires multivariés. 161
1.2 La méthode de Lefèvre et sa généralisation. 162
1.2.1 Motivations. 162
1.2.2 La méthode de Lefèvre. 164
1.2.3 Généralisation de la méthode de Lefèvre. 165
1.2.4 Principaux résultats. 168
1.3 Une première analyse de l'algorithme. 171
1.3.1 Deux bornes dans l'analyse de complexité. 171
1.3.2 Première analyse de complexité. 172
1.3.3 Le cas $d = \alpha = 2$. 173
1.4 Le cas $d = 2$. 176
1.4.1 Borne sur le déterminant du réseau. 176
1.4.2 Borne de Coppersmith et conclusion. 179
1.5 Le cas $d \geqslant 3$. 180
1.5.1 La famille de polynômes choisie. 180
1.5.2 Une méthode pour borner le déterminant d'une matrice. 181
1.5.3 Calcul du déterminant de la matrice B_3 (introduite au paragraphe 1.5.1). 182
1.5.4 Borne de Coppersmith et conclusion. 188
1.6 Le cas de fonctions de plusieurs variables. 189
1.6.1 L'analyse naïve. 190

1.6.2 Analyse du cas $d = 2$. 191
1.6.3 Analyse du cas $d \geqslant 3$. 193
1.7 Données expérimentales. 195
1.7.1 Recherche des pires cas d'une fonction d'une variable. 196
1.7.2 Recherche de pires cas de fonctions de deux variables. 198
1.7.3 Calculs de bornes supérieures pour l'arrondi correct, et inversion d'une suite de bits. 199
1.8 Problèmes ouverts. 200

Chapitre III-2 Amélioration de la méthode de Gal. 203

2.1 La méthode classique de Gal. 207
2.1.1 Le modèle aléatoire. 207
2.1.2 La méthode classique de Gal. 207
2.2 Amélioration de la méthode de Gal. 209
2.2.1 Le cas d'une seule fonction associée. 209
2.2.2 Le cas de deux fonctions associées. 210
2.3 Le calcul des tables. 214
2.3.1 Recherche des mauvais cas simultanés de deux fonctions. 215
2.3.2 Correction de l'algorithme. 217
2.3.3 Analyse de complexité de l'algorithme. 218
2.4 Résultats expérimentaux. 219
2.4.1 La première table pour 2^x. 219
2.4.2 La table pour $\sin x$. 220
2.4.3 La table pour 2^x et $\ln 2 \cdot 2^x$. 220
2.5 Généralisation possible. 221

Chapitre III-3 Cryptanalyse du schéma de chiffrement de J. E. Littlewood. 223

3.1 Présentation et cryptanalyse du schéma original de Littlewood. 224
3.2 Une modification du schéma de Littlewood. 227
3.2.1 Randomisation du générateur de Littlewood. 228
3.3 Une attaque reposant sur l'approximation polynomiale. 229
3.3.1 Approcher f par un polynôme. 229
3.3.2 D'un problème sur les réels à un problème sur les entiers. 230
3.3.3 Comment retrouver a avec de la réduction de réseau. 231

3.4 Conclusion et question ouverte. 232

Conclusion. **235**

Bibliographie **239**

Notations.

Les notations les plus souvent utilisées dans ce document sont listées ci-dessous. Éventuellement elles pourront avoir d'autres significations, mais cela sera indiqué explicitement.

- Par défaut, on se placera dans le modèle de calcul RAM, et on comptera les opérations élémentaires sur les bits.
- On utilisera la notation $M(n)$ pour désigner le coût d'une multiplication de deux entiers d'au plus n bits chacun avec l'algorithme de Schönhage et Strassen [158] : $M(n) = O(n \log n \log\log n)$.
- La notation $\mathcal{P}\mathrm{ol}(k)$ signifiera « polynôme en k ».
- Pour tous les résultats de complexité concernant les réseaux, il est sous-entendu que le réseau est entier, c'est-à-dire qu'il est inclus dans $\mathbb{Z}^n$ pour un certain entier n.

- La fonction lg désignera le logarithme en base 2, et log le logarithme népérien.
- Si a est un entier non nul, sa longueur $l(a)$ est le nombre de chiffres de son écriture en base 2, c'est-à-dire $l(a) = \lceil \lg(|a|+1) \rceil$.
- Si a est un entier non nul, sa valuation 2-adique $\nu_2(a)$ est le nombre de zéros consécutifs dans les chiffres de poids faible de son écriture en base 2. On définit $\nu_2(0) = \infty$.
- Si $n \in \mathbb{N}$, le groupe des permutations de l'ensemble $[|1, n|]$ sera désigné par $\mathcal{S}_n$.
- Les coefficients binomiaux sont notés $\binom{n}{k}$ pour $0 \leqslant k \leqslant n$. Nous rappelons l'identité $\binom{n}{k} = \frac{n!}{k!(n-k)!}$.

- On utilisera les notations classiques $\lfloor \cdot \rfloor$, $\lfloor \cdot \rceil$ et $\lceil \cdot \rceil$ pour la partie entière inférieure, l'entier le plus proche et la partie entière supérieure. Dans le cas de l'entier le plus proche, s'il y a deux solutions possibles, on prendra la solution paire.
- Soient $a \leqslant b$ deux nombres réels. On écrira $[|a, b|]$ l'ensemble des entiers dans l'intervalle $[a, b]$.
- Si a et b sont deux réels, le reste centré $r = a \bmod b$ est le réel $r \in \left]-\frac{b}{2}, \frac{b}{2}\right]$ tel qu'il existe un entier k pour lequel $r = a + kb$.
- On appellera bits fractionnaires d'un nombre réel x les chiffres de sa représentation binaire qui se trouvent après la virgule. Si $n \geqslant 1$ est un entier et $a < b$, on écrira $[a, b]_n$ l'ensemble des nombres réels compris entre a et b et qui ont au plus n bits fractionnaires dans la représentation binaire classique, c'est-à-dire :

$$\left\{ \frac{\lceil 2^n a \rceil}{2^n}, \frac{\lceil 2^n a \rceil + 1}{2^n}, \ldots, \frac{\lfloor 2^n b \rfloor}{2^n} \right\}.$$

  Si $k_2 \geqslant k_1 \geqslant 0$ sont des entiers et x un nombre réel, les bits fractionnaires du rang k_1

au rang k_2 seront représentés par $\mathrm{bits}_{k_1..k_2}[x]$. Plus exactement :

$$\mathrm{bits}_{k_1..k_2}[x] = \left(\lfloor 2^{k_2} x \rfloor \mod 2^{k_2-k_1+1}\right) \in [|0, 2^{k_2-k_1+1}|[.$$

- Si x est un réel, on écrira $\diamond(x)$ un nombre flottant le plus proche de x. L'ensemble des nombres flottants considérés se déduira facilement du contexte, et le non-déterminisme de la notation $\diamond(x)$ ne posera pas de difficulté.

- Tous les vecteurs, et uniquement les vecteurs, sont décrits par des variables en gras, comme par exemple $\mathbf{b}$.
- La norme euclidienne d'un vecteur $\mathbf{b}$ sera notée $\|\mathbf{b}\|$, sa norme ℓ_i sera notée $\|\mathbf{b}\|_i$ et en particulier sa norme infinie sera notée $\|\mathbf{b}\|_\infty$.
- La notation $\mathcal{B}(\mathbf{t}, r)$ désigne la boule de centre $\mathbf{t}$ et de rayon r. La dimension de l'espace considéré sera évidente à partir du contexte.
- L'ensemble des matrices à n lignes et m colonnes à coefficients dans l'anneau R est représenté par $\mathcal{M}_{n,m}(R)$.

- Le résultant de deux polynômes $P_1(x_1, \ldots, x_n)$ et $P_2(x_1, \ldots, x_n)$ par rapport à la variable x_n sera noté $\mathrm{Res}_{x_n}(P_1, P_2)$.

- Si X est une variable aléatoire, on désignera son espérance par $\mathbb{E}[X]$.

Table des figures

1.1 Le réseau est l'ensemble des points d'intersection de la grille. 6
1.2 Le même réseau donné par une autre grille. 6
1.3 Trois vecteurs dans l'espace, et leur orthogonalisation de Gram-Schmidt. . 7
1.4 Le problème du plus proche vecteur. 10
1.5 Le volume d'un réseau est le volume du parallélépipède fondamental de toute base du réseau. 11

2.1 Nombres flottants dans les cinq précisions classiques. 18

1.1 L'algorithme **Demi-pgcd**. 28
1.2 L'algorithme **Pgcd-rapide**. 28
1.3 Représentation graphique de l'algorithme **Demi-pgcd**. 29
1.4 L'algorithme d'Euclide binaire classique. 30
1.5 Un algorithme naïf de division binaire généralisée. 32
1.6 Un exemple d'exécution de l'algorithme **DBG-élémentaire**. 32
1.7 Un algorithme rapide de division binaire généralisée. 33
1.8 L'algorithme d'Euclide binaire généralisé. 33
1.9 Un exemple d'exécution de l'algorithme **Euclide-BG**. 34
1.10 L'algorithme **Demi-pgcd-BG**. On fixera par exemple $k = \lfloor l(a)/2 \rfloor$. . . 38
1.11 Représentation graphique de l'algorithme **Demi-pgcd-BG**. 39
1.12 L'algorithme **Pgcd-BG-rapide**. 39

2.1 L'enchaînement des preuves pour l'analyse de complexité de l'algorithme glouton de la section 2.3. 50
2.2 Une cellule et les vecteurs de Voronoï d'un réseau planaire. 54
2.3 Des vecteurs de Voronoï non stricts. 54
2.4 L'algorithme de Lagrange. 56
2.5 Un exemple d'exécution de l'algorithme de Lagrange. 57
2.6 L'algorithme η-Lagrange. 58
2.7 L'algorithme glouton de réduction en dimension d. 60
2.8 Une description itérative de l'algorithme glouton de réduction. 61
2.9 L'algorithme η-**Glouton** de réduction en dimension d. 67
2.10 Le lemme du vide en dimension 2. Un vecteur de l'extérieur de la couronne qui est transformé en un vecteur de la cellule de Voronoï voit sa norme diminuer significativement. 76

2.11 Une généralisation de l'algorithme glouton de réduction. 95

3.1 Comparaison de différentes variantes de l'algorithme LLL. 100
3.2 L'algorithme LLL. 104
3.3 L'algorithme de proprification. 104
3.4 L'algorithme de factorisation de Cholesky. 110
3.5 L'algorithme paresseux de proprification. 111
3.6 L'algorithme L^2. 112
3.7 L'algorithme de proprification flottant classique. 117

4.1 Pseudo-code de la variante prouvée de `fplll-1.2`. 133
4.2 Évolution de la quantité $\frac{1}{d^2}\lg\frac{\|\mathbf{b}_1\|^d}{\det L}$ en fonction de la dimension, en partant de bases sac-à-dos (à gauche), et en partant de bases d'Ajtai (à droite). . . 137
4.3 Évolution de la quantité $\frac{1}{d^2}\lg\frac{\|\mathbf{b}_1\|^d}{\det L}$ pour $d = 95$, en fonction du facteur δ. . 138
4.4 Forme des bases locales après LLL-réduction, à gauche, et après LLL-réduction avec insertion profonde (voir le paragraphe 4.2.3), à droite. . . . 139
4.5 Répartition des $\mu_{i,i-1}$ (en haut à gauche), $\mu_{i,i-2}$ (en haut à droite), $\mu_{i,i-5}$ (en bas à gauche) et $\mu_{i,i-10}$ (en bas à droite), pour des réseaux de dimension 71, après LLL-réduction. 140
4.6 L'algorithme d'insertion profonde. 141
4.7 Croissance du nombre d'itérations de boucle dans l'exécution de l'algorithme LLL en fonction de la dimension, pour des réseaux aléatoires de type « Ajtai ». 144

1.1 Le graphe d'une fonction et la grille des nombres représentables. 163
1.2 L'algorithme de résolution de l'équation (1.1). 166
1.3 L'algorithme de recherche de mauvais cas du polynôme $P(x)$. 166
1.4 Lien entre les indices i, j et k lorsque $\left(\frac{N_1}{T}\right)^d \leqslant MN_2 \leqslant \left(\frac{N_1}{T}\right)^{d+1}$. 185
1.5 Lien entre les indices i, j et k lorsque $\frac{N_1}{T} \leqslant MN_2 \leqslant \left(\frac{N_1}{T}\right)^d$. 185
1.6 Lien entre les indices i, j et k lorsque $MN_2 \leqslant \frac{N_1}{T}$. 185
1.7 Temps de calcul estimé pour la recherche du pire cas de $\exp x$ sur $[1/2, 1[$, sur un Opteron 2.4GhZ. 197
1.8 Temps de calcul estimé pour l'inversion d'une suite de bits générée par la fonction $\exp x$ sur $[1/2, 1[$ pour différents paramètres, sur un Opteron 2.4GhZ. 200

2.1 Les deux phases de l'évaluation de $f(x)$. 204
2.2 La phase rapide. 205
2.3 L'algorithme de recherche de mauvais cas simultanés. 217

3.1 Un exemple de fonctionnement du générateur pseudo-aléatoire de bits de Littlewood, avec $N = 30, t = 5, n' = 7$ et $m = 9$. 226
3.2 Applications successives de l'opérateur Δ à la séquence obtenue à la figure 3.1. Dans la dernière colonne, les bits dominants de chaque ligne sont nuls. 227
3.3 Une proposition de fonction à sens unique. 233

Première partie

Préliminaires.

Chapitre I-1

Éléments de géométrie des nombres.

J.W.S. Cassels, 1959 — *We owe to Minkowski the fertile observation that certain results which can be made almost intuitive by the consideration of figures in n-dimensional euclidean space have far-reaching consequences in diverse branches of number theory.*

Sommaire

1.1 Définition et représentation. 4
1.2 Quelques invariants d'un réseau. 7
1.2.1 Minima d'un réseau euclidien. 7
1.2.2 Rayon de recouvrement. 9
1.2.3 Volume. 11
1.3 Théorèmes de Minkowski et réductions. 12
1.3.1 Les deux théorèmes de Minkowski. 12
1.3.2 Des réductions fortes, mais difficiles à obtenir. 13
1.3.3 Des réductions plus faibles, mais calculables efficacement. . 15

Dans ce chapitre, nous rappelons quelques définitions et propriétés de base de la théorie des réseaux euclidiens, qui nous seront utiles tout au long des chapitres suivants. Au lieu d'adopter la présentation classique qui consiste à donner dans un premier temps les définitions mathématiques puis dans un deuxième temps les principaux résultats algorithmiques, nous avons choisi de discuter les résultats algorithmiques liés à chaque concept mathématique introduit, au moment où ce dernier est introduit. Les résultats d'existence sont ainsi mis en relief immédiatement avec les résultats de constructibilité et d'effectivité qui leur correspondent.

Dans la section 1.1, nous donnons les définitions des réseaux, et discutons leur représentation. Dans la section 1.2, nous donnons quelques invariants des réseaux euclidiens et les problèmes algorithmiques qui leur sont liés. Certains de ces invariants sont faciles à évaluer, et d'autres semblent difficiles. Le théorème de Minkowski et la notion de réduction permettent d'obtenir de l'information sur les invariants difficiles à calculer à partir de ceux qui sont faciles à évaluer, cela est étudié à la section 1.3.

Les résultats annoncés dans ce chapitre viennent sans preuve. Nous renvoyons le lecteur à des ouvrages classiques comme [38, 42, 122, 164] pour les aspects mathématiques, et [40, 118, 130] pour les aspects algorithmiques.

1.1 Définition et représentation.

Naïvement, un réseau euclidien est l'ensemble des points d'une grille, ou des sommets d'un cristal, dans un espace euclidien de dimension finie.

Définition 1 (Réseau euclidien). *Soit $n \geqslant 0$. Un réseau euclidien L est un sous-groupe discret de $\mathbb{R}^n$ muni de sa structure euclidienne naturelle. Un ensemble $L' \subset \mathbb{R}^n$ est un sous-réseau du réseau L si $L' \subset L$ et si L' est un réseau.*

Les réseaux non triviaux les plus simples sont certainement les points de l'espace à coordonnées entières $\mathbb{Z}^n$, mais tous les sous-groupes de $\mathbb{Z}^n$ sont aussi des réseaux. En particulier, $\mathbb{Z}^k \times \{0\}^{n-k}$ est un sous-réseau de $\mathbb{Z}^n$ pour tout $k \leqslant n$. L'ensemble des points de $\mathbb{Z}^n$ dont la somme des coordonnées est paire en est un autre. On définit un réseau entier comme un sous-groupe de $\mathbb{Z}^n$. S'il n'est pas trivial, un réseau est de cardinalité infinie, et n'est donc pas représentable tel quel. En fait, pour le représenter, on utilise le fait qu'un tel ensemble est un module de type fini, et peut donc s'écrire comme ensemble de combinaisons linéaires à coefficients entiers d'une certaine base.

Définition 2 (Bases). *Soient $\mathbf{b}_1, \ldots, \mathbf{b}_d \in \mathbb{R}^n$ des vecteurs linéairement indépendants. Ils génèrent le réseau :*

$$L(\mathbf{b}_1, \ldots, \mathbf{b}_d) = \left\{ \sum_{i=1}^{d} x_i \mathbf{b}_i, \forall i, x_i \in \mathbb{Z} \right\}.$$

On dit que les vecteurs $\mathbf{b}_1, \ldots, \mathbf{b}_d \in \mathbb{R}^n$ forment une base de ce réseau. La forme quadratique définie positive naturellement associée à la base $\mathbf{b}_1, \ldots, \mathbf{b}_d$ est $q(x_1, \ldots, x_d) = \|x_1\mathbf{b}_1 + \ldots + x_d\mathbf{b}_d\|^2$.

Proposition 1. *Soit L un réseau de $\mathbb{R}^n$. Alors L admet une base, c'est-à-dire il existe des vecteurs $\mathbf{b}_1, \ldots, \mathbf{b}_d \in \mathbb{R}^n$ linéairement indépendants tels que $L = L(\mathbf{b}_1, \ldots, \mathbf{b}_d)$. La cardinalité d de toutes les bases de L est identique, et s'appelle la dimension de L, que l'on notera* $\dim L$.

À un réseau donné L sont donc associées deux dimensions : la dimension d du réseau lui-même, et sa dimension de plongement n. L'invariant d est plus intrinsèque que n, car le réseau L peut être transformé en un réseau L' de dimension de plongement d par une isométrie. Par ailleurs, la dimension de plongement n'a plus de signification du tout si l'on considère la forme quadratique associée.

On représente un réseau donné L par l'une de ses bases $\mathbf{b}_1, \ldots, \mathbf{b}_d$, ou plus précisément par l'expression d'une de ses bases sous forme de matrice dont les colonnes sont les coordonnées des vecteurs de la base dans l'espace de plongement. Cette représentation n'est pas unique, mais elle a l'avantage d'être finie et effective. La matrice en question a n

lignes et d colonnes — si l'on représente les vecteurs sous forme de colonnes. Avec cette représentation, il est simple de tester l'appartenance d'un vecteur $\mathbf{t}$ au réseau L, car il suffit de résoudre le système de n équations et d inconnues suivant :

$$x_1\mathbf{b}_1 + \ldots + x_d\mathbf{b}_d = \mathbf{t},$$

et de vérifier si les solutions $x_1, \ldots, x_d$ obtenues sont des entiers. Il suffit par exemple d'utiliser la méthode du pivot de Gauss (voir [74] pour une analyse détaillée). Lorsque l'on donne des résultats de complexité sur des problèmes liés aux réseaux, on se restreint aux réseaux entiers[1]. L'entrée du problème est alors une matrice d'entiers écrits en représentation binaire, et le temps de calcul est mesuré par rapport à la taille de l'entrée. Dans le cas présent de l'appartenance d'un point à un réseau, le problème peut être résolu en temps polynomial, c'est-à-dire avec un nombre d'opérations élémentaires croissant de façon au plus polynomiale en d, n et la taille des entiers qui sont les coordonnées des vecteurs $\mathbf{b}_1, \ldots, \mathbf{b}_d$ et $\mathbf{t}$.

Tester qu'un réseau L' est un sous-réseau d'un réseau L est aussi réalisable en temps polynomial : il suffit de tester que chaque vecteur de la base donnée du réseau L' appartient bien au réseau L. Une conséquence immédiate est que l'on peut tester en temps polynomial si deux bases représentent un même réseau.

Dès que $d \geqslant 2$, un réseau donné admet une infinité de bases. Celles-ci sont liées entre elles par des transformations unimodulaires, c'est-à-dire des matrices carrées de dimension d à coefficients entiers et de déterminant ± 1. Autrement dit, à partir d'une base donnée $\mathbf{b}_1, \ldots, \mathbf{b}_d$, les opérations valides pour conserver une base sont $\mathbf{b}_i \to -\mathbf{b}_i$ pour un certain $i \leqslant d$, $\mathbf{b}_i \to \mathbf{b}_i + \mathbf{b}_j$ pour $i \neq j$, $\mathbf{b}_i \leftrightarrow \mathbf{b}_j$ pour $i, j \leqslant d$, et les combinaisons de telles opérations. On peut aussi tester si deux bases engendrent le même réseau en calculant la matrice de passage, et en vérifiant si celle-ci est unimodulaire. Aux figures 1.1 et 1.2, nous donnons deux différentes représentations d'un même réseau.

Un outil fondamental pour étudier les bases d'un réseau donné est l'orthogonalisation de Gram-Schmidt.

Définition 3 (Orthogonalisation de Gram-Schmidt). *Soient $\mathbf{b}_1, \ldots, \mathbf{b}_d$ des vecteurs linéairement indépendants. L'orthogonalisation de Gram-Schmidt de la base $\mathbf{b}_1, \ldots, \mathbf{b}_d$ est la famille de vecteurs $\mathbf{b}_1^*, \ldots, \mathbf{b}_d^*$ définie par :*

$$\begin{aligned} \mathbf{b}_1^* &= \mathbf{b}_1 \\ \mathbf{b}_i^* &= \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j}\mathbf{b}_j^* \quad si \quad i \geqslant 2, \end{aligned}$$

où $\mu_{i,j} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\|\mathbf{b}_j^*\|^2}$ pour tous $j < i$.

[1] De nombreux résultats algorithmiques peuvent être adaptés à des réseaux à coordonnées réelles, mais nécessitent de décrire comment sont représentés les réels. Cela soulève des problèmes techniques qui sont hors du contexte de cette thèse. Dans le chapitre III-1, nous étudierons d'un point de vue algorithmique des réseaux à coordonnées non entières, mais nous montrerons comment se ramener au cas des réseaux entiers dans cette situation.

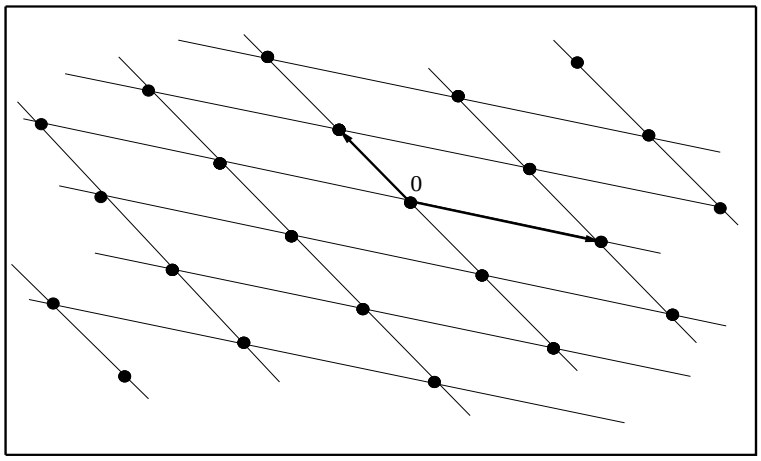


FIG. 1.1 – Le réseau est l'ensemble des points d'intersection de la grille.

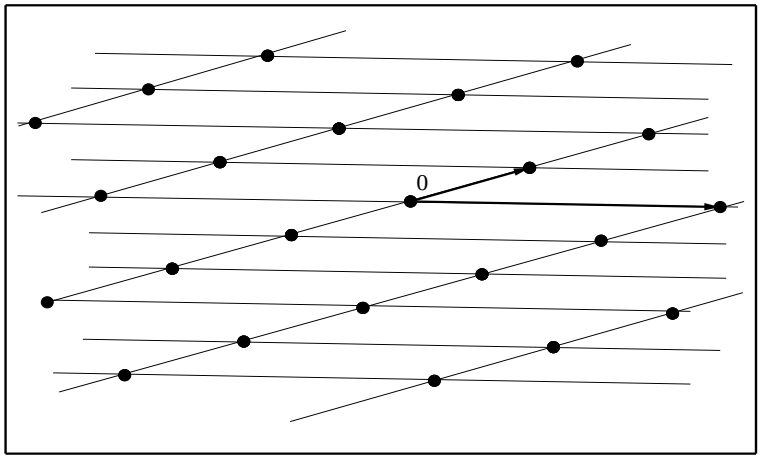


FIG. 1.2 – Le même réseau donné par une autre grille.

La figure 1.3 illustre la définition précédente. On observe aisément que si les vecteurs $\mathbf{b}_1, \ldots, \mathbf{b}_d$ ont des coordonnées entières, alors les coordonnées des vecteurs $\mathbf{b}_1^*, \ldots, \mathbf{b}_d^*$ sont rationnelles. De plus, les vecteurs $\mathbf{b}_i^*$ peuvent être calculés à partir des vecteurs $\mathbf{b}_i$ en temps polynomial : le point central pour montrer cela est de voir que les numérateurs et dénominateurs des quantités impliquées lors du calcul sont de taille bornée polynomialement en la taille de l'entrée (voir par exemple la preuve du théorème 17, page 104).

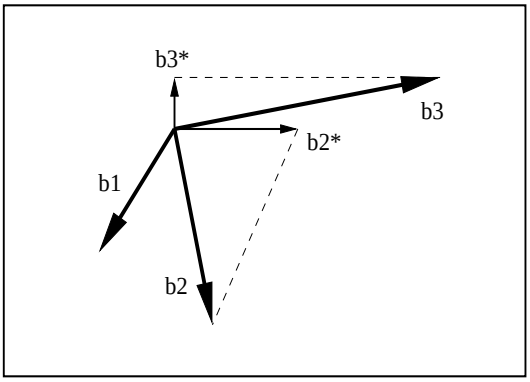


FIG. 1.3 – Trois vecteurs dans l'espace, et leur orthogonalisation de Gram-Schmidt.

1.2 Quelques invariants d'un réseau.

Étant donné un réseau L, on appelle une quantité liée à L un invariant si celle-ci ne dépend pas du choix de base de L que l'on pourrait faire : il s'agit d'une quantité intrinsèque au réseau, qui ne dépend pas de sa représentation. Nous avons déjà défini un invariant simple, la dimension. Nous définissons dans cette section les minima, le rayon de recouvrement et le volume, encore appelé déterminant.

1.2.1 Minima d'un réseau euclidien.

Puisqu'un réseau est un sous-groupe discret de $\mathbb{R}^n$, il existe un vecteur non nul lui appartenant qui est de norme minimale. Cette norme s'appelle le premier minimum du réseau.

Définition 4 (Premier minimum). *Soit L un réseau. Le premier minimum de L est défini par :*

$$\lambda_1(L) = \min\left(r, \mathcal{B}(\mathbf{0}, r) \cap L \neq \{\mathbf{0}\}\right).$$

Nous rappelons que $\mathcal{B}(\mathbf{0}, r)$ est la boule fermée de centre $\mathbf{0}$ et de rayon r.

Le premier minimum est toujours atteint : il existe toujours un vecteur du réseau de norme $\lambda_1(L)$. Étant donnée une base $\mathbf{b}_1, \ldots, \mathbf{b}_d$ d'un réseau L, renvoyer un vecteur atteignant $\lambda_1(L)$ constitue la version calculatoire du problème du plus court vecteur, aussi appelé SVP (*Shortest Vector Problem*). Ce problème est au centre de l'algorithmique des

réseaux euclidiens, et a été étudié de façon très intensive depuis plus de 25 ans. Nous récapitulons les principaux résultats concernant un ensemble de généralisations de sa variante décisionnelle.

Définition 5 (γ-SVP). *Soit $\gamma \geqslant 1$. Le problème γ-SVP est le problème de décision sous promesse[2] suivant :*

- *Les instances positives sont les couples (B, r) pour lesquels $B \in \mathcal{M}_{n,d}(\mathbb{Z}), r \in \mathbb{Q}$ et $\lambda_1(L) \leqslant r$, où L est le réseau engendré par les colonnes de la matrice B.*
- *Les instances négatives sont les couples (B, r) pour lesquels $B \in \mathcal{M}_{n,d}(\mathbb{Z}), r \in \mathbb{Q}$ et $\lambda_1(L) > \gamma r$, où L est le réseau engendré par les colonnes de la matrice B.*

Quand $\gamma = 1$, on obtient le problème SVP.

La complexité de la famille de problèmes γ-SVP est mesurée vis-à-vis de la dimension d du réseau, la taille des entrées de la matrice B n'intervenant que peu dans la difficulté du problème. Bien entendu, plus le facteur d'affaiblissement γ augmente, plus le problème γ-SVP devient facile. Le problème a été conjecturé NP-difficile pour $\gamma = 1$ dès le début des années 1980, par van Emde Boas [59], mais c'est plus de quinze années plus tard que cela a été prouvé : en 1998, Ajtai [7] a montré que pour $\gamma = 1$, le problème est NP-difficile pour des réductions randomisées. Ce résultat est aussi correct pour des réductions non uniformes — un calcul non comptabilisé dans le temps d'exécution peut être effectué avant que l'entrée soit donnée — et des réductions déterministes, en supposant correctes certaines conjectures sur la répartition des nombres premiers. Plus tard, Micciancio [129] a montré que le résultat d'Ajtai était encore correct pour un facteur de relâchement arbitrairement proche de $\sqrt{2}$, et, en 2004, Khot [90] a étendu le résultat à n'importe quelle constante. La réduction développée par Khot, différente de celle d'Ajtai et de Micciancio, permet aussi de montrer que le problème γ-SVP est NP-dur sous des réductions randomisées pour $\gamma = 2^{\left((\log d)^{1/2-\varepsilon}\right)}$, sous l'hypothèse que NP n'est pas inclus dans les problèmes résolubles en temps probabiliste $2^{\mathcal{P}\mathrm{ol}(\log d)}$. Du point de vue effectif, le meilleur algorithme déterministe pour résoudre le problème SVP est dû à Kannan [88] et a été amélioré par Helfrich [75]. Sa complexité est $d^{O(d)}$. Le meilleur algorithme probabiliste est celui de Ajtai, Kumar et Sivakumar [11], dont la complexité est $2^{O(d)}$.

Lorsque $\gamma = \frac{\sqrt{d}}{\log d}$, Goldreich et Goldwasser [68] ont montré que le problème γ-SVP n'est plus NP-difficile, à moins que la hiérarchie polynomiale ne s'effondre. Pour $\gamma = n^c$ pour une certaine constante c, Ajtai [6] a établi une équivalence de complexité entre les instances les plus difficiles et les instances moyennes, pour une distribution de probabilité « naturelle ». Enfin, le problème devient facile lorsque γ est proche de 2^d. En effet, Ajtai, Kumar et Sivakumar [11] ont montré comment le résoudre en temps probabiliste polynomial pour $\gamma = 2^{\frac{d \log\log d}{\log d}}$, et Schnorr [150] a donné un algorithme déterministe polynomial pour $\gamma = 2^{\frac{d(\log\log d)^2}{\log d}}$. Le célèbre algorithme LLL résout le problème en temps polynomial

[2] Le problème est constitué de l'union disjointe de deux langages disjoints, correspondant aux instances positives et aux instances négatives : si on donne à l'algorithme un mot n'appartenant à aucun des deux langages, alors la sortie n'est pas spécifiée. On parle de problème sous promesse parce que l'utilisateur promet en quelque sorte de donner une instance qui fait soit partie des instances positives soit partie des instances négatives.

déterministe pour $\gamma = c^d$ où c est une constante arbitrairement proche de $\sqrt{4/3}$.

De même que l'on a défini le premier minimum, on peut définir les minima successifs.

Définition 6 (Minima successifs). *Soit L un réseau de dimension d. Soit $i \leqslant d$. Le i-ème minimum du réseau L est défini par :*

$$\lambda_i(L) = \min\left(r, \dim(\mathcal{B}(\mathbf{0}, r) \cap L) = i\right).$$

Les minima successifs d'un réseau donné sont tous atteints — il existe des vecteurs du réseau de normes égales aux minima successifs, et peuvent l'être en particulier par des vecteurs linéairement indépendants. Par contre, ces vecteurs ne forment pas nécessairement une base dès que $d \geqslant 4$. En dimension inférieure ou égale à 4, il existe une famille de tels vecteurs qui est une base, et en dimension supérieure ou égale à 5, il existe des réseaux pour lesquels aucune base n'atteint tous les minima. Cependant, quelle que soit la dimension d, il existe toujours une base atteignant les $\min(4, d)$ premiers minima [184]. De plus amples détails sont donnés sur ces questions au chapitre II-2.

Si l'on dispose d'une base $\mathbf{b}_1, \ldots, \mathbf{b}_d$ d'un réseau L, alors les orthogonalisés de Gram-Schmidt donnent de l'information concernant les minima successifs.

Proposition 2. *Soit $\mathbf{b}_1, \ldots, \mathbf{b}_d$ une base d'un réseau L. Alors pour tout $i \in [|1, d|]$, on a :*

$$\lambda_i(L) \geqslant \min_{j \in [|i,d|]} \|\mathbf{b}_j^*\|.$$

En particulier, le premier minimum est minoré par un des $\|\mathbf{b}_j^*\|$.

1.2.2 Rayon de recouvrement.

Le rayon de recouvrement correspond à la distance maximale d'un point de l'espace au réseau.

Définition 7 (Rayon de recouvrement). *Soit L un réseau. Son rayon de recouvrement est défini par :*

$$\rho(L) = \max_{\mathbf{t} \in \mathrm{E}(L)} \min_{\mathbf{b} \in L} \|\mathbf{b} - \mathbf{t}\|,$$

où $\mathrm{E}(L)$ est l'espace vectoriel engendré par les points de L.

De même que pour les minima successifs, l'orthogonalisation de Gram-Schmidt permet d'obtenir facilement une borne sur le rayon de recouvrement : il quantifie donc la « perméabilité » du réseau.

Proposition 3. *Soit $\mathbf{b}_1, \ldots, \mathbf{b}_d$ une base d'un réseau L. Alors, on a :*

$$\rho(L) \leqslant \frac{1}{2}\sqrt{\sum_{i=1}^{d} \|\mathbf{b}_i^*\|^2}.$$

Le problème algorithmique naturellement associé au rayon de recouvrement est celui du plus proche vecteur, aussi appelé CVP (*Closest Vector Problem*) : étant donnés une base d'un réseau et un vecteur de l'espace de plongement (appelé vecteur cible), la version calculatoire du problème est de trouver un point du réseau minimisant la distance au vecteur cible. La figure 1.4 illustre le problème CVP. Une instance de ce problème peut admettre plusieurs solutions, mais leur nombre est borné par le « *kissing number* » [42]. Nous récapitulons les principaux résultats concernant un ensemble de généralisations de sa variante décisionnelle.

Définition 8 (γ-CVP). *Soit $\gamma \geqslant 1$. Le problème γ-CVP est le problème de décision sous promesse suivant :*

- *Les instances positives sont les triplets $(B, \mathbf{t}, r)$ pour lesquels $B \in \mathcal{M}_{n,d}(\mathbb{Z}), \mathbf{t} \in \mathcal{M}_{n,1}(\mathbb{Z}), r \in \mathbb{Q}$ et $\min_{\mathbf{b}\in L} \|\mathbf{b}-\mathbf{t}\| \leqslant r$, où L est le réseau engendré par les colonnes de la matrice B.*
- *Les instances négatives sont les triplets $(B, \mathbf{t}, r)$ pour lesquels $B \in \mathcal{M}_{n,d}(\mathbb{Z}), \mathbf{t} \in \mathcal{M}_{n,1}(\mathbb{Z}), r \in \mathbb{Q}$ et $\min_{\mathbf{b}\in L} \|\mathbf{b}-\mathbf{t}\| > \gamma r$, où L est le réseau engendré par les colonnes de la matrice B.*

Quand $\gamma = 1$, on obtient le problème CVP.

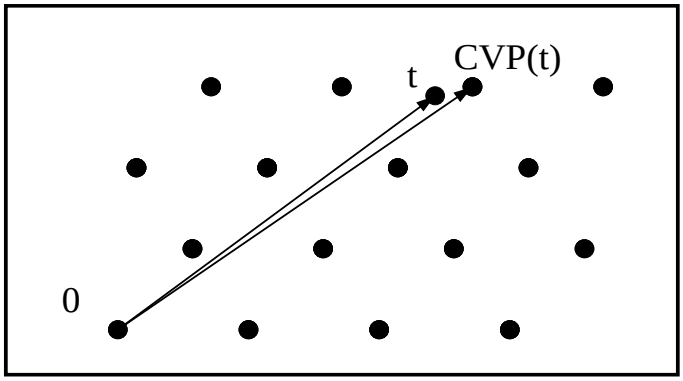


FIG. 1.4 – Le problème du plus proche vecteur.

Le problème CVP est connu comme étant NP-difficile depuis le début des années 1980, ce qui était la raison pour laquelle le problème SVP était déjà conjecturé NP-difficile [59]. Le meilleur algorithme déterministe pour résoudre ce problème est dû à Kannan [88] et à Helfrich [75], et sa complexité est $d^{O(d)}$. Le meilleur algorithme probabiliste a été donné par Ajtai, Kumar et Sivakumar [12], sa complexité est $2^{O(d)}$: il ne résout pas tout à fait CVP, mais la version affaiblie $(1+\varepsilon)$-CVP, où ε peut être arbitrairement proche de 0. Si on s'autorise autant de calculs que l'on veut sur la matrice B avant de connaître le vecteur cible $\mathbf{t}$, et qu'au moment où $\mathbf{t}$ est donné l'espace utilisé est polynomial en la taille de la matrice B, alors le problème CVP reste NP-difficile. Ce dernier résultat est dû à Micciancio [127].

Bien entendu, lorsque le facteur d'affaiblissement γ augmente, alors le problème γ-CVP devient plus facile. Dinur, Kindler et Safra [56] ont montré qu'il restait tout de

même NP-difficile pour $\gamma = 2^{O(\frac{\log d}{\log\log d})}$. Goldreich et Goldwasser [68] ont montré que le problème γ-CVP n'était pas NP-difficile pour $\gamma = \frac{\sqrt{d}}{O(\log d)}$, à moins que la hiérarchie polynomiale ne s'effondre. En utilisant l'algorithme d'Ajtai, Kumar et Sivakumar pour SVP [11], on peut résoudre γ-CVP en temps polynomial probabiliste pour $\gamma = 2^{\frac{d\log\log d}{\log d}}$. En utilisant l'algorithme BKZ de Schnorr [150], on peut résoudre γ-CVP en temps polynomial déterministe pour $\gamma = 2^{\frac{d(\log\log d)^2}{\log d}}$.

1.2.3 Volume.

Les invariants décrits jusqu'ici sont difficiles à évaluer précisément lorsque la dimension du réseau considéré augmente. Le volume du réseau est au contraire facile à déterminer : on peut le calculer en temps polynomial à partir d'une base quelconque du réseau.

Définition 9 (Parallélépipède fondamental). *Soient $\mathbf{b}_1, \ldots, \mathbf{b}_d$ des vecteurs linéairement indépendants de $\mathbb{R}^n$. Le parallélépipède fondamental des $\mathbf{b}_i$ est :*

$$\mathcal{P}(\mathbf{b}_1, \ldots, \mathbf{b}_d) = \{\sum_{i=1}^{d} y_i \mathbf{b}_i, (y_i)_i \in [0,1]^d\}.$$

Mathématiquement, on définit le volume d'un réseau L comme le volume de l'espace quotient $E(L)/L$, où $E(L)$ est l'espace vectoriel engendré par les vecteurs de L. Cela signifie que le volume du réseau est le volume géométrique du parallélépipède fondamental de n'importe quelle de ses bases. Cette définition est illustrée à la figure 1.5.

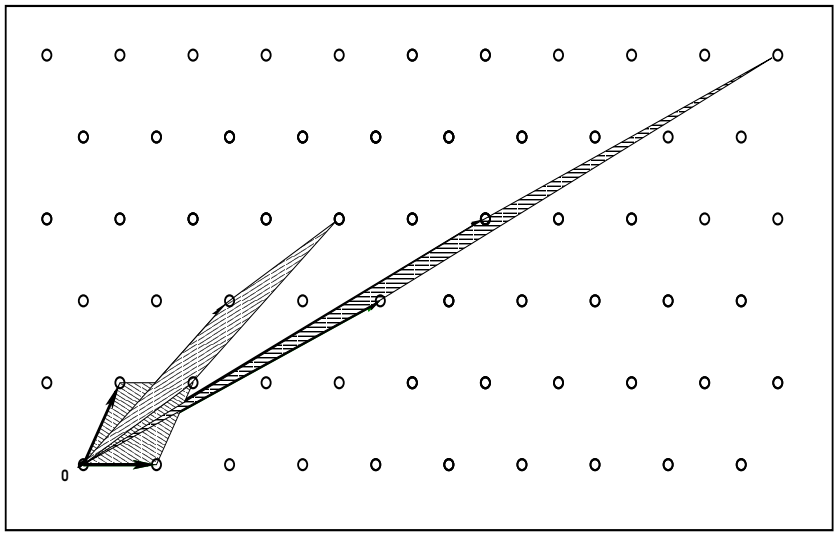


FIG. 1.5 – Le volume d'un réseau est le volume du parallélépipède fondamental de toute base du réseau.

La définition ci-dessous est équivalente.

Définition 10 (Volume). *Soient L un réseau et $\mathbf{b}_1, \ldots, \mathbf{b}_d$ une base de L. Le volume (aussi appelé déterminant) du réseau L est défini par :*

$$\text{vol } L = \prod_{i=1}^{d} \|\mathbf{b}_i^*\|.$$

Cette quantité ne dépend pas du choix de la base $\mathbf{b}_1, \ldots, \mathbf{b}_d$.

D'autres définitions équivalentes sont encore possibles. Nous en donnons certaines dans la proposition suivante.

Proposition 4. *Soit L un réseau de $\mathbb{R}^n$ de dimension d. Soit $\mathbf{b}_1, \ldots, \mathbf{b}_d$ une base de L.*
- *Si $d = n$, alors* $\text{vol } L = |\det(\mathbf{b}_1, \ldots, \mathbf{b}_d)|$.
- *Si $G = (\langle \mathbf{b}_i, \mathbf{b}_j \rangle)_{i,j \leqslant d}$, alors* $\text{vol } L = \sqrt{\det G}$. *La matrice G est appelée la matrice de Gram des vecteurs $\mathbf{b}_1, \ldots, \mathbf{b}_d$.*

1.3 Théorèmes de Minkowski et réductions.

Dans ce chapitre, nous donnons les deux théorèmes de Minkowski, qui permettent de majorer finement les minima d'un réseau donné, puis différentes notions de réduction, qui peuvent être considérées comme des variantes constructives voire effectives des théorèmes de Minkowski (avec des bases à la place de vecteurs linéairement indépendants).

1.3.1 Les deux théorèmes de Minkowski.

Les théorèmes de Minkowski permettent de faire le lien entre un invariant simple à calculer, le volume, et des invariants très difficiles à approcher finement, les minima successifs.

Théorème 1 (Premier théorème de Minkowski). *Soit L un réseau de dimension d. Alors on a :*

$$\lambda_1(L) \leqslant \sqrt{d}(\text{vol } L)^{1/d}.$$

Cette borne est relativement fine « en moyenne », éventuellement au facteur « $\sqrt{d}$ » près. Quand la dimension reste petite, on connaît explicitement la constante d'Hermite $\gamma_d = \max_{L, \dim L = d} \left(\frac{\lambda_1(L)}{(\text{vol } L)^{1/d}} \right)^2 \leqslant d$, mais cette quantité semble extrêmement difficile à calculer, même en dimension 9 [122]. Dans le tableau suivant, nous en donnons les valeurs connues à ce jour, la dernière ayant été calculée très récemment [41].

d	1	2	3	4	5	6	7	8	24
$(\gamma_d)^d$	1	$\frac{4}{3}$	2	4	8	$\frac{64}{3}$	64	2^8	4^{24}

Même si nous ne connaissons pas précisément la constante d'Hermite quand la dimension augmente, nous disposons de bornes asymptotiques assez précises :

$$\frac{d}{2\pi e} + \frac{\log(\pi d)}{2\pi e} + o(1) \leqslant \gamma_d \leqslant \frac{1.744d}{2\pi e}(1 + o(1)).$$

Le premier théorème de Minkowski admet une version plus forte, qui permet de borner le produit des minima.

Théorème 2 (Deuxième théorème de Minkowski). *Soit L un réseau de dimension d. Alors on a :*

$$\prod_{i=1}^{d} \lambda_i(L) \leqslant \gamma_d^d(\text{vol } L) \leqslant d^{d/2}(\text{vol } L).$$

L'un des problèmes algorithmiques centraux concernant les réseaux euclidiens est de rendre effectif ce deuxième théorème avec des bases plutôt que des vecteurs linéairement indépendants, c'est-à-dire de calculer une base dont le produit des longueurs des vecteurs est de l'ordre du volume, à une constante multiplicative près ne dépendant que de la dimension du réseau. Pour mesurer la qualité d'une base, on définit le défaut d'orthogonalité.

Définition 11 (Défaut d'orthogonalité). *Soit $\mathbf{b}_1, \ldots, \mathbf{b}_d$ une base d'un réseau L. On définit le défaut d'orthogonalité de la base $\mathbf{b}_1, \ldots, \mathbf{b}_d$ de la manière suivante :*

$$\delta^{\perp}(\mathbf{b}_1, \ldots, \mathbf{b}_d) = \frac{\prod_{i=1}^{d} \|\mathbf{b}_i\|}{\text{vol } L}.$$

Le défaut d'orthogonalité est toujours supérieur à 1, et vaut 1 si et seulement si la base est orthogonale. Moralement, réduire une base consiste à la transformer en une autre base dont le défaut d'orthogonalité est faible, ou encore en une base dont les longueurs des vecteurs sont proches des minima successifs. Réduire une base revient à la rendre plus orthogonale, et c'est pourquoi certaines réductions tentent d'imiter l'orthogonalisation de Gram-Schmidt (avec la contrainte que les transformations doivent être entières). Il existe de nombreuses notions de réduction, plus ou moins fortes. Nous en listons quelques-unes ci-dessous.

1.3.2 Des réductions fortes, mais difficiles à obtenir.

Nous donnons ici des réductions très fortes, pour lesquelles le premier minimum est atteint. Les calculer est donc au moins NP-difficile sous des réductions randomisées puisque le problème SVP est lui même NP-difficile sous des réductions randomisées. La plus naturelle d'entre elles est peut-être la réduction au sens de Minkowski.

Définition 12 (Réduction au sens de Minkowski). *Une base $\mathbf{b}_1, \ldots, \mathbf{b}_d$ est dite réduite au sens de Minkowski si pour tout $i \leqslant d$, le vecteur $\mathbf{b}_i$ est de norme minimale parmi les vecteurs du réseau engendré tels que les vecteurs $\mathbf{b}_1, \ldots, \mathbf{b}_i$ puissent être complétés pour former une base. De manière équivalente, les inégalités suivantes doivent être satisfaites :*

$$\forall i \leqslant d, \|x_1\mathbf{b}_1 + \ldots + x_d\mathbf{b}_d\| \geqslant \|\mathbf{b}_i\|,$$

pour tous les entiers $x_1, \ldots, x_d$ tels que $x_i, \ldots, x_d$ sont, dans leur globalité, premiers entre eux.

Une base réduite au sens de Minkowski atteint toujours les quatre premiers minima du réseau [184] — la norme du premier vecteur de la base est le premier minimum, la norme du second vecteur de la base est le second minimum, ... — et son défaut d'orthogonalité est borné par $(5/4)^{d^2/2} \cdot d^{d/2}$. La deuxième définition que nous avons donnée pour la réduction au sens de Minkowski fait intervenir une infinité de conditions à vérifier, mais on peut en réalité se contenter d'un nombre fini d'entre elles. L'ensemble minimal de telles conditions forme ce que l'on appelle les conditions de Minkowski. On reviendra sur ce point au chapitre II-2.

La réduction de Hermite-Korkine-Zolotarev n'atteint a priori que le premier minimum du réseau, mais on dispose de meilleures bornes pour son défaut d'orthogonalité. Avant de définir la réduction de Hermite-Korkine-Zolotarev, nous avons besoin de définir ce qu'est une famille propre de vecteurs.

Définition 13 (Propreté). *Des vecteurs $\mathbf{b}_1, \ldots, \mathbf{b}_d$ linéairement indépendants forment une famille propre de vecteurs si pour tous $i > j$, on a $|\mu_{i,j}| \leqslant 1/2$, où les $\mu_{i,j}$ sont les coefficients de l'orthogonalisation de Gram-Schmidt.*

Il est possible de rendre propre une famille de vecteurs en temps polynomial en la taille de la donnée (voir par exemple la description de l'algorithme LLL au chapitre II-3, page 103). Cela permet de diminuer les longueurs des vecteurs, mais ne rend pas la base plus orthogonale. Il ne s'agit donc pas d'une notion de réduction, mais plutôt d'une propriété simple à satisfaire pour éviter d'avoir des vecteurs inutilement longs.

Définition 14 (HKZ-réduction). *Une base $\mathbf{b}_1, \ldots, \mathbf{b}_d$ d'un réseau L est dite réduite au sens de Hermite-Korkine-Zolotarev (HKZ) si $\|\mathbf{b}_1\| = \lambda_1(L)$, si elle est propre, et si la base $\mathbf{b}'_2, \ldots, \mathbf{b}'_d$ est elle-même HKZ-réduite, où le vecteur $\mathbf{b}'_i$ est la projection du vecteur $\mathbf{b}_i$ orthogonalement au vecteur $\mathbf{b}_1$, pour $i \in [|2, d|]$.*

Lagarias, Lenstra et Schnorr [99] ont montré que les bases HKZ-réduites donnent de très bonnes approximations des minima successifs :

$$\forall i \leqslant d, \sqrt{\frac{4}{i+3}} \leqslant \frac{\|\mathbf{b}_i\|}{\lambda_i(L)} \leqslant \sqrt{\frac{i+3}{4}}.$$

Les bases réduites au sens de Hermite-Korkine-Zolotarev peuvent être calculées à l'aide des algorithmes déterministes de Kannan [88] et Helfrich [75], et de l'algorithme probabiliste d'Ajtai, Kumar et Sivakumar [11].

La réduction au sens de Hermite [77], souvent confondue à tort avec la réduction HKZ, est une variante plus exigeante de la réduction au sens de Minkowski : une base Hermite-réduite est une des bases minimales pour l'ordre lexicographique sur les longueurs parmi les bases Minkowski-réduites. On pourrait nommer cette réduction la réduction de Hermite lexicographique. La réduction HKZ diffère des deux autres dès la dimension 3, alors que les réductions aux sens de Minkowski et de Hermite sont identiques jusqu'en dimension 6 incluse, et diffèrent ensuite [149]. Au chapitre II-2, nous analysons un algorithme très naturel qui calcule de telles bases jusqu'en dimension 4 incluse.

1.3.3 Des réductions plus faibles, mais calculables efficacement.

Depuis 1982, la notion de réduction la plus populaire est celle de Lenstra, Lenstra et Lovász [112], appelée LLL-réduction, ou encore L^3-réduction. La raison de cette popularité est que cette réduction peut être calculée en temps polynomial (par exemple avec l'algorithme de Lenstra, Lenstra et Lovász [112]), et qu'elle permet d'obtenir une base de qualité raisonnable, bien que moins bonne que celles réduites au sens de l'une des réductions décrites au paragraphe précédent. Nous proposons au chapitre II-3 un algorithme très efficace pour calculer des bases LLL-réduites.

Définition 15 (LLL-réduction). *Soit $\delta \in]1/4, 1[$. Une base $\mathbf{b}_1, \ldots, \mathbf{b}_d$ est dite LLL-réduite pour le facteur δ si elle est propre et si elle satisfait les $d-1$ conditions de Lovász :*

$$\forall i \in [|2, d|], \|\mathbf{b}_i^* + \mu_{i,i-1}\mathbf{b}_{i-1}^*\| \geqslant \delta \|\mathbf{b}_{i-1}^*\|.$$

Une base LLL-réduite satisfait en particulier les propriétés suivantes.

Théorème 3. *Soient $\delta \in]1/4, 1[$ et $\mathbf{b}_1, \ldots, \mathbf{b}_d$ une base LLL-réduite pour le facteur δ. Soit L le réseau engendré par les vecteurs $\mathbf{b}_1, \ldots, \mathbf{b}_d$.*

- *Pour tout $i \in [|1, d|]$, $\|\mathbf{b}_i\| \leqslant \left(\frac{1}{\delta - 1/4}\right)^{\frac{d-1}{2}} \lambda_i(L)$.*
- $\|\mathbf{b}_1\| \leqslant \left(\frac{1}{\delta - 1/4}\right)^{\frac{d-1}{4}} (\text{vol } L)^{\frac{1}{d}}$.
- $\prod_{i=1}^{d} \|\mathbf{b}_i\| \leqslant \left(\frac{1}{\delta - 1/4}\right)^{\frac{d(d-1)}{4}} (\text{vol } L)$.
- *Pour tous $1 \leqslant i \leqslant j \leqslant d$, $\|\mathbf{b}_j^*\| \geqslant \left(\frac{1}{\delta - 1/4}\right)^{j-i} \|\mathbf{b}_i^*\|$.*

La valeur $1/4$ est exclue pour le paramètre δ pour une raison évidente. Par contre, on pourrait choisir $\delta = 1$, ce qui correspond à la LLL-réduction optimale, mais on ne sait pas calculer de telles bases en temps polynomial. Les meilleures analyses de l'algorithme LLL optimal (c'est-à-dire avec $\delta = 1$) sont dues à Akhavi [14] et à Lenstra [115].

De façon étonnante, la LLL-réduction ressemble à une notion de réduction décrite par Hermite [76] dans le langage des formes quadratiques : on commence par réduire récursivement les vecteurs $\mathbf{b}'_2, \ldots, \mathbf{b}'_d$ (les projetés des vecteurs $\mathbf{b}_2, \ldots, \mathbf{b}_d$ orthogonalement au vecteur $\mathbf{b}_1$), on les « proprifie » par rapport au vecteur $\mathbf{b}_1$, et on regarde si $\|\mathbf{b}_2\| < \|\mathbf{b}_1\|$. Si c'est le cas, on échange les deux premiers vecteurs et on recommence le procédé, sinon la base est réduite. Cela a permis à Hermite de montrer que tout réseau admet une base dont le défaut d'orthogonalité peut être borné par une constante ne dépendant que de la dimension. La constante en question est $\left(\frac{4}{3}\right)^{\frac{d(d-1)}{4}}$, comme dans le cas de l'algorithme LLL optimal.

Schnorr a décrit dans [150] une hiérarchie de réductions allant de la LLL-réduction à la réduction au sens de Hermite-Korkine-Zolotarev, ainsi qu'une hiérarchie d'algorithmes permettant à peu près de les atteindre. Ces algorithmes font appel à des algorithmes pour résoudre exactement le problème SVP, comme les algorithmes de Kannan [88], Helfrich [75] et Ajtai, Kumar et Sivakumar [11], qui sont utilisés sur des réseaux auxiliaires

de dimension $\beta \leqslant d$, où β est appelé la taille de bloc et est pré-spécifié. Quand $\beta = 2$, on obtient la LLL-réduction, et quand $\beta = d$, on obtient la réduction HKZ.

Définition 16 (BKZ-réduction). *Soit $\beta \in [|2, d|]$. Une base $\mathbf{b}_1, \ldots, \mathbf{b}_d$ est dite réduite au sens de Hermite-Korkine-Zolotarev par blocs de taille β (β-BKZ-réduite) si elle est propre, et si pour tout $i \in [|\beta, d|]$, les projetés des vecteurs $\mathbf{b}_{i-\beta+1}, \ldots, \mathbf{b}_i$ orthogonalement aux vecteurs $\mathbf{b}_1, \ldots, \mathbf{b}_{i-\beta}$ sont réduits au sens de Hermite-Korkine-Zolotarev.*

Une base β-BKZ-réduite contient un vecteur de longueur inférieure à $\beta^{O\left(\frac{1}{\beta}\right)}\lambda_1$, où λ_1 est le premier minimum du réseau engendré par les vecteurs $\mathbf{b}_1, \ldots, \mathbf{b}_d$. Schnorr [150] a montré comment obtenir des bases quasiment β-BKZ-réduites (de qualité essentiellement identique) en temps déterministe $\beta^{O(\beta)}$, à des facteurs polynomiaux près en la taille de la donnée, en utilisant les algorithmes de Kannan et Helfrich. Il peut être modifié pour obtenir une borne de complexité probabiliste $2^{O(\beta)}$, à des facteurs polynomiaux près, en utilisant l'algorithme d'Ajtai, Kumar et Sivakumar. Si l'on se restreint aux algorithmes de complexité polynomiale, on obtient des tailles de blocs maximales $O\left(\frac{\log d}{\log\log d}\right)$ pour le cas déterministe et $O(\log d)$ pour le cas probabiliste. On peut donc résoudre les problèmes γ-SVP en temps polynomial déterministe avec $\gamma = 2^{O\left(\frac{\log d}{(\log\log d)^2}\right)}$ et en temps polynomial probabiliste avec $\gamma = 2^{O\left(\frac{\log d}{\log\log d}\right)}$.

Chapitre I-2

Éléments d'arithmétique flottante.

W. Kahan, 1997 — *Twenty years ago, anarchy threatened floating-point arithmetic. Over a dozen commercially significant arithmetics boasted diverse wordsizes, precisions, rounding procedures and over/underflow behaviors, and more were in the works. "Portable" software intended to reconcile that numerical diversity had become unbearably costly to develop.*

Sommaire

2.1 Nombres flottants et norme IEEE-754. 18
2.1.1 Nombres flottants. 18
2.1.2 Opérations arithmétiques et modes d'arrondi. 19
2.2 Deux extensions de la norme IEEE-754. 19
2.2.1 Normalisation des fonctions élémentaires. 19
2.2.2 Calculs en multiprécision. 20
2.3 Sommation de nombres flottants. 21

Jusqu'au début des années 1980, les calculs scientifiques étaient effectués avec des arithmétiques flottantes non normalisées, et dont les sémantiques variaient fortement d'une machine à l'autre, d'un langage à l'autre, voire même d'un compilateur à l'autre. Sous la pression d'industriels, et de scientifiques, dont en particulier William Kahan, la norme IEEE-754 [2] a vu le jour en 1985 et a permis d'uniformiser de manière considérable tous ces calculs, en spécifiant de manière très précise comment représenter des nombres flottants et comment effectuer les principales opérations arithmétiques. Cette norme est devenue très vite la référence, autant dans le domaine matériel, pour lequel elle a été rédigée à l'origine, que dans le domaine logiciel. Son succès trouve probablement sa raison dans le fait qu'elle apporte les solutions les plus rationnelles à un certain nombre de problèmes intrinsèques aux nombres flottants.

Dans la section 2.1, nous rappelons assez brièvement quelques éléments essentiels de la norme IEEE-754. Cette norme sert de référence pour diverses généralisations, comme les calculs en multiprécision et les calculs liés aux fonctions mathématiques élémentaires.

précision	« statut »	taille de la représentation (en bits)	$e_{\min}$	$e_{\max}$	taille de la mantisse (en bits)
simple	existe	32	-126	+127	24
simple étendue	déprécié	43-44	-1022	+1023	32
double	populaire	64	-1022	+1023	53
double étendue	existe	79-80	-16382	+16382	64
quadruple	théorique	128	-16382	+16382	113

FIG. 2.1 – Nombres flottants dans les cinq précisions classiques.

Nous présentons ces extensions à la section 2.2. Enfin, à la section 2.3, nous donnons un résultat simple sur la sommation de flottants, que nous utiliserons au chapitre II-3.

2.1 Nombres flottants et norme IEEE-754.

Nous donnons ici quelques définitions de base sur les nombres flottants de la norme IEEE-754. Pour de plus amples détails, nous renvoyons le lecteur à [66, 134].

2.1.1 Nombres flottants.

Un nombre flottant est un triplet (s, e, m) où $s = \pm 1$ est appelé le signe, $e \in \mathbb{Z}$ est appelé l'exposant et $m \in [1, 2[$ est appelée la mantisse et n'a qu'un nombre fini de bits après la virgule. Le nombre réel naturellement associé au triplet (s, e, m) est $s \cdot m \cdot 2^e$. Le signe est toujours représenté sur 1 bit, mais les nombres de bits utilisés pour la mantisse et pour l'exposant dépendent de la précision choisie. La norme IEEE-754 prévoit deux principales précisions possibles, la simple précision et la double précision, dont les caractéristiques sont précisées dans le tableau de la figure 2.1. La quadruple précision est une extension naturelle de ces deux précisions avec une taille de mantisse plus grande. Pour ces trois précisions, le premier bit de la mantisse n'est pas stocké explicitement, car il vaut toujours 1. Ainsi, un nombre flottant en simple précision est stocké sur 32 bits, un nombre flottant en double précision sur 64 bits, et un nombre flottant en quadruple précision est stocké sur 128 bits. Aujourd'hui, la précision la plus populaire est la double précision.

La norme IEEE-754 prévoit aussi deux précisions optionnelles : la simple précision étendue et la double précision étendue. Une différence importante dans ces cas est que le premier bit n'est plus forcément implicite (cela n'est pas spécifié). La simple précision étendue est aujourd'hui extrêmement rarement utilisée, et la double précision étendue est assez fréquemment disponible, comme par exemple sur les architectures X86.

On remarque que les exposants possibles ne correspondent pas exactement à ceux qui auraient pu l'être si l'on considère l'espace alloué. Cela permet de représenter les valeurs spéciales (± 0, $\pm\infty$, NaN), et éventuellement les nombres dénormalisés (des nombres que l'on peut rajouter entre 0 et le plus petit nombre représentable autrement).

2.1.2 Opérations arithmétiques et modes d'arrondi.

La norme IEEE-754 régit les opérations arithmétiques élémentaires sur les nombres flottants. Quelle que soit la précision choisie, la valeur renvoyée pour l'opération a op b pour op $\in \{+, -, \times, /\}$ doit être l'arrondi de la « vraie » valeur de a op b. Cette règle nécessite plusieurs explications. Tout d'abord, plusieurs modes d'arrondi sont possibles. On distingue les arrondis dirigés des arrondis non dirigés. Le seul mode d'arrondi pour les arrondis non-dirigés est l'arrondi au nombre représentable le plus proche, celui avec une mantisse paire s'il y a plusieurs choix possibles. Cet arrondi sera ici noté $\diamond(\cdot)$, et c'est celui que nous utiliserons par défaut dans cette thèse. Il y a par ailleurs trois modes d'arrondi dirigés : l'arrondi vers $+\infty$ revient à choisir le nombre représentable supérieur ou égal à la vraie valeur qui est le plus proche ; l'arrondi vers $-\infty$ revient à choisir le nombre représentable inférieur ou égal à la vraie valeur qui est le plus proche ; et l'arrondi vers 0 correspond à l'arrondi vers $-\infty$ pour un nombre positif et à l'arrondi vers $+\infty$ pour un nombre négatif.

La norme IEEE-754 régit de la même façon le comportement de la racine carrée : la valeur renvoyée pour $\sqrt{x}$ doit être l'arrondi de $\sqrt{x}$, pour le mode choisi.

2.2 Deux extensions de la norme IEEE-754.

Nous considérons ici deux extensions de la norme IEEE-754 : dans un premier temps son extension à d'autres fonctions mathématiques que les quatre opérations arithmétiques de base et la racine carrée, puis nous considérons les calculs en multiprécision.

2.2.1 Normalisation des fonctions élémentaires.

Un certain nombre de chercheurs [52] souhaiteraient voir apparaître une extension de la norme IEEE-754 à d'autres fonctions mathématiques que la racine carrée. Le but serait que, pour les simple, double et quadruple précisions et les quatre modes d'arrondi, la valeur renvoyée pour $f(x)$ soit l'arrondi de $f(x)$, et ce pour un certain nombre de fonctions f, éventuellement en se restreignant à des sous-domaines du domaine de définition. Une première question est de choisir ces fonctions. Parmi celles-ci devraient très certainement se trouver les fonctions mathématiques élémentaires comme les fonctions exponentielles $\exp(x)$ et 2^x, les fonctions logarithmiques $\log(x)$ et $\log_{10}(x)$, les fonctions trigonométriques $\sin(x), \cos(x), \tan(x)$, $\operatorname{cotan}(x)$, leurs inverses, les fonctions trigonométriques hyperboliques $\sinh(x), \cosh(x)$, $\tanh(x)$, $\operatorname{cotanh}(x)$ et leurs inverses. À celles-ci pourraient éventuellement se rajouter des fonctions comme $\sin(\pi x)$ et $\cos(\pi x)$, des fonctions mathématiques spéciales comme la fonction Γ d'Euler, la fonction ζ de Riemann, ou la fonction erf, ou encore des fonctions de plusieurs variables comme $\sqrt{x^2+y^2}$, $\frac{1}{\sqrt{x^2+y^2}}$ et $\tan\left(\frac{x}{y}\right)$.

La normalisation de ces fonctions est particulièrement étudiée aujourd'hui, en partie à cause de la révision de la norme IEEE-754. Normaliser ces fonctions aurait des aspects positifs indéniables. Cela permettrait de garantir de meilleures portabilités et reproductibilités des calculs scientifiques. Une définition claire de la sémantique de ces fonctions

aiderait aussi à prouver et garantir (formellement) des programmes, comme cela commence à être le cas avec des opérations simples (voir par exemple la thèse de Sylvie Boldo [25]). Les réfractaires à l'extension de la norme IEEE-754 aux fonctions élémentaires avancent l'argument qu'on ne sait pas encore obtenir l'arrondi correct à un faible coût (par exemple pour les fonctions trigonométriques prises en de grandes valeurs, ces fonctions devenant numériquement très instables), et que dans les cas où on sait le faire, le calcul prend beaucoup plus de temps pour certaines entrées que pour d'autres. Une des raisons à cela est la difficulté à résoudre le dilemme du fabricant de tables [107] dans le cas des fonctions élémentaires (pour les quatre opérations arithmétiques de base et pour la racine carrée, il admet une réponse simple). Par exemple pour l'arrondi au plus proche, la valeur exacte de $f(x)$ peut être particulièrement proche du milieu de deux nombres représentables, auquel cas il faut calculer $f(x)$ avec une précision beaucoup plus élevée pour arriver à décider l'arrondi : pour résoudre le dilemme du fabricant de table, il faudrait trouver des bornes a priori sur la précision intermédiaire requise, pour une précision donnée. On propose un algorithme pour ce type de questions au chapitre III-1. Pour illustrer les réticences du comité de révision de la norme IEEE-754, on peut par exemple citer David Hough[3] :

> *[Elementary functions] are too hard to standardize now because nobody knows how to trade-off precision vs performance. If less than correct rounding is specified [...] then [it] is always possible that somebody will come up with a non-standard algorithm that is more accurate AND faster. This can't happen with a correct rounding specification, but correctly-rounded transcendental functions seem to be inherently much more expensive than almost-correctly-rounded. One could instead standardize properties that approximate functions are supposed to obey - but anomalies still abound. All these points argue against standardizing transcendental functions now.*

Cependant, au cours du congrès ARITH'17 à Cape Cod, les partisans de la normalisation ont apporté des preuves de faisabilité convaincantes, en particulier avec la bibliothèque Crlibm développée dans le projet ARÉNAIRE de l'INRIA. La normalisation des fonctions élémentaires a de nouveau été considérée par le comité de révision de la norme :

> *I [David Hough] was inspired by the presentations at Arith-17 to consider a modest beginning of correctly-rounded transcendental functions as an addition to IEEE 754R. Anybody wishing to take a look and comment may find an outline at* [`http://754r.ucbtest.org/ballots/passed/trans.htm`]

2.2.2 Calculs en multiprécision.

Une autre généralisation de la norme IEEE-754 est l'extension des règles à toutes les précisions possibles, en levant les contraintes sur les tailles des mantisses et des exposants. Les opérations deviennent plus complexes : il faut spécifier le mode d'arrondi et les tailles des mantisses des opérandes et des résultats. Quand nous utiliserons la multiprécision (en particulier au chapitre II-3), nous prendrons le modèle classique suivant.

[3] voir l'URL `http://grouper.ieee.org/groups/754/meeting-minutes/01-03-14.html`

En précision n, un nombre flottant est un triplet (s, e, m) avec $s = \pm 1$, e un entier relatif stocké sur $\lceil \lg n \rceil$ bits (le domaine est centré autour de 0), et $m \in [1, 2[$ a au plus n bits après la virgule. Pour un mode d'arrondi choisi, on exigera que le résultat renvoyé pour a op b soit l'arrondi du résultat exact, pour op $\in \{+, -, \times, /\}$. On ne spécifie pas comment doivent être gérées les erreurs (*overflow, underflow, ...*) car nous prendrons une précision n suffisamment élevée pour que la question ne se pose pas.

Parmi les implantations pratiques d'un tel modèle, on peut mentionner par exemple la bibliothèque MPFR [147], qui étend en plus ces règles concernant les arrondis à de nombreuses fonctions élémentaires et spéciales, et la classe RR de la bibliothèque NTL [162] (pour ces deux bibliothèques la plage d'exposant est bornée, mais cette borne est très grande).

2.3 Sommation de nombres flottants.

Nous donnons ici un résultat élémentaire sur la sommation de flottants, que l'on peut trouver dans [78], et dont nous nous servirons au chapitre II-3.

Théorème 4. *Soient ℓ la précision de calcul, et $x_1, \ldots, x_d$ des flottants de même précision ℓ. S'il ne survient pas de dépassement de capacité dans les calculs, on a :*

$$\left| \diamond(x_d + \diamond(x_{d-1} + \ldots + \diamond(x_2 + x_1)\ldots)) - \sum_{i=1}^{d} x_i \right| \leqslant \frac{d2^{-\ell-1}}{1 - d2^{-\ell-1}} \cdot \sum_{i=1}^{d} |x_i|.$$

Ceci n'est pas la méthode de sommation de flottants qui minimise l'erreur (on peut par exemple sommer « en arbre », ou utiliser l'algorithme de Demmel et Hida [54], dont une preuve simplifiée est donnée dans [61]), mais elle est simple et suffira à nos besoins.

Deuxième partie

Algorithmique de la réduction des réseaux.

Chapitre II-1

L'algorithme rapide de pgcd binaire.

Le travail de recherche correspondant à ce chapitre a été initié à l'université de Caen sous la direction et avec les conseils de Brigitte Vallée. Il a fait l'objet d'une publication dans les actes de la conférence ANTS VI (Sixth Algorithmic Number Theory Symposium) [169]. Le présent chapitre est une version étendue de [169].

D. Knuth, 1970 — *Is Euclid's algorithm the "best" way to calculate greatest common divisors ?*

Sommaire

1.1 L'algorithme de Knuth-Schönhage. 26
1.2 La division binaire généralisée. 27
1.2.1 La division binaire généralisée. 29
1.2.2 L'algorithme d'Euclide binaire généralisé. 32
1.2.3 Sur la taille réelle des restes binaires généralisés. 34
1.2.4 Calcul d'inverses modulaires. 37
1.3 L'algorithme binaire rapide. 37
1.4 Correction et analyse de complexité de l'algorithme binaire rapide. 40
1.4.1 Correction de l'algorithme **Pgcd-BG-rapide**. 40
1.4.2 Analyse de complexité de l'algorithme **Pgcd-BG-rapide**. 42
1.5 Remarques sur l'implantation des algorithmes rapides de calcul de pgcd. 43

Ce premier chapitre est consacré aux calculs asymptotiquement rapides de plus grands communs diviseurs (pgcd). Calculer des pgcds est une tâche centrale en arithmétique, en particulier quand on veut calculer sur $\mathbb{Q}$, ou en arithmétique modulaire. L'algorithme d'Euclide permet d'effectuer cela en temps quadratique en la taille n des entrées. Il a été étudié et analysé dans le détail dans les dernières décennies. En particulier une analyse en moyenne extrêmement précise a été effectuée par Vallée [182] pour une très grande classe de variantes. Le premier algorithme de pgcd de complexité quasi-linéaire a été

décrit par Knuth au Congrès International des Mathématiciens de 1970 [91]. Il obtient alors la borne $O(n \log^5 n \log\log n)$. Cette complexité a été améliorée peu de temps après par Schönhage [155], la borne décroît en $O(n \log^2 n \log\log n)$, ce qui reste aujourd'hui la meilleure borne asymptotique. Nous commençons ce chapitre par une description rapide de l'algorithme de Knuth-Schönhage, qui nous permettra de voir quels sont les inconvénients dont souffre cet algorithme. En particulier, la correction est délicate à établir à cause de procédure de réparation laborieuse. Par exemple, la preuve de [190] est fausse, et des erreurs sont répertoriées à l'URL : `http://www.cs.nyu.edu/cs/faculty/yap/book/errata.html`. Cette procédure de réparation est difficile à implanter et a longtemps été un frein à la mise en place d'un pgcd rapide dans les bibliothèques de calcul sur les entiers comme GNU MP [73]. Nous décrivons dans ce chapitre un algorithme rapide de calcul de pgcd alternatif qui ne nécessite pas d'une procédure de réparation. Pour cela nous introduisons dans un premier temps une nouvelle division, la division binaire généralisée (DBG). Cela nous permettra de décrire ensuite l'algorithme quasi-linéaire qui en découle. Nous prouverons alors la correction et donnerons une analyse de complexité de cet algorithme, avant de nous intéresser à des problèmes plus liés à l'implantation.

1.1 L'algorithme de Knuth-Schönhage.

L'algorithme de Knuth-Schönhage simule l'algorithme d'Euclide de calcul de pgcd. Nous commençons donc par brièvement rappeler ce dernier.

Définition 17 (L'algorithme d'Euclide). *Soient $a > b \geqslant 0$. L'algorithme d'Euclide calcule le* pgcd *de a et b de la manière suivante. Une suite de restes $r_0 = a, r_1 = b, r_2, \ldots$ et une suite de quotients $q_1, q_2, \ldots$ sont calculées progressivement : pour $i \geqslant 1$,*

$$q_i = \lfloor r_{i-1}/r_i \rfloor \quad \text{et } r_{i+1} = r_{i-1} - q_i r_i.$$

L'algorithme s'arrête quand un reste nul $r_{\tau+1}$ est calculé. Alors le pgcd *de a et b est r_τ.*

L'algorithme de Knuth-Schönhage a été décrit de nombreuses fois [39, 91, 116, 132, 144, 155, 178, 190] et a fait l'objet de généralisations aux polynômes (voir [5, 31, 64] par exemple), aux entiers de Gauss [185] et aux réseaux Euclidiens planaires [157, 189]. Nous nous reposerons sur la description de [190], dont la preuve associée est erronée comme nous l'avons dit plus haut. Des preuves correctes et complètes peuvent être trouvées dans [157] et [132] (qui utilise la stratégie décrite dans [157]) : dans ces deux références, l'algorithme est modifié pour limiter au plus la procédure de réparation.

L'algorithme de Knuth-Schönhage est une variante récursive de l'algorithme de Lehmer [111]. L'idée de l'algorithme de Lehmer est de calculer le début de la suite des quotients en n'utilisant qu'un ou deux mots-machine des entrées, et d'appliquer en une fois plusieurs quotients ainsi obtenus aux restes courants. La difficulté est de savoir jusqu'à quand les quotients ainsi calculés sont corrects, et en particulier d'utiliser correctement le critère de Jebelean [84]. L'algorithme de Knuth-Schönhage simule donc l'exécution de l'algorithme d'Euclide en calculant tous les quotients qui auraient été calculés, mais très peu de restes. En effet, rien qu'écrire la suite des restes coûterait dans le cas le pire $O(n^2)$ opérations élémentaires où n est la taille des nombres initiaux.

L'algorithme est composé de deux routines : l'algorithme **Demi-pgcd**, décrit à la figure 1.1 et illustré à la figure 1.3, puis l'algorithme **Pgcd-rapide**, décrit à la figure 1.2. L'algorithme **Pgcd-rapide** prend en entrée deux entiers a et b, et renvoie leur pgcd en faisant des appels successifs à l'algorithme **Demi-pgcd**. De son côté, l'algorithme **Demi-pgcd** prend en entrée deux entiers $a \geqslant b$ et renvoie une matrice unimodulaire[4] R telle que si $\begin{pmatrix} c \\ d \end{pmatrix} = R \cdot \begin{pmatrix} a \\ b \end{pmatrix}$, alors c et d sont les deux restes de la suite des restes de (a, b) qui aurait été générée par l'algorithme d'Euclide qui satisfont :

$$\lg c \geqslant 1 + \left\lceil \frac{\lg a}{2} \right\rceil > \lg d.$$

En fait, R est égale au produit matriciel $[q_i] \ldots [q_1]$ où les q_i sont les premiers quotients de la suite des quotients de a par b et $[q] = \begin{pmatrix} 0 & 1 \\ 1 & -q \end{pmatrix}$. L'algorithme **Demi-pgcd** utilise deux appels récursifs de taille moitié, une division élémentaire, et deux appels à une procédure de réparation. Si $H(n)$ et $G(n)$ sont les coûts maximaux respectifs des algorithmes **Demi-pgcd** et **Pgcd-rapide** pour des couples d'entiers de tailles au plus n, alors on a approximativement les équations :

$$\begin{aligned} H(n) &= 2H(n/2) + O(M(n)), \\ G(n) &= H(n) + G(n/2) + O(M(n)). \end{aligned}$$

La procédure de réparation, sur laquelle repose la correction de l'algorithme, est extrêmement laborieuse à décrire car elle fait intervenir de nombreux cas particuliers. Intuitivement, la matrice calculée R peut correspondre à trop ou pas assez de quotients, et s'il y en a le bon nombre, le dernier quotient peut aussi être faux. Cette procédure peut faire appel à $O(1)$ divisions euclidiennes pour avancer ou reculer dans la suite des quotients et/ou $O(1)$ multiplications. Une procédure de réparation plus simple que celle de [190] peut être trouvée dans [116]. Dans [157] et [132], le problème est contourné en modifiant l'algorithme **Demi-pgcd** en lui demandant de renvoyer des restes et une matrice qui « sont allés moins loin » dans l'algorithme d'Euclide. Grâce à cela, on évite d'avoir à revenir en arrière dans la suite des quotients, mais la matrice renvoyée peut elle aussi nécessiter une réparation constituant à avancer dans la suite des quotients. Cela simplifie déjà nettement l'algorithme.

1.2 La division binaire généralisée.

Pour éviter la procédure de réparation dans l'algorithme de Knuth-Schönhage, nous changeons de division : au lieu d'utiliser la division euclidienne classique, nous utilisons une généralisation de la division binaire [171], qui peut être interprétée comme une division sur les nombres 2-adiques. Utiliser la division binaire classique ne semble pas possible parce que celle-ci élimine les bits de poids faible, mais pour cela elle utilise les bits de poids fort. Cette asymétrie n'apparaît plus dans la division binaire généralisée : elle ne considère que

[4] C'est-à-dire à coefficients entiers et de déterminant ± 1.

Algorithme: **Demi-pgcd**.
Entrée : Deux entiers a et b tels que $a \geqslant b > 0$.
Sortie : Une matrice unimodulaire $R \in \mathcal{M}_{2,2}(\mathbb{Z})$ telle que si $\begin{pmatrix} c \\ d \end{pmatrix} = R \cdot \begin{pmatrix} a \\ b \end{pmatrix}$, alors $\lg c \geqslant 1 + \left\lceil \frac{\lg a}{2} \right\rceil > \lg d$.

1. m:=$1 + \left\lceil \frac{\lg a}{2} \right\rceil$.
2. Si $\lg b < m$, renvoyer $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$.
3. a_0:=$1 + \lfloor a \cdot 2^{-m} \rfloor$, b_0:=$\lfloor b \cdot 2^{-m} \rfloor$, t:=$1 + \left\lceil \frac{\lg a_0}{2} \right\rceil$.
4. R_1:=**Réparation**(**Demi-pgcd**$(a_0, b_0), a, b, m, t)$.
5. $\begin{pmatrix} a' \\ b' \end{pmatrix} = R_1 \cdot \begin{pmatrix} a \\ b \end{pmatrix}$.
6. q:=$\left\lfloor \frac{a'}{b'} \right\rfloor$, a'':=b', b'':=$a' - qb'$.
7. Si $\lg b'' < m$, renvoyer $\begin{pmatrix} 0 & 1 \\ 1 & -q \end{pmatrix} \cdot R_1$.
8. l:=$\lceil \lg a'' \rceil$, k:=$2m - l - 1$.
9. a''_0:=$1 + \lfloor a'' \cdot 2^{-k} \rfloor$, b''_0:=$\lfloor b'' \cdot 2^{-k} \rfloor$, t'':=$1 + \left\lceil \frac{\lg a''_0}{2} \right\rceil$.
10. R_2:=**Réparation**(**Demi-pgcd**$(a''_0, b''_0), a'', b'', k, t'')$.
11. Renvoyer $R_2 \cdot \begin{pmatrix} 0 & 1 \\ 1 & -q \end{pmatrix} \cdot R_1$.

FIG. 1.1 – L'algorithme **Demi-pgcd**.

Algorithme: **Pgcd-rapide**.
Entrée : Deux entiers a et b tels que $a \geqslant b > 0$.
Sortie : $g = \text{pgcd}(a, b)$.

1. R:=**Demi-pgcd**(a, b).
2. $\begin{pmatrix} c \\ d \end{pmatrix} = R \cdot \begin{pmatrix} a \\ b \end{pmatrix}$.
3. Si $d = 0$, renvoyer c.
4. Renvoyer **Pgcd-rapide**$(d, c \mod d)$.

FIG. 1.2 – L'algorithme **Pgcd-rapide**.

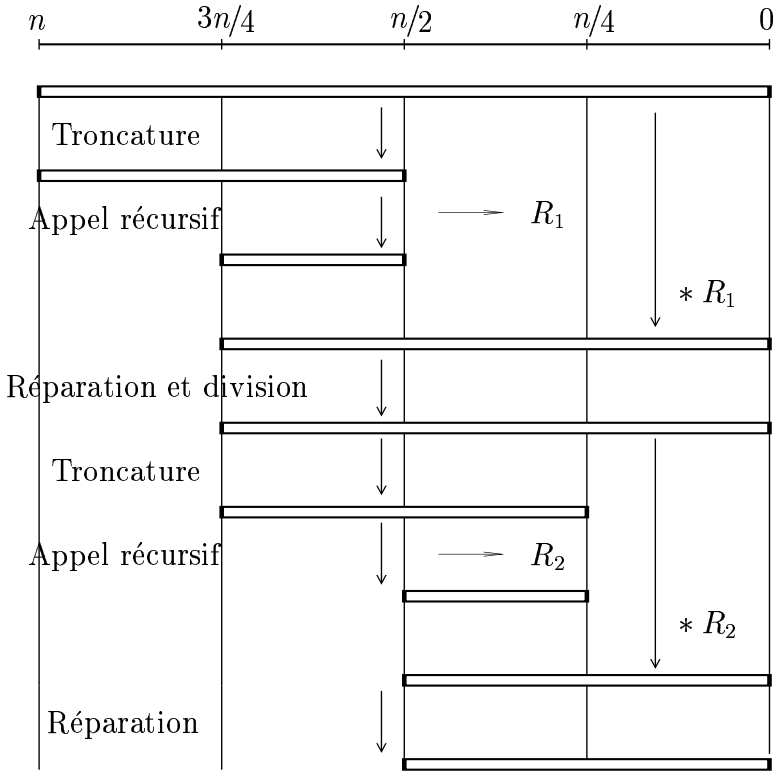


FIG. 1.3 – Représentation graphique de l'algorithme **Demi-pgcd**.

les bits de poids faible. La procédure de réparation de l'algorithme de Knuth-Schönhage provient du fait que les retenues se propagent dans le sens inverse de la diminution de la taille des nombres alors que, dans notre variante, les retenues et la diminution des tailles vont dans le même sens, ce qui supprime la procédure de réparation. Cette stratégie a le défaut de faire intervenir des nombres un peu plus grands : dans l'algorithme d'Euclide classique, la matrice de transformation correspondant à un gain de k bits de poids fort dans la suite des restes (c'est-à-dire les restes courants ont k bits de moins que le plus grand reste initial) a des entrées d'environ k bits ; dans l'algorithme d'Euclide binaire généralisé, la matrice de transformation correspondant à un gain de k bits de poids faible aura des entrées rationnelles dont les numérateurs pourront avoir $\approx 1.36 \cdot k$ bits dans le cas le pire (voir le lemme 5, page 34). En « pratique » il semble que la quantité $\approx 1.36 \cdot k$ soit à remplacer par $\approx (1+\varepsilon) \cdot k$, pour une petite constante $\varepsilon > 0$. Nous revenons sur ce dernier point à la fin du paragraphe 1.2.3, à la page 36.

1.2.1 La division binaire généralisée.

L'algorithme de division binaire repose sur les propriétés suivantes : $\mathrm{pgcd}(2a, 2b) = 2\mathrm{pgcd}(a, b)$, $\mathrm{pgcd}(2a+1, 2b) = \mathrm{pgcd}(2a+1, b)$, et $\mathrm{pgcd}(2a+1, 2b+1) = \mathrm{pgcd}(2a+1, a-b)$. Le principe est d'éliminer un bit de poids faible à chaque itération de boucle. L'algorithme binaire est explicité dans la figure 1.4. Le comportement de cet algorithme est très bien compris, voir [30] par exemple. Asymptotiquement, il a la même complexité

que l'algorithme d'Euclide naïf, c'est-à-dire il a une complexité quadratique en la taille des nombres donnés en entrée.

Algorithme: **Pgcd-binaire**.
Entrée : $a, b \in \mathbb{Z}$.
Sortie : $\mathrm{pgcd}(a, b)$.

1. Si $|b| > |a|$, renvoyer **Pgcd-binaire**(b, a).
2. Si $b = 0$, renvoyer a.
3. Si a et b sont tous les deux pairs, renvoyer $2 \cdot$ **Pgcd-binaire**$(a/2, b/2)$.
4. Si a est pair et b est impair, renvoyer **Pgcd-binaire**$(a/2, b)$.
5. Si a est impair et b est pair, renvoyer **Pgcd-binaire**$(a, b/2)$.
6. Sinon renvoyer **Pgcd-binaire**$((|a| - |b|)/2, b)$.

FIG. 1.4 – L'algorithme d'Euclide binaire classique.

Dans le cas de la division Euclidienne classique de a par b avec $|a| > |b|$, on calcule un quotient q tel que lorsque qb est soustrait à a, le reste obtenu est plus court que b. Intuitivement, pour calculer un quotient, des décalés « gauches » de b sont soustraits à a autant que possible, c'est-à-dire quand a a perdu ses $l(a)-l(b)$ bits les plus significatifs. La division binaire généralisée est le symétrique de cela : pour diviser a par b avec $|a|_2 > |b|_2$, où $|a|_2 = 2^{-\nu_2(a)}$ est la norme 2-adique de a, on calcule un quotient $\frac{q}{2^k}$ tel que lorsque $\frac{q}{2^k}b$ est additionné à a, le reste obtenu est plus petit que b pour la norme 2-adique : intuitivement, des décalés droits de b sont soustraits à a autant que possible, c'est-à-dire quand a a perdu ses $\nu_2(b) - \nu_2(a)$ bits les moins significatifs.

Définition 18 (Division binaire généralisée). *Soient a, b des entiers non nuls avec $\nu_2(a) < \nu_2(b)$. Il existe une unique paire d'entiers (q, r) telle que :*

$$\begin{aligned} r &= a + q\frac{b}{2^{\nu_2(b)-\nu_2(a)}}, \\ |q| &< 2^{\nu_2(b)-\nu_2(a)}, \\ \nu_2(r) &> \nu_2(b). \end{aligned}$$

Nous appellerons q et r respectivement le DBG-quotient et le DBG-reste de a par b, et noterons $\mathrm{DBG}(a, b) = (q, r)$.

Démonstration. Des deux premières équations on a directement que :

$$q = -\frac{a}{2^{\nu_2(a)}} \cdot \left(\frac{b}{2^{\nu_2(b)}}\right)^{-1} \bmod 2^{\nu_2(b)-\nu_2(a)+1},$$

où $x \bmod y$ est l'entier z congrus à x modulo y qui appartient à l'intervalle $\left]\frac{-y}{2}, \frac{y}{2}\right]$. Comme q est impair, la deuxième condition est alors vérifiée et donne l'unicité de q. L'unicité et l'existence de r en découlent immédiatement, grâce à la première équation. Finalement, comme $r = a + q\frac{b}{2^{\nu_2(b)-\nu_2(a)}}$, nous avons $r = 0 \mod 2^{\nu_2(b)+1}$, ce qui donne la dernière équation. □

La division binaire généralisée ressemble à la division impaire de Hensel, qui date du début du XX^e siècle, et dont une description peut être trouvée dans [161]. Pour deux entiers a et b avec b impair, on calcule une paire d'entiers (q, r) telle que :

$$a = -bq + 2^p r \quad \text{et} \quad r < 2b,$$

où $p = l(a) - l(b)$ est la différence entre les longueurs de a et b. Dit différemment, la division impaire de Hensel calcule $r := (2^{-p}a) \mod b$, ce qui peut être fait efficacement en utilisant [133] par exemple. Par ailleurs, il y a aussi des similarités entre la division binaire généralisée et la division « Plus-Moins » de Brent et Kung [32]. Quand $\nu_2(a) = 0$ et $\nu_2(b) = 1$, c'est-à-dire quand a est impair et $b = 2b'$ avec b' i mpair, la DBG trouve $q = \pm 1$ tel que $(a + qb')/2$ est pair, ce qui est exactement l'algorithme « Plus-Moins ». Contrairement à la division binaire classique, à la division impaire de Hensel et à la division « Plus-Moins », la DBG ne considère que les bits de poids faible : elle ne requiert pas de comparer les deux nombres ou de calculer la différence de leurs longueurs[5]. Elle est très naturelle quand on considère les entiers comme des nombres 2-adiques.

Il existe une autre division qui ne nécessite que les bits de poids faible, il s'agit de celle de Purdy [148]. Le pgcd de deux entiers a et b est calculé en calculant d'abord sa valuation 2-adique (c'est le minimum des valuations de a et de b), puis en utilisant la formule :

$$\text{pgcd}(a,b) = \begin{cases} \text{pgcd}\left(\frac{a}{2}, b\right) & \text{si} \quad a \text{ est pair,} \\ \text{pgcd}\left(a, \frac{b}{2}\right) & \text{si} \quad b \text{ est pair,} \\ \text{pgcd}\left(\frac{a+b}{2}, \frac{a-b}{2}\right) & \text{si} \quad a \text{ et } b \text{ sont impairs.} \end{cases}$$

Le problème avec cet algorithme est que le nombre de divisions nécessaires pour aboutir au pgcd peut éventuellement être quadratique en $\max(l(a), l(b))$: pour le voir, on peut par exemple prendre l'exemple de Purdy $(a, b) = (1, 2^n - 1)$.

Nous donnons maintenant deux algorithmes pour effectuer la division binaire généralisée. Le premier, décrit à la figure 1.5 et illustré à la figure 1.6, est l'équivalent de la division euclidienne classique et est asymptotiquement plus lent que le second, décrit à la figure 1.7 qui est l'équivalent de la division rapide. Pour la plupart de ses paires d'entrée, l'algorithme d'Euclide reposant sur la DBG effectue quasiment toutes ses divisions sur des paires (c, d) pour lesquelles $\nu_2(d) - \nu_2(c)$ est petit. Pour cette raison, le premier algorithme (naïf) suffit en pratique.

Lemme 1. *L'algorithme* **DBG-élémentaire** *de la figure 1.5 est correct, et si l'entrée (a, b) satisfait $l(a), (b) \leqslant n$, alors il finit en temps $O\left(n \cdot [\nu_2(b) - \nu_2(a)]\right)$.*

Il est aussi possible d'effectuer une division binaire généralisée en temps quasi-linéaire, dans le cas où l'on utilise la multiplication de Schönhage et Strassen [158], en se servant du relèvement de Hensel (qui est l'analogue p-adique de l'itération de Newton).

[5]La division « Plus-Moins » est en réalité quasiment indépendante des bits de poids fort : au début de l'algorithme euclidien correspondant, la différence des longueurs des opérandes est calculée. Cela implique en particulier que les quotients calculés au cours des l'algorithme dépendent des longueurs et donc des poids forts des restes initiaux. Cependant, ces longueurs ne sont calculées qu'une seule fois, au début de l'algorithme. Il est peut-être possible de construire une variante récursive rapide de cet algorithme, qui considère les restes successifs par les bits de poids faible.

Algorithme: **DBG-élémentaire.**
Entrée : Deux entiers a, b tels que $\nu_2(a) < \nu_2(b) < \infty$.
Sortie : $(q, r) = \mathrm{DBG}(a, b)$.

1. q:=0, r:=a.
2. Tant que $\nu_2(r) \leqslant \nu_2(b)$, faire
3. $\quad q$:=$q - 2^{\nu_2(r)-\nu_2(a)}$,
4. $\quad r$:=$r - 2^{\nu_2(r)-\nu_2(b)}b$.
5. q:=$q \bmod 2^{\nu_2(b)-\nu_2(a)+1}$, r:=$q\frac{b}{2^{\nu_2(b)-\nu_2(a)}} + a$.
6. Renvoyer (q, r).

FIG. 1.5 – Un algorithme naïf de division binaire généralisée.

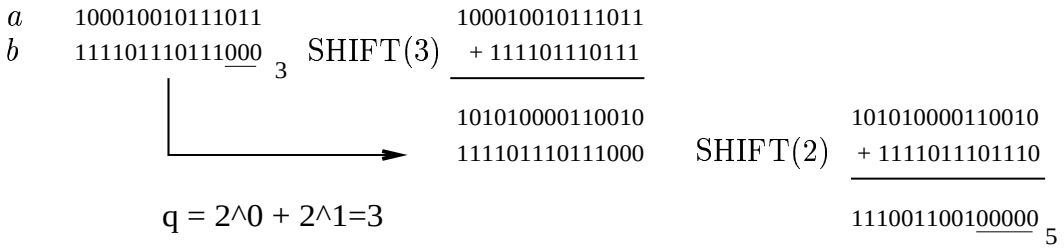


FIG. 1.6 – Un exemple d'exécution de l'algorithme **DBG-élémentaire**.

Lemme 2. *L'algorithme* **DBG-rapide** *de la figure 1.7 est correct, et avec l'algorithme de multiplication de Schönhage et Strassen, si a et b satisfont $l(a), l(b) \leqslant 2n$ et $\nu_2(b) - \nu_2(a) \leqslant n$, alors il finit en temps $O(M(n))$.*

Dans l'algorithme de la figure 1.7, les étapes 3 et 4 servent à calculer l'inverse de B modulo 2^n. L'étape 5 calcule le DBG-quotient q, et le DBG-reste correspondant est calculé à l'étape 6.

1.2.2 L'algorithme d'Euclide binaire généralisé.

Il est très naturel de dériver un algorithme Euclidien binaire généralisé de la division binaire généralisée. Cela est fait à la figure 1.8. L'algorithme est illustré par un exemple à la figure 1.9.

Lemme 3. *L'algorithme* **Euclide-BG** *de la figure 1.8 est correct, et si nous utilisons l'algorithme* **DBG-élémentaire** *de la figure 1.5, alors pour une entrée (a, b) satisfaisant $l(a), l(b) \leqslant n$, il finit en temps $O(n^2)$.*

Démonstration. Soit $r_0 = a, r_1 = b, r_2, \ldots$ la suite des restes apparaissant lors de l'exécution de l'algorithme. Nous montrons dans un premier temps que cette suite est finie et donc que l'algorithme termine.

Algorithme: **DBG-rapide**.
Entrée : Deux entiers a, b tels que $\nu_2(a) < \nu_2(b) < \infty$.
Sortie : $(q, r) = \mathrm{DBG}(a, b)$.

1. $A{:=}-\frac{a}{2^{\nu_2(a)}}, B{:=}\frac{b}{2^{\nu_2(b)}}, n{:=}\nu_2(b) - \nu_2(a) + 1$.
2. $q{:=}1$.
3. Pour i de 1 à $\lceil \lg n \rceil$, faire
4. $\quad q{:=}q + q(1 - Bq) \mod 2^{2^i}$.
5. $q{:=}Aq \mod 2^n$.
6. $r{:=}a + \frac{q}{2^{n-1}}b$.
7. Renvoyer (q, r).

FIG. 1.7 – Un algorithme rapide de division binaire généralisée.

Algorithme: **Euclide-BG**.
Entrée : Deux entiers a, b tels que $\nu_2(a) < \nu_2(b)$.
Sortie : La partie impaire $\frac{g}{2^{\nu_2(g)}}$ de $g = \mathrm{pgcd}(a, b)$.

1. Si $b = 0$, renvoyer $\frac{a}{2^{\nu_2(a)}}$.
2. $(q, r){:=}\mathrm{DBG}(a, b)$.
3. Renvoyer **Euclide-BG**(b, r).

FIG. 1.8 – L'algorithme d'Euclide binaire généralisé.

Pour tout k positif ou nul, les équations de la définition 18 assurent que $|r_{k+2}| \leqslant |r_{k+1}| + |r_k|$. On en déduit que $|r_k| \leqslant 2^{n+1}\left(\frac{1+\sqrt{5}}{2}\right)^k$. De plus, 2^k divise $|r_k|$, ce qui donne

$$2^k \leqslant |r_k| \leqslant 2^{n+1}\left(\frac{1+\sqrt{5}}{2}\right)^k,$$

et donc $\left(1 - \lg\frac{1+\sqrt{5}}{2}\right)k \leqslant n + 1$. Il y a donc $\leqslant 3.28(n+1) = O(n)$ restes dans la suite des restes. Soit $t + 1 = O(n)$ la longueur de cette suite : r_t est le dernier reste non nul. Le lemme 1 nous permet de borner le coût de chaque appel à une division binaire généralisée : le coût de diviser r_k par r_{k+1} est borné par $O\left(l(|r_k|) \cdot [\nu_2(r_{k+1}) - \nu_2(r_k)]\right) = O\left(n \cdot [\nu_2(r_{k+1}) - \nu_2(r_k)]\right)$, et donc le coût total est borné par $O(n\nu_2(r_t)) = O(n^2)$.

Pour la correction de l'algorithme, il suffit de remarquer que le reste binaire généralisé r de a par b satisfait $r = a \mod b'$ où $b' = \frac{b}{2^{\nu_2(b)}}$ est la partie impaire de b. Par conséquent $\mathrm{pgcd}(a, b') = \mathrm{pgcd}(r, b')$. □

Pour $n \geqslant 1$ et $q \in [|-2^n + 1, 2^n - 1|]$, nous définissons la matrice $[q]_n = \begin{pmatrix} 0 & 2^n \\ 2^n & q \end{pmatrix}$. Soient r_0, r_1 deux entiers non nuls avec $0 = \nu_2(r_0) < \nu_2(r_1)$. Soient $r_0, r_1, r_2, \ldots$ leur suite

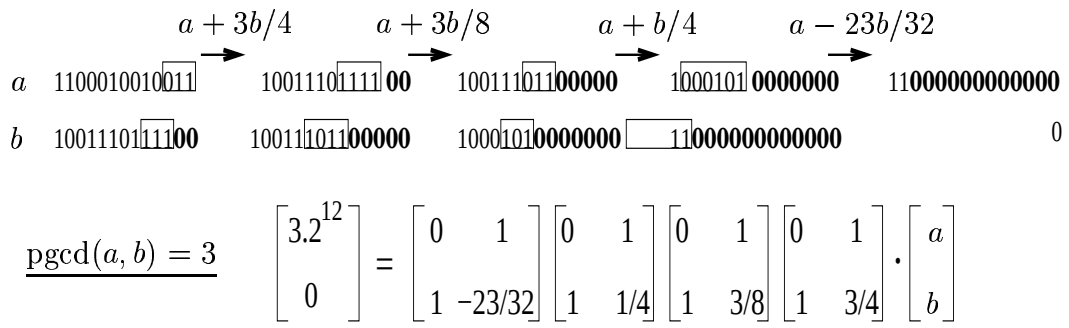


FIG. 1.9 – Un exemple d'exécution de l'algorithme **Euclide-BG**.

de restes binaires généralisés, et $q_1, q_2, \ldots$ leur suite de quotients binaires généralisés :

$$\forall i \geqslant 1, r_{i+1} = r_{i-1} + q_i \frac{r_i}{2^{\nu_2(r_i)-\nu_2(r_{i+1})}}.$$

Alors on a la relation vectorielle suivante :

$$\forall i \geqslant 1, \begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix} = \frac{1}{2^{\nu_2(r_i)}} [q_i]_{n_i} \ldots [q_1]_{n_1} \cdot \begin{pmatrix} r_0 \\ r_1 \end{pmatrix},$$

où $n_i = \nu_2(r_i) - \nu_2(r_{i-1})$ pour tout $i \geqslant 1$.

Dans la suite, nous utilisons le résultat élémentaire suivant :

Lemme 4. *Soient r_0, r_1 deux entiers non nuls avec $0 = \nu_2(r_0) < \nu_2(r_1)$, et $r_0, r_1, r_2, \ldots$ leur suite de restes binaires généralisés. Soit $d \geqslant 0$. Alors il existe un unique entier $i \geqslant 0$ tel que $\nu_2(r_i) \leqslant d < \nu_2(r_{i+1})$.*

1.2.3 Sur la taille réelle des restes binaires généralisés.

Nous montrons maintenant qu'une meilleure borne que $|r_k| \leqslant 2^{n+1} \left(\frac{1+\sqrt{5}}{2}\right)^k$ et donc que $t \leqslant \frac{n+1}{1-\lg\frac{1+\sqrt{5}}{2}}$ peut être obtenue. Les bornes précédentes sont suffisantes pour garantir la quasi-linéarité de l'algorithme de la section suivante. Les bornes améliorées permettent de diminuer la constante multiplicative de la complexité asymptotique, et de mieux appréhender la réalité puisque certaines (en particulier celle sur la taille des coefficients des matrices de transformation) sont atteintes dans le cas le pire.

Le principe est de borner la taille de la matrice $\frac{1}{2^{\nu_2(r_i)}}[q_i]_{n_i} \ldots [q_1]_{n_1}$, où les q_j sont les quotients binaires généralisés d'une suite tronquée de restes binaires généralisés avec $\nu_2(r_i) \leqslant \nu_2(r_0) + d < \nu_2(r_{i+1})$ pour un certain $d \geqslant 1$. Cela permettra de borner plus finement les longueurs et le nombre de restes binaires généralisés.

Lemme 5. *Soit $d \geqslant 0$. Soient r_0, r_1 avec $0 = \nu_2(r_0) < \nu_2(r_1)$ et $r_0, r_1, \ldots, r_{i+1}$ leurs premiers restes binaires généralisés, où i est choisi tel que $\nu_2(r_i) \leqslant d < \nu_2(r_{i+1})$. On considère la matrice $\frac{1}{2^{\nu_2(r_i)}}[q_i]_{n_i} \ldots [q_1]_{n_1}$, où les q_j sont les quotients binaires généralisés de (r_0, r_1), et $n_j = \nu_2(r_j) - \nu_2(r_{j-1})$ pour tout $1 \leqslant j \leqslant i$. Soit M le maximum des valeurs absolues des quatre entrées de la matrice. Alors :*

1. *Si* $d = 0$ *ou* $d = 1$, *alors* $M = 1$.
2. *Si* $d = 3$, *alors* $M \leqslant \frac{11}{8}$.
3. *Si* $d = 5$, *alors* $M \leqslant \frac{67}{32}$.
4. *Si* $d = 2$, *ou* $d = 4$, *ou* $d \geqslant 6$, *alors* $M \leqslant \frac{2}{\sqrt{17}} \left(\left(\frac{1+\sqrt{17}}{4} \right)^{d+1} - \left(\frac{1-\sqrt{17}}{4} \right)^{d+1} \right)$.

De plus, toutes ces bornes sont atteintes, et la dernière l'est en particulier quand tous les n_j *et tous les* q_j *valent* 1.

La preuve de ce lemme est similaire à l'analyse dans le cas le pire de l'algorithme de Lagrange de [181]. L'algorithme de Lagrange [102] a aussi été décrit, ultérieurement, par Gauss [65], et est souvent appelé, en particulier dans [181], algorithme de Gauss.

Démonstration. Nous introduisons un ordre partiel sur les matrices 2×2 : $A \leqslant_{2\times 2} B$ si et seulement si pour toute coordonnée $[i, j] \in \{1, 2\}^2$, on a $A[i, j] \leqslant B[i, j]$. Tout d'abord, on peut restreindre l'étude au cas où les q_j sont tous positifs, grâce à l'inégalité :

$$|[q_i]_{n_i} \dots [q_1]_{n_1}| \leqslant_{2\times 2} [|q_i|]_{n_i} \dots [|q_1|]_{n_1}.$$

Celle-ci s'obtient facilement par récurrence sur i en utilisant les inégalités $|A \cdot B| \leqslant_{2\times 2} |A| \cdot |B|$, et en utilisant le fait que si les entrées des matrices A, A', B, B' sont positives, alors $A \leqslant_{2\times 2} A'$ et $B \leqslant_{2\times 2} B'$ impliquent $A \cdot B \leqslant_{2\times 2} A' \cdot B'$.

Par conséquent on cherche les éléments maximaux pour l'ordre partiel $\leqslant_{2\times 2}$ dans l'ensemble :

$$\{\Pi_{n_1+\dots+n_i \leqslant d} [q_i]_{n_i} \dots [q_1]_{n_1} \ , \ \forall 1 \leqslant j \leqslant i, 0 < q_j < 2^{n_j} \ \text{ et } n_j \geqslant 1\}.$$

On peut restreindre l'analyse au cas où $n_1 + \dots + n_i = d$ et où tous les q_j sont maximaux. Ceci donne l'ensemble restreint :

$$\{\Pi_{n_1+\dots+n_i=d} [2^{n_i} - 1]_{n_i} \dots [2^{n_1} - 1]_{n_1}\}.$$

Remarquons maintenant que $[2^n - 1]_n \leqslant_{2\times 2} [1]_1^n$ dès que $n \geqslant 3$. Il suffit donc de considérer la situation où tous les n_j valent soit 1 soit 2. De plus, si $j \geqslant 0$, alors $[3]_2 \cdot [1]_1^j [3]_2 \leqslant_{2\times 2} [1]_1^{+4}$, et on a aussi les relations $[3]_2^2 \leqslant_{2\times 2} [1]_1^4$, $[1]_1^5 \cdot [3]_2 \leqslant_{2\times 2} [1]_1^7$, $[3]_2 \cdot [1]_1^5 \leqslant_{2\times 2} [1]_1^7$ et $[1]_1^2 \cdot [3]_2 \cdot [1]_1^2 \leqslant_{2\times 2} [1]_1^6$. Ceci nous permet d'obtenir facilement les éléments maximaux :

- Pour $d = 0$: Id_2.
- Pour $d = 1$: $[1]_1$.
- Pour $d = 2$: $[1]_1^2$ et $[3]_2$.
- Pour $d = 3$: $[1]_1^3$, $[3]_2 \cdot [1]_1$ et $[1]_1 \cdot [3]_2$.
- Pour $d = 4$: $[1]_1^4$, $[3]_2 \cdot [1]_1^2$, $[1]_1 \cdot [3]_2 \cdot [1]_1$ et $[1]_1^2 \cdot [3]_2$.
- Pour $d = 5$: $[1]_1^5$, $[3]_2 \cdot [1]_1^3$, $[1]_1 \cdot [3]_2 \cdot [1]_1^2$, $[1]_1^2 \cdot [3]_2 \cdot [1]_1$ et $[1]_1^3 \cdot [3]_2$.
- Pour $d = 6$: $[1]_1^6$, $[3]_2 \cdot [1]_1^4$, $[1]_1 \cdot [3]_2 \cdot [1]_1^3$, $[1]_1^3 \cdot [3]_2 \cdot [1]_1$ et $[1]_1^4 \cdot [3]_2$.
- Pour $d = 7$: $[1]_1^7$, $[1]_1 \cdot [3]_2 \cdot [1]_1^4$ et $[1]_1^4 \cdot [3]_2 \cdot [1]_1$.
- Pour $d \geqslant 8$: $[1]_1^8$.

La fin de la preuve est évidente : il suffit de remarquer que $2^{-d}[1]_1^d = \begin{pmatrix} u_{d-1} & u_d \\ u_d & u_{d+1} \end{pmatrix}$, où $u_0 = 0$, $u_1 = 1$ et $u_i = u_{i-2} + \frac{1}{2}u_{i-1}$ pour $i \geqslant 2$. □

De ce résultat sur les matrices des quotients on peut obtenir des bornes sur les tailles des restes binaires généralisés, et sur la longueur de la suite de ces restes.

Théorème 5. *Soient r_0, r_1 deux entiers non nuls satisfaisant $0 = \nu_2(r_0) < \nu_2(r_1)$, et $r_0, r_1, \ldots, r_{t+1}$ leur suite de restes binaires généralisés avec $r_{t+1} = 0$. Soit $9 \leqslant j \leqslant t$. Alors :*

$$2^{\nu_2(r_j)} \leqslant |r_j| \leqslant \frac{2}{\sqrt{17}} \left[|r_0| \left(\omega^{\nu_2(r_j)-1} - (\bar{\omega})^{\nu_2(r_j)-1} \right) + |r_1| \left(\omega^{\nu_2(r_j)} - (\bar{\omega})^{\nu_2(r_j)} \right) \right],$$

avec $\omega = \frac{1+\sqrt{17}}{4}$ et $\bar{\omega} = \frac{1-\sqrt{17}}{4}$. De plus, si $l(r_0), l(r_1) \leqslant n$, alors on a :

$$t \leqslant \frac{n+1}{\lg(\sqrt{17}-1) - 1}.$$

Démonstration. La borne inférieure sur $|r_j|$ vient du fait que $2^{\nu_2(r_j)}$ divise $|r_j|$. Pour la borne supérieure, il suffit d'utiliser la preuve du lemme précédent avec $d = \nu_2(r_{j-1})$: on a $|M| \leqslant_2 \begin{pmatrix} u_{d-1} & u_d \\ u_d & u_{d+1} \end{pmatrix}$ si $d \geqslant 9$. En majorant la borne supérieure, on obtient :

$$2^{\nu_2(r_j)} \leqslant \frac{2^{n+2}}{\sqrt{17}} \omega^{\nu_2(r_j)} \left(1 + \frac{1}{\omega}\right) \leqslant 2^{n+1} \omega^{\nu_2(r_j)},$$

ce qui donne $\nu_2(r_t) \leqslant \frac{n+1}{1-\lg\omega}$. On peut finir la preuve en remarquant que $t \leqslant \nu_2(r_t)$. □

Pour comparer, nous rappelons que le pire cas pour le nombre d'itérations t de l'algorithme d'Euclide classique est atteint par la suite de Fibonacci, avec $t \leqslant \frac{n}{\lg(\sqrt{5}+1)-1} + o(n)$. Remarquons enfin que :

$$\frac{1}{\lg(\sqrt{5}+1) - 1} \approx 1.440 \quad \text{et} \quad \frac{1}{\lg(\sqrt{17}-1) - 1} \approx 1.555.$$

Une analyse en moyenne de l'algorithme d'Euclide binaire généralisé a été effectuée par Daireaux, Maume-Deschamps et Vallée [48]. L'algorithme peut être assimilé à une course entre un lièvre 2-adique qui fait diminuer de 2 bits en moyenne la longueur d'un reste à chaque itération, et une tortue réelle qui fait augmenter la longueur du reste de $\gamma \approx 0.05$ bits en moyenne à chaque itération, avec

$$\gamma = \lim_{k \to \infty} \frac{1}{k} \mathbb{E}[\log \|N_1 \cdot N_2 \cdot \ldots \cdot N_k\|],$$

où N_i est tirée aléatoirement uniformément parmi les $2^{-i}[q]_i$, et $\|A\|$ est le maximum des valeurs absolues des entrées de la matrice A. Cette analyse montre que le nombre moyen d'itérations de boucles pour des restes initiaux de longueurs inférieures à n bits est de l'ordre de $\frac{n}{2-\gamma} \approx 0.51 \cdot n$, et que les longueurs des entrées de la matrice produit des matrices des quotients sont de l'ordre de $\left(1 + \frac{\gamma}{2}\right) n \approx 1.025 \cdot n$. La thèse de Daireaux [47] contient une analyse plus détaillée.

1.2.4 Calcul d'inverses modulaires.

Ce paragraphe est indépendant du reste du chapitre, mais trouve sa justification dans le fait qu'une des principales applications de l'algorithme d'Euclide est le calcul d'inverses modulaires.

Soient a et b deux entiers non nuls satisfaisant $0 = \nu_2(a) < \nu_2(b)$ et $l(a), l(b) \leqslant n$. Supposons que l'on veuille calculer l'inverse de b modulo a, en utilisant un algorithme d'Euclide étendu reposant sur la division binaire généralisée. L'exécution de cet algorithme fournit deux entiers A et B tels que $Aa + Bb = 2^\alpha g$, où $\alpha = O(n)$ et $g = \text{pgcd}(a, b)$. Avec une telle relation il est très facile de vérifier si $g = 1$, et donc si l'inverse modulaire B' de b modulo a existe bien. De la relation $Aa + Bb = 2^\alpha$ on déduit alors :

$$B' = \frac{B}{2^\alpha} \mod a.$$

Par conséquent, pour obtenir B' il suffit de calculer l'inverse de 2^α modulo a. En utilisant le relèvement de Hensel (comme pour l'algorithme **DBG-rapide** de la figure 1.7), on peut obtenir l'inverse de a modulo 2^α. Cela donne deux entiers x et y tels que $xa + y2^\alpha = 1$. Clairement, y est l'inverse de 2^α modulo a.

La multiplication, le relèvement de Hensel et la division d'entiers de taille $O(n)$ peuvent être effectuées en temps $O(M(n))$ (voir [64]). Étant donnés deux entiers a et b, le temps additionnel pour calculer l'inverse de b modulo a une fois que l'on a effectué l'algorithme d'Euclide étendu binaire généralisé est donc en $O(M(n))$.

1.3 L'algorithme binaire rapide.

Nous décrivons maintenant l'algorithme récursif rapide reposant sur la division binaire généralisée. Cette description est proche de celle de [190], en ce qui concerne la récursion. Nous introduisons deux routines : l'algorithme **Demi-pgcd-BG**, décrit à la figure 1.10 et illustré par la figure 1.11, puis l'algorithme **Pgcd-BG-rapide**, décrit à la figure 1.12. Si on lui donne en entrée deux entiers r_0 et r_1 satisfaisant $0 = \nu_2(r_0) < \nu_2(r_1)$, l'algorithme **Demi-pgcd-BG** renvoie en sortie les deux DBG-restes r_i et r_{i+1} de la suite des DBG-restes de (r_0, r_1) qui satisfont $\nu_2(r_i) \leqslant k < \nu_2(r_{i+1})$: le paramètre k est donné en entrée et est voué à être proche de $\ell(r_0)/2$. L'algorithme renvoie aussi la matrice correspondante $2^{-(n_1+\ldots+n_i)}[q_i]_{n_i} \ldots [q_1]_{n_1}$. De son côté, l'algorithme **Pgcd-BG-rapide** prend en entrée deux entiers a et b, et renvoie leur pgcd en faisant un appel à l'algorithme **Demi-pgcd-BG**, ainsi qu'un appel récursif.

L'algorithme **Demi-pgcd-BG** fonctionne de la façon suivante : un premier quart des bits de poids faible de a et b est éliminé grâce à un appel récursif sur les $k/2$ bits de poids faible de a et b. Le point crucial est que lors de cet appel récursif, les quotients calculés sont des premiers DBG-quotients de (a, b). Ainsi, en multipliant a et b par la matrice obtenue récursivement, on obtient deux restes a' et b' de la suite des restes binaires généralisés de (a, b). On effectue alors une division binaire généralisée sur (a', b'), ce qui crée une nouvelle paire de restes (b', r). À ce moment-là on effectue un deuxième appel récursif sur les $\approx k/2$ bits de poids faible de b' et r. La taille des entrées de ce deuxième appel récursif est donc similaire à celle du premier appel récursif. Enfin, les restes (c, d)

Algorithme: **Demi-pgcd-BG**.
Entrée : Deux entiers a, b tels que $0 = \nu_2(a) < \nu_2(b)$, et $k \geqslant 0$.
Sortie : Un entier j, une matrice à coefficients entiers R et deux entiers c et d tels que $0 = \nu_2(c) < \nu_2(d)$, $\begin{pmatrix} c \\ d \end{pmatrix} = 2^{-2j} R \cdot \begin{pmatrix} a \\ b \end{pmatrix}$, et $(c^* = 2^j c, d^* = 2^j d)$ sont les deux restes consécutifs de la suite des DBG-restes de (a, b) qui satisfont $\nu_2(c^*) \leqslant k < \nu_2(d^*)$.

1. Si $\nu_2(b) > k$, renvoyer $0, \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, a, b$.
2. $k_1 := \lfloor k/2 \rfloor$.
3. $a := a_1 2^{2k_1+1} + a_0, b := b_1 2^{2k_1+1} + b_0$ avec $0 \leqslant a_0, b_0 < 2^{2k_1+1}$.
4. $j_1, R_1, c_1, d_1 :=$ **Demi-pgcd-BG**(a_0, b_0, k_1).
5. $\begin{pmatrix} a' \\ b' \end{pmatrix} := 2^{2k_1+1-2j_1} R_1 \cdot \begin{pmatrix} a_1 \\ b_1 \end{pmatrix} + \begin{pmatrix} c_1 \\ d_1 \end{pmatrix}, \quad j_0 := \nu_2(b')$.
6. Si $j_0 + j_1 > k$, renvoyer j_1, R_1, a', b'.
7. $(q, r) :=$ DBG$'(a', b')$.
8. $k_2 := k - (j_0 + j_1)$.
9. $\frac{b'}{2^{j_0}} := b'_1 2^{2k_2+1} + b'_0$, $r := r_1 2^{2k_2+1} + r_0$ avec $0 \leqslant b'_0, r_0 < 2^{2k_2+1}$.
10. $j_2, R_2, c_2, d_2 :=$ **Demi-pgcd-BG**(b'_0, r_0, k_2).
11. $\begin{pmatrix} c \\ d \end{pmatrix} := 2^{2k_2+1-2j_2} R_2 \cdot \begin{pmatrix} b'_1 \\ r_1 \end{pmatrix} + \begin{pmatrix} c_2 \\ d_2 \end{pmatrix}$.
12. Renvoyer $j_1 + j_0 + j_2, R_2 \cdot [q]_{j_0} \cdot R_1, c, d$.

FIG. 1.10 – L'algorithme **Demi-pgcd-BG**. On fixera par exemple $k = \lfloor l(a)/2 \rfloor$.

correspondant dans la suite des DBG-restes de (a, b) sont calculés grâce à la matrice de transformation. La matrice de transformation globale est calculée à partir des deux matrices de transformation obtenues lors des appels récursifs et du quotient de la division binaire généralisée effectuée entre ces deux appels récursifs.

Il est utile de remarquer que dans la description de l'algorithme **Demi-pgcd-BG** donnée à la figure 1.10, on utilise une routine DBG'. Celle-ci est une simple modification de la division binaire généralisée : si on lui donne en entrée deux entiers a et b satisfaisant $0 = \nu_2(a) < \nu_2(b)$, sa sortie sera le DBG-quotient q et $\frac{r}{2^{\nu_2(b)}}$, où r est le DBG-reste de (a, b). Dans le même esprit, l'entier j renvoyé par l'algorithme **Demi-pgcd-BG** correspond à la plus grande puissance de 2 apparaissant dans les dénominateurs de la matrice de transformation. En considérant ainsi les dénominateurs à part, on travaille sur des matrices de transformation à coefficients entiers.

L'avantage principal de notre algorithme par rapport aux autres algorithmes quasi-linéaires de calcul de pgcd est que la matrice R renvoyée par un appel récursif de l'algorithme **Demi-pgcd-BG** est correcte : elle est le produit de matrices de quotients qui sont les « vrais quotients ». Il n'y a pas besoin de réparer cette matrice pour la rendre correcte, et donc en particulier il n'y a pas besoin de stocker une suite des quotients. La raison fondamentale est que les restes voient leur taille diminuer par les bits de poids faible vers les bits de poids fort, et comme les retenues vont aussi dans ce sens, elles n'in-

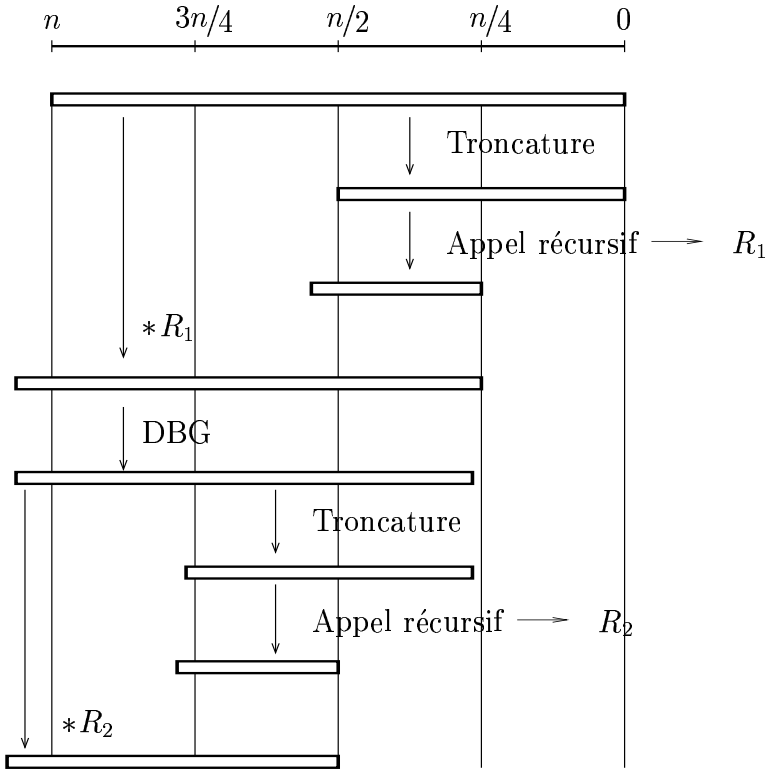


FIG. 1.11 – Représentation graphique de l'algorithme **Demi-pgcd-BG**.

Algorithme: **Pgcd-BG-Rapide**.
Entrée : Deux entiers a, b tels que $0 = \nu_2(a) < \nu_2(b)$.
Sortie : $g = \mathrm{pgcd}(a, b)$.

1. j, R, a', b':=**Demi-pgcd-BG**$(a, b, \lfloor l(a)/2 \rfloor)$.
2. Si $b' = 0$, renvoyer a'.
3. (q, r):=DBG$'(a', b')$.
4. Renvoyer **Pgcd-BG-rapide**(b', r').

FIG. 1.12 – L'algorithme **Pgcd-BG-rapide**.

terfèrent pas. L'algorithme est aussi simple que la version polynomiale de l'algorithme de Knuth-Schönhage.

1.4 Correction et analyse de complexité de l'algorithme binaire rapide.

Pour montrer la correction de l'algorithme **Pgcd-BG-rapide**, nous procédons comme suit : nous montrons d'abord des propriété sur la division binaire généralisée, puis prouvons la correction de l'algorithme **Demi-pgcd-BG**, qui implique directement celle de l'algorithme **Pgcd-BG-rapide**. Nous montrons dans un deuxième temps que cet algorithme a une complexité asymptotique $O(M(n)\log n)$.

1.4.1 Correction de l'algorithme Pgcd-BG-rapide.

Les propriétés décrites ci-dessous sont très similaires à celles qui rendent possible l'algorithme de Knuth-Schönhage dans le cas de la division euclidienne classique. Le premier lemme explique que les $\nu_2(b) - \nu_2(a) + 1$ bits de poids faible de a et de b suffisent à déterminer de manière unique le DBG-quotient de (a, b).

Lemme 6. *Soient a, a', b et b' des entiers tels que $a' = a \mod 2^l$, $b' = b \mod 2^l$ pour un certain $l \geqslant 2\nu_2(b) + 1$. Supposons de plus que $0 = \nu_2(a) < \nu_2(b)$. Soient q et r le DBG-quotient et le DBG-reste de (a, b), et q' et r' le DBG-quotient et le DBG-reste de (a', b'). Alors :*

$$q = q' \quad \text{et } r = r' \mod 2^{l-\nu_2(b)}.$$

Démonstration. Par définition, on a $q = -a\left(\frac{b}{2^{\nu_2(b)}}\right)^{-1} \operatorname{cmod} 2^{\nu_2(b)+1}$ et de manière identique on a aussi $q' = -a'\left(\frac{b'}{2^{\nu_2(b')}}\right)^{-1} \operatorname{cmod} 2^{\nu_2(b')+1}$. Comme $l \geqslant 2\nu_2(b) + 1$, on a $\frac{b}{2^{\nu_2(b)}} = \frac{b'}{2^{\nu_2(b')}} \mod 2^{\nu_2(b)+1}$, et aussi $a = a' \mod 2^{\nu_2(b)+1}$. Par conséquent $q = q'$. De plus, comme $r = a + q\frac{b}{2^{\nu_2(b)}}$ et $r' = a' + q\frac{b'}{2^{\nu_2(b')}}$, on a $r = r' \mod 2^{l-\nu_2(b)}$. □

Ce résultat peut être interprété comme une propriété de continuité : deux paires d'entiers 2-adiques (a, b) et (a', b') qui sont suffisamment proches pour la norme 2-adique (c'est-à-dire les bits de poids faible de a et de a' sont identiques jusqu'à un certain rang et il en est de même pour b et b'), ont le même DBG-quotient et des DBG-restes similaires (plus les paires sont proches, plus les restes sont proches aussi). Le lemme suivant étend le lemme 6 au développement en fraction continue pour la division binaire généralisée : si les paires (a, b) et (a', b') sont assez proches, alors les premiers termes de leurs suites de DBG-quotients sont identiques. On obtient ce résultat en appliquant plusieurs fois le lemme 6.

Lemme 7. *Soient a, a', b et b' des entiers tels que $a = a' \mod 2^{2k+1}$ et $b = b' \mod 2^{2k+1}$ pour un certain $k \geqslant 0$. Supposons que $0 = \nu_2(a) < \nu_2(b)$. Soient $r_0 = a, r_1 = b, r_2, \ldots$ la suite des DBG-restes de (a, b) et $q_1, q_2, \ldots$ les DBG-quotients correspondants. Soient $r'_0 = a', r'_1 = b', r'_2, \ldots$ la suite des DBG-restes de (a', b') et $q'_1, q'_2, \ldots$ les DBG-quotients*

correspondants, c'est-à-dire que pour tout $j \geqslant 1$, $r_{j+1} = r_{j-1} + q_j \frac{r_j}{2^{n_j}}$ *et* $r'_{j+1} = r'_{j-1} + q'_j \frac{r'_j}{2^{n'_j}}$ *avec* $n_j = \nu_2(r_j) - \nu_2(r_{j-1})$ *et* $n'_j = \nu_2(r'_j) - \nu_2(r'_{j-1})$.

Alors si r_{i+1} *est le premier DBG-reste tel que* $\nu_2(r_{i+1}) > k$, *nous avons :*

$$\forall j \leqslant i, q_j = q'_j \quad et \quad r_{j+1} = r'_{j+1} \mod 2^{2k+1-\nu_2(r_j)}.$$

Démonstration. Nous prouvons ce résultat par récurrence sur $j \geqslant 0$. Il est correct pour $j = 0$ car $a = a' \mod 2^{2k+1}$ et $b = b' \mod 2^{2k+1}$. Supposons désormais que $1 \leqslant j \leqslant i$. On utilise le lemme 6 avec $\frac{r_{j-1}}{2^{\nu_2(r_{j-1})}}$, $\frac{r_j}{2^{\nu_2(r_{j-1})}}$, $\frac{r'_{j-1}}{2^{\nu_2(r_{j-1})}}$, $\frac{r'_j}{2^{\nu_2(r_{j-1})}}$ et $l = 2k + 1 - 2\nu_2(r_{j-1})$. Par récurrence, nous avons :

$$r_{j-1} = r'_{j-1} \mod 2^{2k+1-\nu_2(r_{j-1})} \text{ et } r_j = r'_j \mod 2^{2k+1-\nu_2(r_{j-1})}.$$

Comme $j \leqslant i$, par définition de i nous avons $l \geqslant 2(\nu_2(r_j) - \nu_2(r_{j-1})) + 1$, et donc les hypothèses du lemme 6 sont vérifiées. Ainsi $q_j = q'_j$ et $r_{j+1} = r'_{j+1} \mod 2^{2k+1-\nu_2(r_j)}$. □

D'un point de vue pratique, ce lemme explique que k bits peuvent être gagnés par rapport à la paire initiale (a, b) en n'utilisant que $2k$ bits de a et $2k$ bits de b. L'amélioration par rapport à la division euclidienne classique est que pour cette dernière le lemme est seulement « à peu près vrai », car les derniers quotients avant de gagner k bits peuvent différer, et ont donc à être « réparés ».

Pour montrer la correction de l'algorithme **pgcd-BG-rapide**, il suffit de montrer celle de l'algorithme **Demi-pgcd-BG**, que l'on établit au théorème suivant. Plus précisément, on montre que l'algorithme **Demi-pgcd-BG** calcule exactement le début de la suite des DBG-quotients (jusqu'aux restes successifs correspondants déterminés uniquement comme annoncé à la figure 1.10). Cela nous garantira alors que l'algorithme 1.10) simule l'algorithme d'Euclide binaire généralisé (ils calculent tous les deux les mêmes suites de quotients), et donc qu'il est correct. Cela nous donnera aussi la terminaison de l'algorithme **pgcd-BG-rapide** : en effet, la suite des quotients est finie, et chaque appel à l'algorithme **pgcd-BG-rapide** qui ne termine pas à l'étape 2 effectue au moins une division binaire, donc progresse dans la suite des quotients.

Théorème 6. *L'algorithme* **Demi-pgcd-BG** *de la figure 1.10 est correct.*

Démonstration. Nous prouvons ce résultat par récurrence sur $k \geqslant 0$. Si $k = 0$, l'algorithme finit à l'étape 1 car $\nu_2(b) \geqslant 1$. Supposons désormais que $k \geqslant 1$ et que $\nu_2(b) \leqslant k$.

Comme $\lfloor \frac{k}{2} \rfloor < k$, l'étape 4 est un appel récursif strict (ses entrées satisfont bien la condition des valuations). Par récurrence, j_1, R_1, c_1 et d_1 satisfont :

$$\begin{pmatrix} c_1 \\ d_1 \end{pmatrix} = 2^{-2j_1} R_1 \cdot \begin{pmatrix} a_0 \\ b_0 \end{pmatrix},$$

et $2^{j_1} c_1$ et $2^{j_1} d_1$ sont exactement les restes consécutifs r'_{i_1} et r'_{i_1+1} de la suite des DBG-restes de $(r'_0 = a_0, r'_1 = b_0)$ qui satisfont $\nu_2(r'_{i_1}) \leqslant k_1 < \nu_2(r'_{i_1+1})$. Grâce au lemme 7, on sait que les entrées de $2^{-j_1} R_1 \cdot \begin{pmatrix} a \\ b \end{pmatrix}$ sont deux restes consécutifs $2^{j_1} a' = r_{i_1}$ et $2^{j_1} b' = r_{i_1+1}$ de la suite des DBG-restes de $(r_0 = a, r_1 = b)$, et ils satisfont $r_{i_1} = r'_{i_1} \mod 2^{k_1+1}$

et $r_{i_1+1} = r'_{i_1+1} \mod 2^{k_1+1}$. À partir de ces égalités, nous avons directement que $\nu_2(r_{i_1}) = \nu_2(r'_{i_1}) \leqslant k_1 < \nu_2(r_{i_1+1}) \leqslant \nu_2(r'_{i_1+1})$. Ainsi, si l'exécution de l'algorithme s'achève à l'étape 6, la sortie est correcte.

Sinon, on calcule r_{i_1+2} à l'étape 7. À l'étape 8 on calcule $k_2 = k - \nu_2(r_{i_1+1})$. Le test de l'étape 6 nous assure que k_2 est bien positif. Comme $\nu_2(r_{i_1+1}) > \lfloor \frac{k}{2} \rfloor$, nous avons $k_2 \leqslant \lceil \frac{k}{2} \rceil - 1 < k$. Ainsi l'étape 10 est un appel récursif strict (et ses entrées satisfont bien la condition des valuations). Par récurrence, j_2, R_2, c_2 et d_2 satisfont :

$$\begin{pmatrix} c_2 \\ d_2 \end{pmatrix} = 2^{-2j_2} R_2 \cdot \begin{pmatrix} b'_0 \\ r_0 \end{pmatrix},$$

et $2^{j_2}c_2$ et $2^{j_2}d_2$ sont exactement les restes consécutifs r'_{i_2} et r'_{i_2+1} de la suite des DBG-restes de (b_0, r'_0) qui satisfont $\nu_2(r'_{i_2}) \leqslant k_2 < \nu_2(r'_{i_2+1})$. Le lemme 7 nous assure que les entrées du vecteur $2^{-j_2} R_2 \cdot \begin{pmatrix} 2^{j_1} b' \\ 2^{j_1} r' \end{pmatrix}$ sont deux restes consécutifs r_i et r_{i+1} (avec $i = i_1 + i_2 + 1$) de la suite des DBG-restes de (a, b), que $\frac{r_i}{2^{j_1+\nu_2(b')}} = 2^{j_2}c_2 \mod 2^{k_2+2}$ et que $\frac{r_{i_1+1}}{2^{j_1+\nu_2(b')}} = 2^{j_2}d_2 \mod 2^{k_2+1}$. Ainsi la suite d'inégalités $\nu_2(r_i) = j_1 + j_2 + \nu_2(b') \leqslant k < \nu_2(r_{i+1})$ est correcte. Ceci termine la preuve du théorème. □

1.4.2 Analyse de complexité de l'algorithme Pgcd-BG-rapide.

Dans ce qui suit, on notera $H(n)$ et $G(n)$ les nombres maximaux d'opérations élémentaires effectuées respectivement par les algorithmes **Demi-pgcd-BG** et **Pgcd-BG-rapide** quand on leur donne en entrée n'importe quelle paire valide d'entiers de longueurs au plus n.

Lemme 8. *Soit* $c = \frac{1}{2}\lg\frac{1+\sqrt{17}}{2} \approx 0.679$. *Nous avons les deux relations suivantes :*

$$\begin{aligned} H(n) &= 2H\left(\left\lfloor \frac{n}{2} \right\rfloor + 1\right) + O(M(n)), \\ G(n) &= H(n) + G\left(\lceil cn \rceil\right) + O(M(n)). \end{aligned}$$

Démonstration. La deuxième relation est une conséquence du lemme 5, car $\frac{1}{2}\lg\frac{1+\sqrt{17}}{4} + \frac{1}{2} = c$: après l'appel à l'algorithme **Demi-pgcd-BG**, le deuxième reste (c'est-à-dire le premier au début de l'appel récursif à l'algorithme **pgcd-BG-rapide**) a « perdu » plus de $n/2$ bits de poids faible, et gagné moins de $\frac{1+\sqrt{17}}{4}n$ bits de poids fort[6].

Nous nous intéressons désormais à la première relation. Les coûts des étapes $1, 2, 3, 6, 8$ et 9 de l'algorithme **Demi-pgcd-BG** sont négligeables. Les étapes 4 et 10 sont des appels récursifs donc les coûts sont chacun bornés par $H\left(\lfloor \frac{n}{2} \rfloor + 1\right)$. Les étapes $5, 11$ et 12 consistent en des multiplications d'entiers de taille $O(n)$. Enfin, l'étape 7 est une simple division binaire généralisée, et on a vu au lemme 2 que cela peut être effectué en $O(M(n))$ opérations élémentaires. □

[6]Le « **demi** » dans le nom de l'algorithme **demi-pgcd-BG** peut sembler inapproprié puisque l'on ne diminue pas de $n/2$ bits la taille des restes courants, mais de seulement $\approx 0.321 \cdot n$ bits (dans le cas le pire). Il faut plutôt interpréter le « **demi** » de la manière suivante : on gagne $n/2$ bits par les poids faibles, mais malencontreusement, pendant le processus on a rajouté des bits de poids fort.

À partir de ce résultat et du fait que $c < 1$, on déduit aisément que :

Théorème 7. *L'algorithme* **Pgcd-BG-rapide** *de la figure 1.12 termine en temps quasi-linéaire. Plus précisément,* $G(n) = O(M(n)\log n)$.

Les constantes multiplicatives que l'on peut dériver des preuves qui précèdent sont plutôt grandes et ne correspondent pas à grand chose en pratique. En fait, pour des entrées aléatoires de n-bits, les termes de la suite des DBG-quotients sont de l'ordre de $O(1)$, et donc l'étape 7 de l'algorithme **Demi-pgcd-BG** a un coût négligeable. De plus, la borne $O\left(\left(\frac{1+\sqrt{17}}{4}\right)^d\right)$ du lemme 5 qui est atteinte dans le cas le pire n'est vraisemblablement pas réaliste dans la pratique. En particulier, l'analyse en moyenne de [48] inciterait à remplacer le c du lemme 8 par ≈ 0.513. Supposons pour simplifier que la matrice de transformation qui fait gagner k bits pour les restes ait des entrées de longueur k. Alors pour des entrées de taille n, l'étape 5 de l'algorithme **Demi-pgcd-BG** coûte $4M(3n/8)$, car multiplier un nombre de $n/4$ bits par un nombre de $n/2$ bits coûte à peu près une multiplication de deux nombres de $3n/8$ bits. L'étape 11 coûte $4M(n/4)$. Pour le produit matriciel de l'étape 12, on peut utiliser l'algorithme de Strassen [175] pour n'utiliser que 7 multiplications $n/4 \times n/4$ au lieu de 8. L'étape 12 coûte donc $7M(n/4)$. Si l'on regroupe les différents coûts, on obtient que le nombre d'opérations élémentaires effectuées par l'algorithme **demi-pgcd-BG** quand on lui donne en entrée deux entiers d'au plus n bits dont l'un est pair et l'autre impair, et l'entier $k = \lfloor n/2 \rfloor$, est[7] :

$$H(n) \approx 2H\left(\frac{n}{2}\right) + \frac{17}{4}M(n) \approx \frac{17}{8}M(n)\lg n.$$

On obtient ainsi que l'algorithme **pgcd-BG-rapide** calcule le pgcd de deux entiers d'au plus n bits dont l'un est pair et l'autre impair en $G(n) \approx \frac{17}{4}M(n)\lg n$ opérations élémentaires.

1.5 Remarques sur l'implantation des algorithmes rapides de calcul de pgcd.

Nous avons implanté l'algorithme **Pgcd-BG-rapide** décrit plus haut en GNU MP [73], mais le code de Niels Möller reposant sur la méthode de [157] et décrit dans [132] est le plus efficace. Möller a comparé différents algorithmes de calcul rapide de pgcd et a montré expérimentalement qu'utiliser la division classique permet d'obtenir un algorithme plus rapide. Cela peut s'expliquer par le fait que dans notre algorithme, la matrice de transformation qui fait gagner k bits (de poids faible) sur les restes a des entrées plus grandes que celle qui fait gagner k bits (de poids fort) pour la division classique. Cela est vrai dans le cas le pire (d'après le lemme 5), et semble l'être expérimentalement dans le cas moyen. Cela induit des multiplications de nombres un peu plus gros.

Quelle que soit la division, autant dans notre code que celui de Möller, quelques astuces sont utilisées. Tout d'abord, certaines multiplications sont inutiles quand l'algorithme

[7] Dans cette équation, nous nous servons du fait que $M(n) = O(n\log n\log\log n)$, autrement nous obtiendrions 17/4 à la place de 17/8.

Pgcd-BG-rapide appelle l'algorithme **Demi-pgcd-BG**, on n'a pas besoin de calculer la matrice de transformation, le produit matriciel de l'étape 12 est superflu. Ensuite, en-dessous d'un certain seuil nous avons utilisé un algorithme naïf quadratique qui a les mêmes spécifications que l'algorithme **Demi-pgcd-BG**. De plus, nous avons transformé l'étape 7 : au lieu d'effectuer une seule division binaire généralisée, nous en faisons plusieurs de façon à ce que la matrice de transformation correspondante remplisse quatre entiers signés de 32 bits. Pour faire cela, nous ne considérons que les 64 bits de poids faible de chacun des opérandes, ce qui est suffisant pour obtenir les bons quotients d'après le lemme 7 (nous appliquons ce lemme avec $k = 31$: un bit est réservé pour le signe). On s'arrête donc d'effectuer des divisions binaires généralisée aux entiers quand les valeurs absolues des entrées de la matrice de transformation deviennent $\geqslant 2^{31}$ ou quand les 32 bits de poids faible d'un des deux entiers sont tous nuls.

D'autres idées non implantées permettraient d'accélérer de façon (très) significative :

1. Utiliser un algorithme de multiplication optimisé pour des entrées déséquilibrées de longueurs $(n, 2n)$, pour rendre plus efficace l'étape 5 de l'algorithme **Demi-pgcd-BG**. Dans le cas de la multiplication de Schönhage et Strassen, il faudrait calculer trois transformées de Fourier discrètes de taille $3n$, alors que pour le moment on divise le grand nombre en deux nombres de longueur n et on calcule six transformées de Fourier discrètes de taille $2n$. Pour la multiplication de Toom-Cook [179, 92], qui est celle utilisée quand les algorithmes de pgcd rapide deviennent compétitifs, la solution est moins simple.
2. On peut aussi remarquer que de très nombreuses transformées de Fourier discrètes peuvent être réutilisées dans le cas où on utilise la multiplication de Schönhage-Strassen, en particulier lors des produits matrice-vecteur. Cette remarque s'applique aussi à la multiplication de Toom-Cook. Malheureusement, pour le moment il n'est pas facile de « casser » les algorithmes de multiplication de GNU MP pour rendre cela possible (à part pour la transformée de Fourier discrète).
3. Utiliser la multiplication matricielle de Strassen à l'étape 12. Pour le moment on fait huit multiplication au lieu de sept.
4. Une autre amélioration peut être apportée quand l'algorithme est utilisé pour des entiers de tailles en-dessous du domaine de la multiplication de Schönhage-Strassen. Nous sommes en fait dans le domaine de la multiplication de Toom-Cook : multiplier deux entiers de n bits chacun coûte $TC(n) = O\left(n^{\log_3 5}\right)$, avec $\log_3 5 \approx 1.465$. Dans ce contexte, l'étape 5 de l'algorithme **Demi-pgcd-BG** coûte $8TC(n/4)$, l'étape 11 coûte $4TC(n/4)$, et l'étape 12 coûte $7TC(n/4)$. Si l'on regroupe les différents coûts, on obtient $H(n) \approx 6.562 \cdot TC(n)$. Supposons maintenant qu'à l'étape 1 de l'algorithme **Pgcd-BG-rapide**, on appelle l'algorithme **Demi-pgcd-BG** non pas sur a et b en entier, mais sur leurs γn bits de poids faible seulement, avec $\gamma \in [0, 1]$: on appelle l'algorithme **demi-pgcd-BG** sur la partie basse de a et b, avec $k \approx \frac{\gamma}{2}n$, pour « gagner » $\approx \frac{\gamma}{2}n$ bits de poids faible, en manipulant des entiers de longueurs au plus γn. Après cet appel, il faut donc maintenant utiliser la matrice de transformation pour retrouver les « vrais restes », ce qui coûte 4 multiplications $(1-\gamma)n \times \frac{\gamma}{2}n$ (pour mettre à jour les bits de poids fort des restes initiaux, qui avaient été délaissés avant l'appel à l'algorithme **demi-pgcd-BG**). À l'appel suivant de l'algorithme

Pgcd-BG-rapide, les entrées ont au plus $\left(1-\frac{\gamma}{2}\right)n$ bits. Faisons l'hypothèse simplificatrice que la multiplication d'un entier de an bits par un entier de bn bits pour $a > b$ coûte a/b multiplications de deux entiers de bn bits. Alors si $\gamma > 2/3$, reconstruire les restes coûte $\frac{2\gamma}{1-\gamma}TC((1-\gamma)n)$, et sinon cela coûte $8\frac{1-\gamma}{\gamma}TC\left(\frac{\gamma}{2}n\right)$. Trouver le coefficient γ qui minimise le coût revient à trouver le x minimum des solutions des deux équations suivantes en x et γ, où x est la constante telle que $G(n) = xTC(n)$:

$$x = 6.562 \cdot \gamma^{1.465} + 2\gamma(1-\gamma)^{0.465} + x\left(1-\frac{\gamma}{2}\right)^{1.465} \quad \text{avec} \quad \gamma \geqslant \frac{2}{3},$$
$$x = 6.562 \cdot \gamma^{1.465} + 2^{1.535}(1-\gamma)\gamma^{0.465} + x\left(1-\frac{\gamma}{2}\right)^{1.465} \quad \text{avec} \quad \gamma \leqslant \frac{2}{3}.$$

On obtient la solution optimale $\gamma_0 \approx 0.63$ et $x_0 \approx 9.85$. Pour comparer, si l'on avait pris $\gamma = 1$, on aurait obtenu $x \approx 10.3$. Cela représente un gain d'environ 5.

5. Il est aussi possible d'accélérer significativement la complexité asymptotique de l'algorithme en ne faisant pas renvoyer les restes courants à l'algorithme **Demi-pgcd-BG** et en utilisant la remarque suivante sur la multiplication rapide : multiplier deux entiers de αn et βn bits pour arriver au produit modulo $2^{\gamma n}+1$ coûte $O\left(M\left(\frac{\gamma}{2}n\right)\right)$. Ce résultat provient directement de la structure de l'algorithme de multiplication rapide. Nous modifions donc l'algorithme **Demi-pgcd-BG**. Maintenant, il ne renvoie que la matrice de transformation. À l'étape 5, pour obtenir a' et b', il faut maintenant effectuer le produit $R_1^t(a,b)$. Soit n le maximum des longueurs binaires de a et de b. On sait que a' et b' vont avoir une longueur de $\approx 3n/4$ bits chacun (sous les même hypothèses heuristiques qu'à la section 1.4). On peut donc effectuer les 4 multiplications modulo $2^{\approx 3n/4}+1$, ce qui coûte $\approx 4M(3n/8)$. On supprime l'étape 11 de l'algorithme **Demi-pgcd-BG**, et dans l'algorithme **Pgcd-BG-rapide**, on rajoute une multiplication matrice-vecteur pour calculer les restes courants après l'étape 1. On arrive donc aux équations :

$$\begin{aligned} H(n) &\approx 2H\left(\frac{n}{2}\right) + \frac{13}{4}M(n) \approx \frac{13}{8}M(n)\lg n, \\ G(n) &\approx \frac{13}{4}M(n)\lg n, \end{aligned}$$

où n est la taille maximale des deux restes initiaux donnés en entrée respectivement aux algorithmes **demi-pgcd-BG** et **pgcd-BG-rapide**, et où l'on a choisi $k \approx n/2$ dans l'algorithme **demi-pgcd-BG**. Il est bien entendu possible de jumeler cette remarque avec les autres, en particulier la seconde.

Chapitre II-2

La réduction de réseaux en petite dimension

Le travail de recherche correspondant à ce chapitre a été initié lors de mon stage de DEA, à l'École Normale Supérieure sous la direction et avec les conseils de Phong Nguyễn. Il a fait l'objet d'une publication dans les actes de la conférence ANTS VI (Sixth Algorithmic Number Theory Symposium) [138]. Le présent chapitre est une version étendue de [138].

Sommaire

2.1 Préliminaires. 50
2.1.1 Minima successifs. 51
2.1.2 Réduction au sens de Minkowski. 51
2.1.3 Cellule de Voronoï. 53
2.2 Deux analyses de l'algorithme de Lagrange. 56
2.2.1 L'analyse globale de l'algorithme de Lagrange. 57
2.2.2 L'analyse locale de l'algorithme de Lagrange. 58
2.3 Une généralisation gloutonne de l'algorithme de Lagrange. 59
2.3.1 L'algorithme glouton de réduction. 59
2.3.2 Une description itérative de l'algorithme glouton de réduction. 61
2.3.3 G-réduction. 61
2.4 Le problème CVP en petite dimension. 63
2.5 L'approche globale. 65
2.5.1 Deux phases dans l'algorithme glouton récursif. 66
2.5.2 L'algorithme glouton avec une base plutôt orthogonale. . . 67
2.6 La preuve de la complexité quadratique. 68
2.7 L'approche locale. 70
2.7.1 Une analyse géométrique unifiée jusqu'en dimension 4. . . 71
2.7.2 Fin de l'analyse en dimension 4. 76
2.8 Résultats sur la géométrie des réseaux en petite dimension. 80
2.8.1 Les cellules de Voronoï pour des bases réduites au sens de Minkowski. 80

2.8.2 Les cellules de Voronoï pour des vecteurs ε-G-réduits. . . . 84
2.8.3 Le lemme du vide. 88
2.9 Qu'en est-il en plus grande dimension ? 94

Dans ce chapitre, nous étudions la réduction des réseaux Euclidiens en petite dimension. Il s'agit d'une stratégie tout à fait naturelle pour appréhender les problèmes qui surviennent quand la dimension augmente. Par exemple, certaines parties de l'analyse de l'algorithme LLL en virgule flottante du chapitre II-3 s'inspirent directement de la présente étude. En plus de cette tentative de compréhension des phénomènes mathématiques sous-jacents se rajoute une motivation plus pragmatique : en effet, les améliorations en petite dimension sont réutilisables aussi bien en théorie qu'en pratique en grande dimension grâce à l'algorithme de Schnorr de réduction Hermite-Korkine-Zolotarev par bloc [150], puisque ce dernier n'est qu'une succession de réductions en petites dimensions.

Nous analysons un algorithme de réduction glouton que l'on peut raisonnablement considérer comme l'algorithme le plus naturel de réduction de réseaux, car il s'agit d'une généralisation immédiate de l'algorithme de Lagrange [102] qui fonctionne en dimension 2. L'algorithme de Lagrange a aussi été décrit, ultérieurement, par Gauss [65], et est plus connu sous le nom d'algorithme de Gauss. Les résultats présentés sont de deux types. D'un point de vue mathématique, nous montrons que jusqu'en dimension 4 la sortie de l'algorithme est optimale : la base renvoyée atteint les minima successifs du réseau. Par contre, dès que la dimension du réseau est supérieure ou égale à 5, la base renvoyée peut ne même pas atteindre le premier minimum. Plus important : d'un point de vue calculatoire, nous montrons que jusqu'en dimension 4, la complexité (comptée en nombre d'opérations élémentaires) est quadratique, sans faire appel à l'arithmétique rapide des entiers [158][8]. Cela avait été déjà prouvé en dimension 3 par Semaev [159], qui utilise une méthode assez laborieuse, mais la question était ouverte en dimension 4. Nous proposons deux analyses différentes : une approche globale qui repose sur la compréhension de la géométrie de la base courante quand on n'a plus une décroissance significative du produit des longueurs, et une approche locale qui montre que toutes les $O(1)$ étapes le produit des longueurs des vecteurs courants décroît sensiblement. Nos analyses s'appuient fortement sur les propriétés géométriques des réseaux de petites dimensions, simplifient considérablement l'étude de Semaev en dimensions 2 et 3, et unifient les cas des dimensions 2 à 4. Bien que l'approche globale soit plus rapide, nous considérons aussi l'approche locale car elle donne une meilleure compréhension du comportement de l'algorithme.

Comme nous l'avons vu au chapitre I-1, il y a différentes notions de réduction d'un réseau Euclidien. On peut citer par exemple celles de Hermite, Minkowski, Korkine et Zolotarev (KZ), Venkov, des frères Lenstra et de Lovász (LLL). La plus intuitive est peut-être celle de Minkowski, et, en dimension inférieure ou égale à 4, elle est optimale

[8]Dans [57], Eisenbrand et Rote donnent un algorithme de réduction de complexité quasi-linéaire en dimension fixée (en faisant appel à de l'arithmétique rapide), mais la dépendance en la dimension semble très mauvaise. L'algorithme repose sur de très coûteuses recherches exhaustives.

puisqu'elle donne des vecteurs qui atteignent les quatre minima successifs du réseau. C'est donc à cette réduction que nous nous intéresserons.

Le célèbre algorithme de Lagrange, exprimé à l'origine en termes de formes quadratiques, calcule des bases Minkowski-réduites en dimension 2. Il généralise l'algorithme d'Euclide centré (le reste est choisi inférieur ou égal à la moitié du diviseur en valeur absolue), et a la même complexité : il termine en un temps quadratique en la taille de l'entrée. Cet algorithme a été généralisé en dimension 3 par Vallée en 1986 [180], mais la complexité devenait cubique, puis par Semaev en 2001 [159] qui obtient une complexité quadratique. De façon plus générale, Helfrich [75] a montré en 1985 comment utiliser l'algorithme LLL [112] pour calculer en temps cubique une base Minkowski-réduite en n'importe quelle dimension. Avec les résultats du chapitre II-3, le temps de calcul de l'algorithme de Helfrich devient quadratique à dimension fixée, mais n'est ni très pratique ni très révélateur en petite dimension puisqu'il repose essentiellement sur une recherche (très) exhaustive.

Ici, nous généralisons l'algorithme de Lagrange pour n'importe quelle dimension. Bien que l'algorithme de réduction obtenu soit le plus simple que l'on connaisse, son analyse gagne en difficulté de manière assez surprenante quand la dimension augmente. Semaev [159] a été le premier à montrer que sa complexité restait polynomiale en dimension 3, mais la question restait ouverte en dimension supérieure ou égale à 4. Nous montrons que jusqu'en dimension 4 l'algorithme glouton calcule une base Minkowski-réduite en temps quadratique en la taille de l'entrée sans utiliser d'arithmétique rapide. Pour arriver à ces résultats, nous avons deux approches différentes, toutes les deux très géométriques, qui généralisent chacune une analyse de l'algorithme de Lagrange. L'approche globale généralise jusqu'en dimension 4 les analyses de Vallée [181] et d'Akhavi [14]. Notons qu'elle a été utilisée par Akhavi pour borner le nombre d'itérations de boucles de l'algorithme LLL optimal en dimension quelconque. On peut résumer l'approche globale par la question suivante : « Que se passe-t-il quand l'algorithme arrête de faire décroître de façon significative le produit des longueurs à chaque itération de boucle ? » L'approche locale répond à la question duale : « Peut-on borner le nombre maximal d'itérations de boucles successives pour que le produit des longueurs décroisse de façon significative ? » En dimension 2, cette deuxième approche est celle de Semaev [159], qui est elle-même assez différente des précédentes analyses de l'algorithme de Lagrange [86, 96, 181]. En dimension 3, l'analyse de Semaev repose sur une étude exhaustive des comportements possibles de l'algorithme. Elle fait intervenir des calculs assez laborieux et rend difficile une généralisation en dimension supérieure. Nous remplaçons la plupart des arguments techniques par des considérations géométriques sur les réseaux de dimension 2. Cela rend possible la généralisation en dimension 4, par une étude précise de la géométrie des réseaux en dimension 3, bien que quelques difficultés viennent s'ajouter. Bien que l'approche locale soit plus laborieuse, elle n'est pas inintéressante : en effet, elle donne une compréhension plus précise de l'algorithme.

Dans la section 2.1 nous donnerons quelques définitions élémentaires qui nous seront indispensables pour la suite du chapitre. Ensuite nous décrirons l'algorithme de Lagrange, avec deux analyses différentes à la section 2.2. Cela nous permettra de décrire à la section 2.3 l'algorithme glouton qui généralise l'algorithme de Lagrange et que nous allons

étudier. La section 2.4 fournit un algorithme qui résout très efficacement CVP en petite dimension et qui est le centre de l'algorithme glouton. La section 2.5 est consacrée à l'approche globale, qui borne le nombre d'itérations de boucles de l'algorithme, et à la section 2.6 nous montrons la borne de complexité quadratique. À la section 2.7 nous donnons l'approche locale, qui est une alternative à la borne sur le nombre d'itérations de boucles qui aura été obtenue à la section 2.5 : ainsi, jusqu'en dimension 4, la borne de complexité quadratique peut être obtenue avec les sections 2.6 et 2.7 indépendamment de la section 2.5. Enfin, à la section 2.8 nous prouvons des résultats géométriques sur les réseaux de petite dimension. Ces résultats sont cruciaux pour prouver le lemme du vide qui est l'ingrédient essentiel de l'approche locale décrite à la section 2.7. Dans la section 2.9, nous dressons une liste des difficultés qui apparaissent lorsque l'on essaie de généraliser l'algorithme et les deux approches en dimension 5. La figure 2.1 donne l'enchaînement des preuves pour l'analyse de complexité.

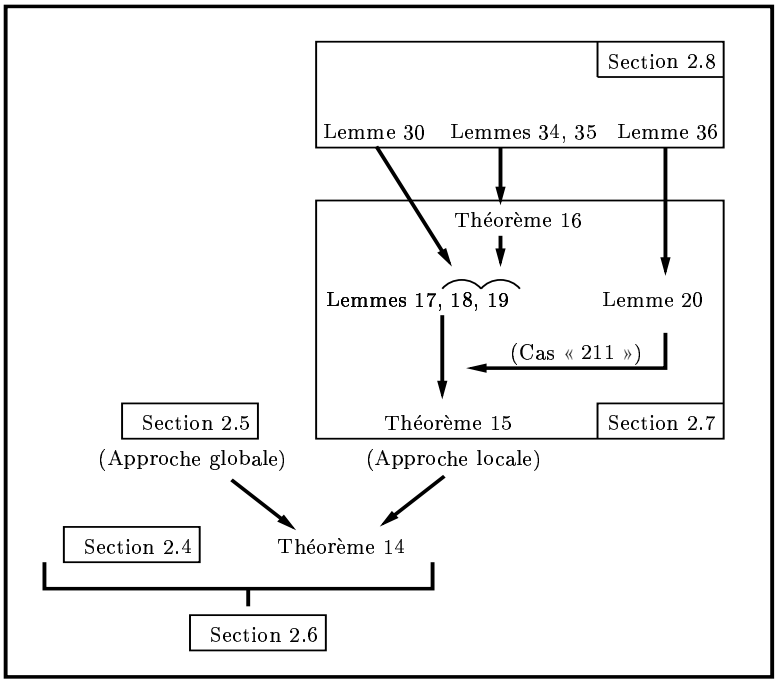


FIG. 2.1 – L'enchaînement des preuves pour l'analyse de complexité de l'algorithme glouton de la section 2.3.

Notation : Dans ce chapitre, quand la notation $[\mathbf{b}_1, \ldots, \mathbf{b}_d]$ est utilisée, elle signifie que les vecteurs $\mathbf{b}_1, \ldots, \mathbf{b}_d$ sont ordonnés par normes Euclidiennes croissantes (si des vecteurs sont de même norme, alors la notation $[\ldots]$ peut être utilisée pour plusieurs permutations de $\mathbf{b}_1, \ldots, \mathbf{b}_d$). On dira dans ce cas que la base est ordonnée.

2.1 Préliminaires.

Nous rappelons ici quelques définitions et résultats de base sur les minima d'un réseau, sur les réductions aux sens de Minkowski et de Hermite-Korkine-Zolotarev, et sur la cellule

de Voronoï d'un réseau.

2.1.1 Minima successifs.

Soit L un réseau Euclidien de dimension d inclus dans $\mathbb{R}^n$. Nous rappelons que le i-ème minimum $\lambda_i(L)$, pour $i \in [|1, d|]$, est le rayon de la plus petite boule fermée centrée en $\mathbf{0}$ et qui contient au moins i vecteurs du réseau linéairement indépendants. Quel que soit le réseau considéré, il existe toujours des vecteurs lui appartenant qui sont linéairement indépendants et qui atteignent les minima successifs. Il peut paraître surprenant que, lorsque $d \geqslant 4$, des vecteurs atteignant ces minima ne forment pas nécessairement une base, et que, lorsque $d \geqslant 5$, il existe des réseaux pour lesquels aucune base n'atteint les minima simultanément. En effet, il suffit de considérer le réseau engendré par les colonnes de la matrice suivante [93] :

$$(\mathbf{b}_1, \ldots, \mathbf{b}_5) = \begin{pmatrix} 2 & 0 & 0 & 0 & 1 \\ 0 & 2 & 0 & 0 & 1 \\ 0 & 0 & 2 & 0 & 1 \\ 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \ \mathbf{b} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 2 \end{pmatrix}.$$

Soient $\mathbf{b}_i$ la colonne i de cette matrice, et $\mathbf{b} = 2\mathbf{b}_5 - \mathbf{b}_1 - \mathbf{b}_2 - \mathbf{b}_3 - \mathbf{b}_4$. Les vecteurs $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_4, \mathbf{b}$ sont linéairement indépendants, la norme de $\mathbf{b}$ est 2, et il est facile de voir que n'importe quel vecteur non nul est de norme au moins 2. Ainsi, $\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = \lambda_5 = 2$. Cependant, dans ce réseau, aucune base n'est composée de vecteurs de normes toutes égales à 2 : en effet, puisque le réseau est entier, tous les carrés des normes des vecteurs seraient paires, ce qui n'est bien entendu pas le cas pour $\mathbf{b}_5$.

2.1.2 Réduction au sens de Minkowski.

Nous nous intéressons dans ce chapitre à la réduction au sens de Minkowski, dont nous rappelons ici la définition :

Définition 19 (Réduction de Minkowski). *Une base $[\mathbf{b}_1, \ldots, \mathbf{b}_d]$ d'un réseau L est Minkowski réduite si pour tout $i \in [|1, d|]$, le vecteur $\mathbf{b}_i$ est de norme minimale parmi ceux tels que la famille $(\mathbf{b}_1, \ldots, \mathbf{b}_i)$ peut être complétée en une base de L (on parle d'un ensemble primitif de vecteurs).*

La condition de primitivité des vecteurs peut être rendue plus explicite :

Lemme 9. *Une base $[\mathbf{b}_1, \ldots, \mathbf{b}_d]$ d'un réseau L est Minkowski-réduite si et seulement si pour tout $i \in [|1, d|]$ et pour tous les entiers $x_1, \ldots, x_d$ tels que $x_i, \ldots, x_d$ sont premiers entre eux dans leur ensemble, on a :*

$$\|x_1\mathbf{b}_1 + \ldots + x_d\mathbf{b}_d\| \geqslant \|\mathbf{b}_i\|.$$

Il semble donc que même à dimension fixée il y a un nombre infini de conditions à vérifier pour s'assurer qu'une base est Minkowski-réduite. Ce n'est pas le cas : à dimension

fixée, il existe un sous-ensemble fini de ces conditions qui suffit à s'assurer qu'une base est Minkowski-réduite. Ce résultat est connu sous le nom de second théorème de finitude, et le lecteur pourra en trouver une preuve dans [164]. Par contre, la borne obtenue dans le cas général n'est pas du tout pratique. On appelle *conditions de Minkowski* un ensemble minimal de telles conditions. Jusqu'en dimension 6, ces conditions ont été calculées par Tammela [176]. On peut donc décider rapidement si une base est réduite au sens de Minkowski en les testant.

Théorème 8 (Conditions de Minkowski [176]). *Soit $d \leqslant 6$. Une base $[\mathbf{b}_1, \ldots, \mathbf{b}_d]$ d'un réseau L est Minkowski-réduite si et seulement si pour tout $i \in [|1, d|]$ et pour tous les entiers $x_1, \ldots, x_d$ vérifiant les deux conditions ci-dessous, nous avons :*

$$\|x_1\mathbf{b}_1 + \ldots + x_d\mathbf{b}_d\| \geqslant \|\mathbf{b}_i\|.$$

1. *Les entiers $x_i, \ldots, x_d$ sont premiers entre eux dans leur ensemble.*
2. *Pour une certaine permutation $\sigma \in \mathcal{S}_d$, $(|x_{\sigma(1)}|, \ldots, |x_{\sigma(d)}|)$ apparaît dans la liste ci-dessous (où les blancs peuvent compter comme des zéros).*

2	*1*	*1*				
3	*1*	*1*	*1*			
4	*1*	*1*	*1*	*1*		
5	*1*	*1*	*1*	*1*	*1*	
	1	*1*	*1*	*1*	*2*	
6	*1*	*1*	*1*	*1*	*1*	*1*
	1	*1*	*1*	*1*	*1*	*2*
	1	*1*	*1*	*1*	*2*	*2*
	1	*1*	*1*	*1*	*2*	*3*

De plus, cette liste est minimale, ce qui signifie que si l'une des lignes du tableau est écartée, alors une base peut satisfaire les autres sans pour autant être Minkowski-réduite.

Par exemple, en dimension 3, la base $[\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3]$ est réduite si et seulement si on a les inégalités suivantes :

$$\begin{array}{rcl@{\qquad}rcl}
\|\mathbf{b}_1 + \mathbf{b}_2\| & \geqslant & \|\mathbf{b}_2\| & \|\mathbf{b}_1 - \mathbf{b}_2\| & \geqslant & \|\mathbf{b}_2\| \\
\|\mathbf{b}_1 + \mathbf{b}_3\| & \geqslant & \|\mathbf{b}_3\| & \|\mathbf{b}_1 - \mathbf{b}_3\| & \geqslant & \|\mathbf{b}_3\| \\
\|\mathbf{b}_2 + \mathbf{b}_3\| & \geqslant & \|\mathbf{b}_3\| & \|\mathbf{b}_2 - \mathbf{b}_3\| & \geqslant & \|\mathbf{b}_3\| \\
\|\mathbf{b}_1 + \mathbf{b}_2 + \mathbf{b}_3\| & \geqslant & \|\mathbf{b}_3\| & \|\mathbf{b}_1 + \mathbf{b}_2 - \mathbf{b}_3\| & \geqslant & \|\mathbf{b}_3\| \\
\|\mathbf{b}_1 - \mathbf{b}_2 + \mathbf{b}_3\| & \geqslant & \|\mathbf{b}_3\| & \|\mathbf{b}_1 - \mathbf{b}_2 - \mathbf{b}_3\| & \geqslant & \|\mathbf{b}_3\|
\end{array}$$

Une base d'un réseau de dimension d qui atteint les d minima successifs est nécessairement Minkowski-réduite, mais la réciproque est fausse : une base Minkowski-réduite n'atteint pas forcément les minima successifs. Par contre, elle atteint toujours les quatre premiers (voir [184]) : si $[\mathbf{b}_1, \ldots, \mathbf{b}_d]$ est une base Minkowski-réduite de L, alors pour tout $1 \leqslant i \leqslant \min(d, 4)$, on a $\|\mathbf{b}_i\| = \lambda_i(L)$. Ainsi, une base Minkowski-réduite est optimale

dans un sens très naturel jusqu'en dimension 4. Il est classique que le défaut d'orthogonalité d'une base Minkowski-réduite peut être borné par une constante ne dépendant que de la dimension du réseau (voir [184] par exemple).

Un autre type de réduction extrêmement classique est celle de Hermite-Korkine-Zolotarev (HKZ) [93, 77]. Nous avons déjà défini cette notion de réduction au chapitre I-1, à la page 14. Nous donnons une définition équivalente, dont la description est plus itérative.

Définition 20 (HKZ-réduction). *Une base $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ d'un réseau L est dite réduite au sens de Hermite-Korkine-Zolotarev (HKZ) si $\|\mathbf{b}_1\| = \lambda_1(L)$, si pour tout $i \in [|2, d|]$, le vecteur $\mathbf{b}_i$ est tel que $\|\mathbf{b}_i^*\|$ est minimale (sous la contrainte que $\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}$ sont déjà fixés) et si la base est propre (ses coefficients de Gram-Schmidt $\mu_{i,j}$ pour $i > j$ sont inférieurs ou égaux à $1/2$ en valeur absolue).*

Les réductions HKZ et de Minkowski dont équivalentes en dimension 2, mais dès la dimension 3, il y a des réseaux pour lesquels aucune base Minkowski-réduite n'est HKZ-réduite :

Lemme 10. *Soient $\mathbf{b}_1 = (100, 0, 0), \mathbf{b}_2 = (49, 100, 0)$ et $\mathbf{b}_3 = (0, 62, 100)$. Alors le réseau $L(\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3)$ n'a pas de base à la fois réduite aux sens de Minkowski et de Hermite-Korkine-Zolotarev.*

Démonstration. Tout d'abord, la base $(\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3)$ n'est pas HKZ-réduite car $|\mu_{3,2}| = \frac{62}{100} > 1/2$. Cependant elle est Minkowski-réduite (il suffit de vérifier les conditions de Minkowski pour s'en assurer). Ainsi les vecteurs $\mathbf{b}_1, \mathbf{b}_2$ et $\mathbf{b}_3$ atteignent les trois minima. Plus précisément, $\pm\mathbf{b}_1$ sont les deux seuls vecteurs atteignant le premier minimum, $\pm\mathbf{b}_2$ sont les deux seuls vecteurs atteignant le second, et $\pm\mathbf{b}_3$ sont les deux seuls atteignant le troisième.

Si $L(\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3)$ admettait une base réduite à la fois au sens de Minkowski et de Hermite-Korkine-Zolotarev, cette base atteindrait les trois minima. On en déduit donc qu'elle serait de la forme $(\pm\mathbf{b}_1, \pm\mathbf{b}_2, \pm\mathbf{b}_3)$, ce qui contredit le fait que $(\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3)$ n'est pas HKZ-réduite. □

2.1.3 Cellule de Voronoï.

Les rappels de ce paragraphe ne sont nécessaires que pour l'approche locale décrite aux sections 2.7 et 2.8. On aura alors besoin de déterminer, étant donnés une base réduite (dans un sens qui sera précisé alors) et un vecteur du réseau engendré par cette base, quelles coordonnées peut avoir ce vecteur s'il est proche de la cellule de Voronoï de ce réseau.

La cellule de Voronoï [183] d'un réseau est l'ensemble des vecteurs de l'espace qui sont plus proches de $\mathbf{0}$ que de n'importe quel autre vecteur du réseau. Autrement dit :

Définition 21 (Cellule de Voronoï). *Soit L un réseau. La cellule de Voronoï $\mathrm{Vor}(L)$ est définie par :*

$$\mathrm{Vor}(L) = \{\mathbf{x} \in \mathrm{E}(L), \forall \mathbf{v} \in L, \|\mathbf{x} - \mathbf{v}\| \geqslant \|\mathbf{x}\|\},$$

où $\mathrm{E}(L)$ *est le sous-espace vectoriel engendré par* L.

De manière équivalente, $\mathrm{Vor}(L) = \{\mathbf{x} \in \mathrm{E}(L),\ \forall \mathbf{v} \in L, 2|\langle \mathbf{v}, \mathbf{x}\rangle| \leqslant \|\mathbf{v}\|^2\}$. La cellule de Voronoï est un polytope fini dont les translatés par les vecteurs du réseau pavent l'espace vectoriel engendré par L. Par abus de notation, $\mathrm{Vor}(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ désignera la cellule de Voronoï du réseau engendré par les vecteurs $\mathbf{b}_1, \ldots, \mathbf{b}_d$, qui ne sont pas forcément linéairement indépendants.

Définition 22. *Soient* L *un réseau de dimension* d *et* $\mathbf{v} \in L$. *Le vecteur* $\mathbf{v}$ *est un vecteur de Voronoï si* $\mathbf{v}/2$ *appartient à* $\mathrm{Vor}(L)$. *De plus,* $\mathbf{v}$ *est un vecteur de Voronoï strict si* $\mathbf{v}/2$ *appartient à l'intérieur d'une face de dimension* $d-1$ *de* $\mathrm{Vor}(L)$.

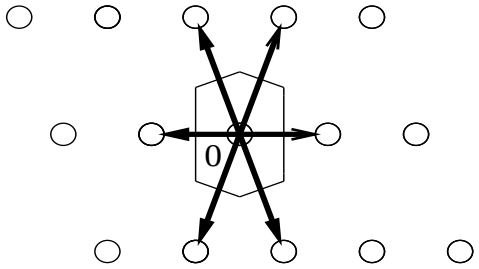


FIG. 2.2 – Une cellule et les vecteurs de Voronoï d'un réseau planaire.

Les deux définitions précédentes sont illustrées à la figure 2.2. Remarquons que si un vecteur est un vecteur de Voronoï, alors $\mathbf{v}/2$ est en fait sur la frontière de $\mathrm{Vor}(L)$. De plus, comme l'illustre la figure 2.3, un vecteur de Voronoï n'est pas forcément un vecteur de Voronoï strict. Il est classique que les vecteurs de Voronoï correspondent aux minima des classes d'équivalences du groupe quotient $L/2L$: un vecteur de Voronoï est strict si la classe d'équivalence correspondante n'admet que deux minima.

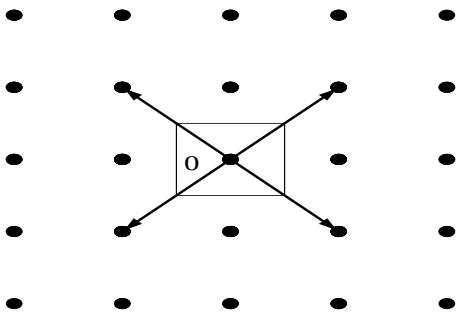


FIG. 2.3 – Des vecteurs de Voronoï non stricts.

Définition 23 (Coordonnée de Voronoï possible). *Soit* $(x_1, \ldots, x_d) \in \mathbb{Z}^d$. *On dit que* $(x_1, \ldots, x_d)$ *est une coordonnée de Voronoï possible (respectivement coordonnée de Voronoï stricte possible) s'il existe une base ordonnée* $[\mathbf{b}_1, \ldots, \mathbf{b}_d]$ *réduite au sens de Minkowski telle que* $x_1\mathbf{b}_1 + \cdots + x_d\mathbf{b}_d$ *est un vecteur de Voronoï (respectivement vecteur de Voronoï strict) du réseau* $L(\mathbf{b}_1, \ldots, \mathbf{b}_d)$.

Dans sa thèse, Tammela [176] a dressé une liste exhaustive des coordonnées de Voronoï strictes possibles jusqu'en dimension 6. Nous ne pourrons pas utiliser directement le résultat de Tammela car il concerne les coordonnées de Voronoï strictes alors que nous aurons besoin de considérer les coordonnées de Voronoï au sens large. Nous prouverons que ce sont les mêmes en dimension 2 (lemme 23, page 80), et qu'il faut rajouter la ligne « 1 1 2 » en dimension 3 (lemme 24, page 81).

Théorème 9. [9] *Soit $d \leqslant 6$. Les coordonnées de Voronoï strictes possibles en dimension d sont celles de la dimension $d-1$, auxquelles il faut ajouter les d-uplets $(x_1, \ldots, x_d)$ tels qu'il existe une permutation $\sigma \in \mathcal{S}_d$ pour laquelle $(|x_{\sigma(1)}|, \ldots, |x_{\sigma(d)}|)$ apparaît dans le d-ième bloc de la table ci-dessous :*

1	1				
2	1	1			
3	1	1	1		
4	1	1	1	1	
	1	1	1	2	
5	1	1	1	1	1
	1	1	1	1	2
	1	1	1	2	2
	1	1	1	2	3

6	1	1	1	1	1	1
	1	1	1	1	1	2
	1	1	1	1	1	3
	1	1	1	1	2	2
	1	1	1	1	2	3
	1	1	1	2	2	2
	1	1	1	2	2	3
	1	1	1	2	2	4
	1	1	2	2	2	3
	1	1	1	2	3	3
	1	1	1	2	3	4
	1	1	2	2	3	4
	1	2	2	2	3	3

Nous nous intéresserons aussi à des coordonnées de Voronoï vis-à-vis d'autres réductions que celle de Minkowski. Cela sera alors précisé explicitement.

Définition 24 (Rayon de recouvrement). *Le rayon de recouvrement $\rho(L)$ d'un réseau L est la moitié du diamètre de la cellule de Voronoï.*

On peut remarquer que le problème CVP consiste, à partir d'un vecteur cible $\mathbf{t}$ et d'un réseau donné par une base, à trouver un vecteur $\mathbf{v} \in L$ tel que le vecteur $\mathbf{t} - \mathbf{v}$ soit dans la cellule de Voronoï du réseau.

Nous avons vu que si $[\mathbf{b}_1, \ldots, \mathbf{b}_d]$ est une base Minkowski-réduite avec $d \leqslant 6$, les coordonnées de Voronoï sont bornées (on a même une liste exhaustive des coordonnées possibles). Delone et Sandakova [53, 173] ont montré un résultat beaucoup plus fort : si la base est réduite, alors les coordonnées (réelles) de n'importe quel point de la cellule de Voronoï vis-à-vis des vecteurs de la base sont bornées. Ce résultat est valable quelle que soit la dimension et pour de nombreux types de réduction, mais l'énoncé suivant nous suffira :

Théorème 10. *Les énoncés suivants sont valides :*

[9] Bien qu'ils aient des formes similaires, il ne semble pas a priori qu'il y ait un rapport entre ce théorème et le théorème 8.

1. *Soient* $[\mathbf{b}_1, \mathbf{b}_2]$ *une base Minkowski-réduite et* $\mathbf{u} \in \mathrm{Vor}(\mathbf{b}_1, \mathbf{b}_2)$. *Écrivons* $\mathbf{u} = x\mathbf{b}_1 + y\mathbf{b}_2$. *Alors* $|x| < 3/4$ *et* $|y| \leqslant 2/3$.
2. *Soient* $[\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3]$ *une base Minkowski-réduite et* $\mathbf{u} \in \mathrm{Vor}(\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3)$. *Écrivons* $\mathbf{u} = x\mathbf{b}_1 + y\mathbf{b}_2 + z\mathbf{b}_3$. *Alors* $|x| < 3/2$, $|y| \leqslant 4/3$ *et* $|z| \leqslant 1$.

2.2 Deux analyses de l'algorithme de Lagrange.

L'algorithme de Lagrange, décrit à la figure 2.4, peut être vu comme une généralisation

Entrée : Une base $[\mathbf{u}, \mathbf{v}]$ avec sa matrice de Gram.
Sortie : Une base de $L(\mathbf{u}, \mathbf{v})$, avec sa matrice de Gram.

1. Répéter
2. $\quad \mathbf{r} := \mathbf{v} - x\mathbf{u}$ avec $x := \left\lfloor \frac{\langle \mathbf{u}, \mathbf{v} \rangle}{\|\mathbf{u}\|^2} \right\rceil$,
3. $\quad \mathbf{v} := \mathbf{u}$,
4. $\quad \mathbf{u} := \mathbf{r}$,
5. $\quad$ Mettre à jour la matrice de Gram de $(\mathbf{u}, \mathbf{v})$,
6. Jusqu'à ce que $\|\mathbf{u}\| \geq \|\mathbf{v}\|$.
7. Renvoyer $[\mathbf{v}, \mathbf{u}]$ et sa matrice de Gram.

FIG. 2.4 – L'algorithme de Lagrange.

bi-dimensionnelle de l'algorithme d'Euclide centré [181]. Un exemple d'exécution est donné à la figure 2.5.

À la seconde étape de chaque itération de boucle, le vecteur $\mathbf{u}$ est plus court que le vecteur $\mathbf{v}$, et on cherche donc à raccourcir $\mathbf{v}$, tout en préservant la propriété que $[\mathbf{u}, \mathbf{v}]$ est une base du réseau. Pour faire cela, on utilise la transformation unimodulaire non triviale la plus simple : on soustrait à $\mathbf{v}$ un multiple entier $x\mathbf{u}$ de $\mathbf{u}$. Le choix optimal (pour faire décroître la norme de $\mathbf{v}$ le plus possible à cette étape) correspond au cas où $x\mathbf{u}$ est le plus proche possible de $\mathbf{v}$, ce qui revient à résoudre un problème de CVP où la base est $[\mathbf{u}]$ et la cible est $\mathbf{v}$. On trouve ainsi $x := \left\lfloor \frac{\langle \mathbf{u}, \mathbf{v} \rangle}{\|\mathbf{u}\|^2} \right\rceil$. Dans cette formule, les valeurs de $\langle \mathbf{u}, \mathbf{v} \rangle$ et de $\|\mathbf{u}\|^2$ sont obtenues par simple lecture de la matrice de Gram $G(\mathbf{u}, \mathbf{v})$, qui est elle-même mise à jour à l'étape 5 de chaque itération de boucle. Le résultat suivant donne la complexité de l'algorithme de Lagrange :

Théorème 11. *Étant donnée en entrée une base* $[\mathbf{u}, \mathbf{v}]$ *d'un réseau* L, *l'algorithme de Lagrange renvoie une base Minkowski-réduite de* L *en temps* $O(\log \|\mathbf{v}\| \cdot [1 + \log \|\mathbf{v}\| - \log \lambda_1(L)])$.

Ce résultat n'est pas trivial à prouver, et en particulier, il n'est même pas clair a priori que la base renvoyée est bien Minkowski-réduite. La correction de l'algorithme vient en fait directement des conditions de Minkowski en dimension 2, c'est-à-dire du théorème 8. Nous donnons ici deux analyses différentes de l'algorithme de Lagrange : une analyse globale et une analyse locale. Nous généraliserons ces deux analyses en dimension 4.

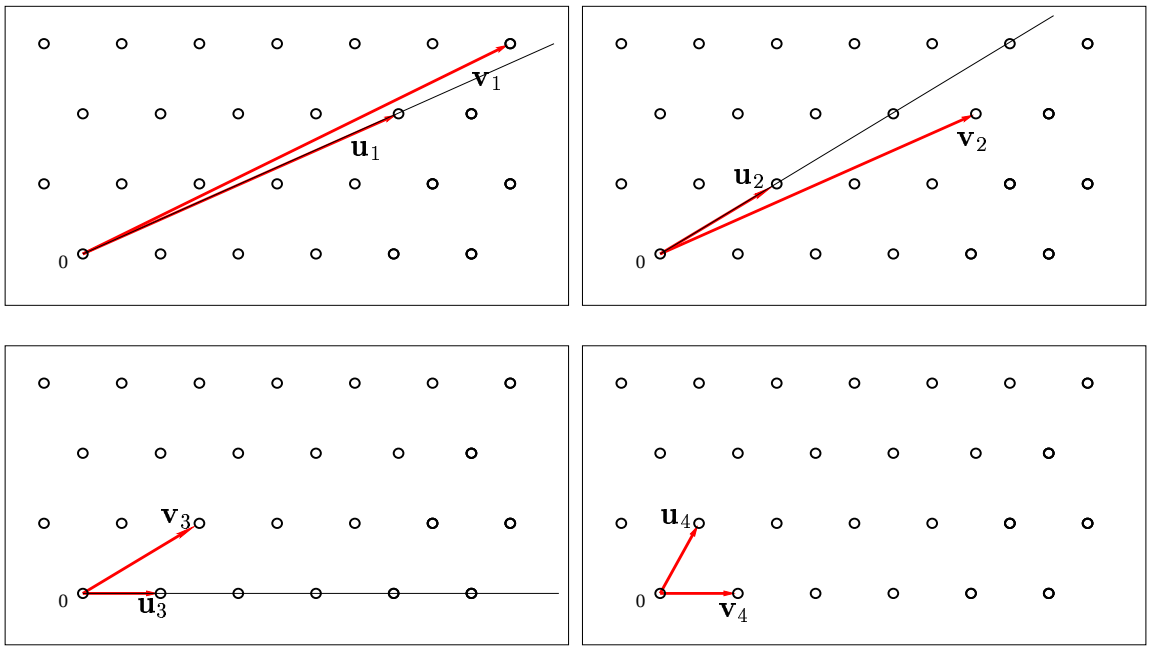


FIG. 2.5 – Un exemple d'exécution de l'algorithme de Lagrange.

2.2.1 L'analyse globale de l'algorithme de Lagrange.

Il s'agit de la stratégie adoptée dans [181] et dans [14]. Deux phases de l'exécution de l'algorithme sont dissociées : dans la première phase, chaque vecteur créé est nettement plus court que celui qu'il remplace, plus précisément, plus court d'un facteur supérieur à $\sqrt{1+\eta}$ pour un certain $\eta > 0$ qui sera choisi ultérieurement ; la seconde phase commence lorsque la propriété précédente n'est pas vérifiée (même si elle est vérifiée de nouveau par la suite), et nous montrons que cette phase contient $O(1)$ itérations de boucle.

Pour l'analyse, nous avons besoin de définir l'algorithme η-Lagrange, dont nous nous servirons exclusivement pour analyser l'algorithme de Lagrange (à aucun moment cet algorithme ne sera exécuté). L'algorithme η-Lagrange est décrit à la figure 2.6. La première phase de l'exécution de l'algorithme de Lagrange est exactement identique à l'exécution de l'algorithme η-Lagrange, alors que la seconde phase de l'exécution correspond à l'exécution de l'algorithme de Lagrange sur une entrée qui est déjà η-Lagrange réduite, c'est-à-dire une base laissée invariante par l'algorithme η-Lagrange.

Le nombre d'itérations de boucle de la première phase (c'est-à-dire l'exécution de l'algorithme η-Lagrange) se borne très simplement par $O(1+\log\|\mathbf{v}\|)$, car le produit des longueurs des vecteurs de la base décroît d'un facteur au moins $\sqrt{1+\eta}$ à chaque itération de boucle. Nous bornons maintenant la longueur de la deuxième phase, c'est-à-dire le nombre d'itérations de boucle de l'algorithme de Lagrange quand on lui donne en entrée une base de sortie de l'algorithme η-Lagrange. Soit $[\mathbf{u}, \mathbf{v}]$ la base courante à la fin de la première phase, et soit $\mathbf{v}' = \mathbf{v} - x\mathbf{u}$ où $x \in \mathbb{Z}$ est choisi de façon à minimiser la longueur de $\mathbf{v} - x\mathbf{u}$. On a :

$$\|\mathbf{v}'\| \leqslant \|\mathbf{v}\| \leqslant \sqrt{1+\eta}\|\mathbf{v}'\|.$$

Entrée : Une base $[\mathbf{u}, \mathbf{v}]$.
Sortie : Une base du réseau $L(\mathbf{u}, \mathbf{v})$.

1. Répéter
2. $\quad \mathbf{r}{:=}\mathbf{v} - x\mathbf{u}$ avec $x{:=}\left\lfloor \frac{\langle \mathbf{u},\mathbf{v}\rangle}{\|\mathbf{u}\|^2} \right\rceil$,
3. $\quad$ Si $(1+\eta)\|\mathbf{r}\|^2 \leqslant \|\mathbf{v}\|^2$ alors
4. $\quad\quad \mathbf{v}{:=}\mathbf{u}$,
5. $\quad\quad \mathbf{u}{:=}\mathbf{r}$,
6. $\quad$ Sinon, aller à l'étape 7.
7. Renvoyer $[\mathbf{u}, \mathbf{v}]$.

FIG. 2.6 – L'algorithme η-Lagrange.

Nous affirmons que si $\eta > 0$ est suffisamment petit, alors l'algorithme de Lagrange termine en $O(1)$ itérations de boucle. Écrivons $\mathbf{v} = \mathbf{v}^* + \mathbf{v}^-$, où $\mathbf{v}^- = \frac{\langle \mathbf{u},\mathbf{v}\rangle}{\|\mathbf{u}\|^2}\mathbf{u}$. Puisque $\mathbf{v}' - \mathbf{v}^* = y\mathbf{u}$ avec $|y| \leqslant 1/2$, le théorème de Pythagore donne que $\|\mathbf{v}'\|^2 \leqslant \|\mathbf{v}^*\|^2 + \|\mathbf{u}\|^2/4$, ce qui implique que :

$$\|\mathbf{v}^*\|^2 \geqslant \left(\frac{1}{1+\eta} - \frac{1}{4}\right)\|\mathbf{v}\|^2,$$

car $\|\mathbf{u}\| \leqslant \|\mathbf{v}\|$. Soit $\mathbf{b} = x_1\mathbf{u} + x_2\mathbf{v}$ un vecteur apparaissant dans la deuxième phase de l'algorithme. On a $\|\mathbf{v}\|^2 \geqslant \|\mathbf{b}\|^2 \geqslant x_2^2\|\mathbf{v}^*\|^2$, et donc $|x_2| \leqslant 1$ si $\eta > 0$ est choisi suffisamment petit. À cause du choix glouton pour x, il ne peut y avoir plus de quatre itérations de boucle dans la deuxième phase : dans toutes sauf une, un vecteur strictement plus court que les précédents est créé. Dans chacune de ces itérations, un élément le plus court de la forme $x\mathbf{v} + y\mathbf{u}$ est choisi, avec $x \in \{-1, 0, 1\}$ et y déterminé de façon unique pour chaque x possible.

Il reste maintenant à estimer le coût de chaque étape 2, c'est-à-dire le coût du calcul de x. Puisque $\frac{|\langle \mathbf{u},\mathbf{v}\rangle|}{\|\mathbf{u}\|^2} \leqslant \frac{\|\mathbf{v}\|}{\|\mathbf{u}\|}$, le coût de l'étape 2 est borné par $O(\log\|\mathbf{v}\| \cdot [1 + \log\|\mathbf{v}\| - \log\|\mathbf{u}\|])$. Soient $\mathbf{u}_i$ et $\mathbf{v}_i$ les valeurs de $\mathbf{u}$ et $\mathbf{v}$ au début de la i-ème itération. On a $\mathbf{v}_{i+1} = \mathbf{u}_i$, et on en déduit que le coût total de l'algorithme de Lagrange est borné par :

$$\begin{aligned} O\Big(\sum_{i=1}^{\tau} \log\|\mathbf{v}_i\| \cdot [1 + \log\|\mathbf{v}_i\| - \log\|\mathbf{u}_i\|]\Big) &= O\Big(\log\|\mathbf{v}\| \cdot \sum_{i=1}^{\tau} [1 + \log\|\mathbf{v}_i\| - \log\|\mathbf{v}_{i+1}\|]\Big) \\ &= O(\log\|\mathbf{v}\| \cdot [\tau + \log\|\mathbf{v}\| - \log\lambda_1(L)]), \end{aligned}$$

où $\tau = O(\log\|\mathbf{v}\| - \log\lambda_1)$ est le nombre d'itérations de boucle. Ceci conclut la preuve du théorème 11.

2.2.2 L'analyse locale de l'algorithme de Lagrange.

Vis-à-vis des analyses existantes de l'algorithme de Lagrange, notre approche locale est proche de celle de Semaev [159], qui est elle-même assez différente de [96, 181, 86, 16]. L'analyse n'est pas optimale (contrairement à celle de [181] par exemple, qui donne les

pires cas de l'algorithme), mais elle peut être généralisée jusqu'en dimension 4. Ceci est moins bon que l'approche globale, mais cela apporte une compréhension plus précise de l'algorithme.

Considérons la valeur de x à l'étape 2 :
- Si $x = 0$, alors nous sommes à la dernière itération de boucle.
- Si $|x| = 1$, deux cas sont possibles :
  - Si $\|\mathbf{v} - x\mathbf{u}\| \geqslant \|\mathbf{u}\|$, alors nous sommes à la dernière itération de boucle.
  - Sinon, puisque $|x| = 1$, on a $\|\mathbf{u} - x\mathbf{v}\| < \|\mathbf{u}\|$, ce qui signifie que $\mathbf{u}$ peut être raccourci à l'aide de $\mathbf{v}$. Cela ne peut arriver éventuellement qu'à la première itération de boucle, du fait de la stratégie gloutonne de l'algorithme : le vecteur $\mathbf{u}$ est l'ancien vecteur $\mathbf{v}$.
- Sinon, $|x| \geqslant 2$, ce qui implique que $x\mathbf{u}$ n'est pas un vecteur de Voronoï du réseau engendré par $\mathbf{u}$. Intuitivement, cela signifie que $x\mathbf{u}$ est très éloigné de $\mathrm{Vor}(\mathbf{u})$, de telle sorte que $\mathbf{v} - x\mathbf{u}$ est considérablement plus petit que $\mathbf{v}$. Plus précisément, si $\mathbf{v}^*$ est la composante du vecteur $\mathbf{v}$ orthogonale au vecteur $\mathbf{u}$, on a :

$$\|\mathbf{v}\|^2 \geqslant \left(\frac{3}{2}\right)^2 \|\mathbf{u}\|^2 + \|\mathbf{v}^*\|^2 \geqslant 2\|\mathbf{u}\|^2 + \|\mathbf{v} - x\mathbf{u}\|^2 > 3\|\mathbf{v} - x\mathbf{u}\|^2,$$

  où dans la dernière inégalité nous avons supposé que nous n'étions pas à la dernière itération de boucle.

Cela met en évidence que le produit des longueurs des vecteurs de la base décroît d'un facteur au moins $\sqrt{3}$ à chaque itération de boucle sauf éventuellement la première et la dernière. Ainsi, le nombre τ d'itérations de boucle est borné par : $\tau = O(1 + \log\|\mathbf{v}\| - \log\lambda_1(L))$. La fin de l'analyse de complexité est similaire à celle de l'approche globale : la borne linéaire sur le nombre d'itérations de boucle est suffisante pour faire fonctionner le décompte total des opérations élémentaires.

2.3 Une généralisation gloutonne de l'algorithme de Lagrange.

Dans la section précédente, nous avons décrit l'algorithme de Lagrange comme un algorithme glouton reposant sur le problème CVP en dimension 1. Ceci suggère une généralisation naturelle en dimension arbitraire d, que nous appelons l'algorithme glouton de réduction. Nous étudions les bases renvoyées par l'algorithme en définissant un nouveau type de réduction et en comparant celle-ci à la réduction au sens de Minkowski.

2.3.1 L'algorithme glouton de réduction.

L'algorithme de Lagrange suggère une généralisation en dimension supérieure : elle est décrite à la figure 2.7. Cette généralisation utilise de la réduction et un algorithme qui résout le problème CVP en dimension $d - 1$, pour réduire des réseaux en dimension d.

Algorithme: **Glouton**$(\mathbf{b}_1, \ldots, \mathbf{b}_d)$.
Entrée : Une base $[\mathbf{b}_1, \ldots, \mathbf{b}_d]$ avec sa matrice de Gram.
Sortie : Une base ordonnée de $L(\mathbf{b}_1, \ldots, \mathbf{b}_d)$, avec sa matrice de Gram.

1. Si $d = 1$, renvoyer $\mathbf{b}_1$.
2. Répéter
3. Trier $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ par longueurs croissantes, et mettre à jour la matrice de Gram,
4. $[\mathbf{b}_1, \ldots, \mathbf{b}_{d-1}]$:=**Glouton**$(\mathbf{b}_1, \ldots, \mathbf{b}_{d-1})$,
5. Calculer un vecteur $\mathbf{c} \in L(\mathbf{b}_1, \ldots, \mathbf{b}_{d-1})$ le plus proche de $\mathbf{b}_d$,
6. $\mathbf{b}_d$:=$\mathbf{b}_d - \mathbf{c}$, mettre à jour la matrice de Gram,
7. Jusqu'à ce que $\|\mathbf{b}_d\| \geqslant \|\mathbf{b}_{d-1}\|$.
8. Renvoyer $[\mathbf{b}_1, \ldots, \mathbf{b}_d]$ et sa matrice de Gram.

FIG. 2.7 – L'algorithme glouton de réduction en dimension d.

Un certain nombre de remarques simples doivent être faites sur cet algorithme. Tout d'abord, si la matrice de Gram n'est pas donnée, on peut la calculer en temps $O(\log^2 \|\mathbf{b}_d\|)$ pour une dimension d fixée. L'algorithme met à jour la matrice de Gram à chaque fois que la base est modifiée. L'étape 3 est simple : si l'on est à la première itération de boucle, la base est déjà ordonnée, sinon $[\mathbf{b}_1, \ldots, \mathbf{b}_{d-1}]$ est déjà ordonnée, et seul $\mathbf{b}_d$ doit être inséré à la bonne position. À l'étape 4, l'algorithme s'appelle récursivement en dimension $d-1$: $G(\mathbf{b}_1, \ldots, \mathbf{b}_{d-1})$ n'a pas besoin d'être calculée puisque c'est une sous-matrice de $G(\mathbf{b}_1, \ldots, \mathbf{b}_d)$, qui est déjà connue. Pour le moment, nous n'expliquons pas comment l'étape 5 est effectuée : bien sûr, on pourrait utiliser des algorithmes standards qui résolvent le problème CVP, comme celui de Kannan [88], mais ils ne suffisent pas à obtenir la complexité quadratique annoncée. Nous décrirons une procédure efficace qui résout le problème CVP en dimension fixée à la section 2.4. Remarquons qu'en dimension 2, l'algorithme de réduction glouton est exactement l'algorithme de Lagrange. D'un point de vue géométrique, le but des étapes 5 et 6 est de faire en sorte que la projection orthogonale de $\mathbf{b}_d$ sur le réseau engendré par $[\mathbf{b}_1, \ldots, \mathbf{b}_{d-1}]$ appartienne à la cellule de Voronoï de ce réseau. Finalement, la G-réduction est définie de la façon suivante : une base est dite G-réduite si elle n'est pas changée par l'algorithme glouton de réduction.

Une simple preuve par récurrence permet de montrer que l'algorithme termine. En effet, le nouveau vecteur $\mathbf{b}_d$ créé à l'étape 6 est strictement plus court que $\mathbf{b}_{d-1}$ si l'on ne sort pas de la boucle à l'étape suivante. Ainsi, le produit des normes des vecteurs $\mathbf{b}_i$ décroît strictement à chaque itération de boucle sauf éventuellement à la dernière. Puisque pour chaque réel B, le nombre de points du réseau de norme inférieure à B est fini, cela montre que l'algorithme termine.

Bien que la description de l'algorithme glouton de réduction soit relativement simple, analyser sa complexité semble être difficile. Comme on l'a vu, même le cas de la dimension 2 n'était pas trivial.

2.3.2 Une description itérative de l'algorithme glouton de réduction.

Nous donnons à la figure 2.8 une description itérative de l'algorithme glouton de réduction. Avec cette description alternative, les similarités avec l'algorithme LLL (décrit au chapitre I-1) sont plus évidentes : dans l'algorithme LLL, la résolution exacte du problème CVP de l'étape 2 est remplacée par une solution approchée, et la condition sur les longueurs de l'étape 4 est remplacée par la condition de Lovász, qui fait intervenir les orthogonalisés de Gram-Schmidt $\mathbf{b}_i^*$. Nous utiliserons cette description itérative dans la preuve de la complexité quadratique de l'algorithme, c'est-à-dire à la section 2.6, tandis que la description récursive sera utilisée aux sections 2.5 et 2.7 pour montrer que le nombre d'itérations de boucle de la variante itérative de l'algorithme est au plus linéaire en la taille de l'entrée.

Algorithme: **Glouton-itératif.**
Entrée : Une base $[\mathbf{b}_1, \ldots, \mathbf{b}_d]$ avec sa matrice de Gram.
Sortie : Une base ordonnée du réseau $L(\mathbf{b}_1, \ldots, \mathbf{b}_d)$, avec sa matrice de Gram.

1. k:=2. Tant que $k \leqslant d$, répéter
2. Calculer un vecteur $\mathbf{c} \in L(\mathbf{b}_1, \ldots, \mathbf{b}_{k-1})$ le plus proche de $\mathbf{b}_k$,
3. $\mathbf{b}_k$:=$\mathbf{b}_k - \mathbf{c}$, mettre à jour la matrice de Gram,
4. Si $\|\mathbf{b}_k\| \geqslant \|\mathbf{b}_{k-1}\|$, alors k:=$k+1$
5. Sinon insérer $\mathbf{b}_k$ à son rang k' (les vecteurs sont rangés par longueurs croissantes), mettre à jour la matrice de Gram, k:=$k'+1$.

FIG. 2.8 – Une description itérative de l'algorithme glouton de réduction.

Le résultat principal de ce chapitre est le suivant :

Théorème 12. *Soit $d \leqslant 4$. Étant donnée en entrée une base ordonnée $[\mathbf{b}_1, \ldots, \mathbf{b}_d]$, l'algorithme glouton de réduction décrit aux figures 2.7 et 2.8, qui repose sur l'algorithme de résolution du problème CVP décrit à la section 2.4, renvoie une base Minkowski-réduite du réseau $L(\mathbf{b}_1, \ldots, \mathbf{b}_d)$, en utilisant un nombre d'opérations élémentaires borné par $O(\log \|\mathbf{b}_d\| \cdot [1 + \log \|\mathbf{b}_d\| - \log \lambda_1(L)])$.*
Par ailleurs, pour $d = 5$, la base renvoyée peut ne pas être Minkowski-réduite.

2.3.3 G-réduction.

Nous étudions ici les propriétés des bases renvoyées pas l'algorithme glouton de réduction. Comme nous l'avons déjà dit, il n'est pas évident a priori que l'algorithme de Lagrange renvoie des bases Minkowski-réduites, mais il est cependant clair que la base renvoyée $[\mathbf{u}, \mathbf{v}]$ satisfait :

$$\|\mathbf{u}\| \leqslant \|\mathbf{v}\| \leqslant \|\mathbf{v} - x\mathbf{u}\|, \forall x \in \mathbb{Z}.$$

Ceci suggère la définition suivante :

Définition 25 (G-réduction). *Une base ordonnée* $[\mathbf{b}_1, \ldots, \mathbf{b}_d]$ *est G-réduite si pour tout* $i \in [|2, d|]$ *et pour tous* $x_1, \ldots, x_{i-1} \in \mathbb{Z}$ *:*

$$\|\mathbf{b}_i\| \leqslant \|\mathbf{b}_i + x_1\mathbf{b}_1 + \cdots + x_{i-1}\mathbf{b}_{i-1}\|.$$

Dit autrement, on a la définition récursive suivante : en dimension 1, toute base est G-réduite ; en dimension $d > 1$, une base ordonnée $[\mathbf{b}_1, \ldots, \mathbf{b}_d]$ est G-réduite si $[\mathbf{b}_1, \ldots, \mathbf{b}_{d-1}]$ l'est et si la projection orthogonale du vecteur $\mathbf{b}_d$ sur l'espace vectoriel engendré par les vecteurs $\mathbf{b}_1, \ldots, \mathbf{b}_{d-1}$ appartient à la cellule de Voronoï $\mathrm{Vor}(\mathbf{b}_1, \ldots, \mathbf{b}_{d-1})$. L'algorithme glouton de réduction renvoie toujours une base G-réduite, et si une base G-réduite est donnée en entrée à l'algorithme glouton de réduction, alors elle n'est pas modifiée et est renvoyée en sortie. Le fait que l'algorithme de Lagrange renvoie une base Minkowski-réduite est un cas particulier du résultat suivant, qui permet de comprendre le lien entre G-réduction et réduction au sens de Minkowski.

Lemme 11. *Les énoncés suivants sont valides :*

1. *Toute base Minkowski-réduite est G-réduite.*
2. *Une base constituée de* $d \leqslant 4$ *vecteurs est Minkowski-réduite si et seulement si elle est G-réduite.*
3. *Si* $d \geqslant 5$*, il existe une base de* d *vecteurs qui est G-réduite mais qui n'est pas réduite au sens de Minkowski.*

Démonstration. Le premier point vient directement des définitions de la G-réduction et de la réduction au sens de Minkowski. Le deuxième point est une conséquence immédiate du théorème 8 : jusqu'en dimension 4 les conditions de Minkowski ne font que des zéros et des uns. Il reste à donner un contre-exemple en dimension 5. Nous considérons pour cela le réseau engendré par les colonnes de la matrice suivante :

$$\begin{bmatrix} 2 & 0 & 0 & 0 & 1 \\ 0 & 2 & 0 & 0 & 1 \\ 0 & 0 & 2 & 0 & 1 \\ 0 & 0 & 0 & 2+\varepsilon & 1+\frac{\varepsilon}{2} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

avec $\varepsilon \in \left]0, -2 + 4\frac{\sqrt{3}}{3}\right[$. Remarquons que $-2 + 4\frac{\sqrt{3}}{3} < 0.309$.

La borne supérieure sur ε implique que $\|\mathbf{b}_1\| \leqslant \|\mathbf{b}_2\| \leqslant \|\mathbf{b}_3\| < \|\mathbf{b}_4\| < \|\mathbf{b}_5\|$. La base donnée n'est pas Minkowski-réduite car elle n'atteint pas les quatre premiers minima du réseau, ce qui ne peut pas être le cas pour une base Minkowski-réduite [184] : le vecteur $2\mathbf{b}_5 - \mathbf{b}_4 - \mathbf{b}_3 - \mathbf{b}_2 - \mathbf{b}_1 = {}^t(0, 0, 0, 0, 2)$ est linéairement indépendant avec $[\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3]$ et strictement plus court que $\mathbf{b}_4$ et $\mathbf{b}_5$. Cependant, la base est G-réduite : $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_4$ sont deux à deux orthogonaux, et $\mathbf{b}_5$ ne peut être raccourci en lui ajoutant une combinaison linéaire entière des quatre premiers vecteurs. □

On déduit de ce lemme que l'algorithme glouton de réduction renvoie toujours une base Minkowski-réduite jusqu'en dimension 4, base qui atteint ainsi les quatre premiers

minima du réseau. Au-delà de la dimension 4, l'algorithme peut renvoyer une base G-réduite qui n'est pas Minkowski-réduite. Le lemme suivant montre que la situation est encore pire que cela : le plus court vecteur de la base renvoyée peut être arbitrairement long par rapport au premier minimum du réseau.

Lemme 12. *Soit $d \geq 5$. Pour tout $\varepsilon > 0$, il existe un réseau L et une base G-réduite $[\mathbf{b}_1, \ldots, \mathbf{b}_d]$ de L tels que :*

$$\frac{\lambda_1(L)}{\|\mathbf{b}_1\|} \leqslant \varepsilon \text{ et } \frac{\text{vol } (L)}{\prod_{i=1}^{d} \|\mathbf{b}_i\|} \leqslant \varepsilon.$$

Démonstration. Considérons la base G-réduite constituée des colonnes $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_4, \mathbf{b}_5$ de la matrice suivante :

$$\begin{bmatrix} 2 & 0 & 0 & 0 & 1 \\ 0 & 2 & 0 & 0 & 1 \\ 0 & 0 & 2 & 0 & 1 \\ 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & \varepsilon \end{bmatrix},$$

où $\varepsilon > 0$ est petit. Alors $2\mathbf{b}_5 - \mathbf{b}_1 - \mathbf{b}_2 - \mathbf{b}_3 - \mathbf{b}_4$ est un vecteur du réseau, de longueur 2ε. Par ailleurs, le vecteur $\mathbf{b}_1$ est de norme 2, ce qui nous donne la première partie du résultat. Pour finir, on a vol $(L) = 16\varepsilon$, et le produit des longueurs des $\mathbf{b}_i$ est > 16. □

L'exemple donné dans la preuve précédente aurait aussi convenu pour le troisième point du lemme 11 : la base donnée alors permet d'illustrer le fait que les deux réductions peuvent différer même quand on considère des bases plutôt orthogonales.

Les propriétés décrites au lemme 12 ne peuvent pas être vérifiées par des bases réduites au sens de Minkowski. La première inégalité montre que le plus court vecteur d'une base G-réduite peut être arbitrairement long par rapport au premier minimum du réseau, et la deuxième inégalité montre qu'une base G-réduite peut être arbitrairement peu orthogonale.

2.4 Le problème CVP en petite dimension.

Nous expliquons maintenant comment les étapes 5 de l'algorithme glouton récursif et 2 de l'algorithme glouton itératif peuvent être effectuées efficacement jusqu'en dimension 5. La solution est immédiate uniquement quand $d \leqslant 2$. Sinon, commençons par remarquer que la base $[\mathbf{b}_1, \ldots, \mathbf{b}_{d-1}]$ est G-réduite et donc réduite au sens de Minkowski (quand $d \leqslant 5$), et que nous connaissons la matrice de Gram de $[\mathbf{b}_1, \ldots, \mathbf{b}_{d-1}, \mathbf{b}_d]$.

Théorème 13. *Soit $d \geqslant 1$ un entier. Il existe un algorithme qui, étant donnés en entrée une base Minkowski-réduite $[\mathbf{b}_1, \ldots, \mathbf{b}_{d-1}]$, un vecteur cible $\mathbf{t}$ plus long que tous les $\mathbf{b}_i$, et la matrice de Gram de $(\mathbf{b}_1, \ldots, \mathbf{b}_{d-1}, \mathbf{t})$, renvoie un vecteur $\mathbf{c}$ le plus proche de $\mathbf{t}$ (parmi les vecteurs du réseau $L(\mathbf{b}_1, \ldots, \mathbf{b}_{d-1})$), en temps :*

$$O(\log \|\mathbf{t}\| \cdot [1 + \log \|\mathbf{t}\| - \log \|\mathbf{b}_\alpha\|]),$$

où $\alpha \in [|1, d|]$ est un entier tel que $[\mathbf{b}_1, \ldots, \mathbf{b}_{\alpha-1}, \mathbf{t}]$ est Minkowski-réduite.

L'indice α qui apparaît dans l'énoncé du théorème nécessite une explication. Regardons la version itérative de l'algorithme. L'indice k peut croître et décroître arbitrairement. À un instant donné, l'indice α permet de savoir quels vecteurs n'ont pas changé depuis la dernière fois que l'indice k a eu sa valeur. Supposons par exemple que $k = d = 4$, puis k décroisse jusqu'à 2, puis remonte à 4. À ce dernier moment, on peut choisir $\alpha = 2$ car le vecteur $\mathbf{b}_1$ n'a pas changé depuis que l'indice k a valu 4. Intuitivement, la raison pour laquelle on peut faire apparaître l'indice α dans la complexité est que l'on est en train de travailler dans l'orthogonal de $\mathbf{b}_1, \ldots, \mathbf{b}_{\alpha-1}$, et donc que l'on ne réduit plus par rapport à ces vecteurs. La présence de α dans le théorème est cruciale : en effet, une borne de complexité de l'ordre de $O(\log \|\mathbf{t}\| \cdot [1 + \log \|\mathbf{t}\| - \log \|\mathbf{b}_1\|])$ serait trop grande pour aboutir à une complexité quadratique pour l'algorithme glouton de réduction : on aboutirait à une complexité cubique.

L'algorithme du théorème fonctionne de la manière suivante : une approximation des coordonnées (réelles) de $\mathbf{t}$ par rapport aux $\mathbf{b}_i$ est calculée par algèbre linéaire, puis cette approximation est corrigée par une recherche exhaustive adéquate.

Démonstration. Soit $\mathbf{h}$ la projection orthogonale du vecteur $\mathbf{t}$ sur l'espace vectoriel engendré par $\mathbf{b}_1, \ldots, \mathbf{b}_{d-1}$. On ne calcule pas $\mathbf{h}$ explicitement, mais on l'introduit pour simplifier la description de l'algorithme. Il existe $y_1, \ldots, y_{d-1} \in \mathbb{R}$ tels que $\mathbf{h} = \sum_{i=1}^{d-1} y_i \mathbf{b}_i$. Si $\mathbf{c} = \sum_{i=1}^{d-1} x_i \mathbf{b}_i$ est un vecteur le plus proche de $\mathbf{t}$, alors :

$$\mathbf{h} - \mathbf{c} \in \mathrm{Vor}(\mathbf{b}_1, \ldots, \mathbf{b}_{d-1}).$$

De plus, pour tout $C > 0$, si le défaut d'orthogonalité de la base est inférieur à C, alors les coordonnées (vis-à-vis de la base) de n'importe quel point à l'intérieur de la cellule de Voronoï sont bornées indépendamment du réseau (voir [173]). Il s'ensuit que si nous connaissons des approximations de y_i avec suffisamment de précision, alors $\mathbf{c}$ peut être dérivé d'une recherche exhaustive parmi $O(1)$ vecteurs : en effet, le défaut d'orthogonalité de la base Minkowski-réduite $[\mathbf{b}_1, \ldots, \mathbf{b}_{d-1}]$ est borné. Cette recherche exhaustive coûte $O(\log \|\mathbf{t}\|)$ car la matrice de Gram de $[\mathbf{b}_1, \ldots, \mathbf{b}_{d-1}, \mathbf{t}]$ est connue.

Pour approcher les y_i, nous utilisons de l'algèbre linéaire. Soient $G = G(\mathbf{b}_1, \ldots, \mathbf{b}_{d-1})$ et $H = \left(\frac{\langle \mathbf{b}_i, \mathbf{b}_j \rangle}{\|\mathbf{b}_i\|^2}\right)_{1 \leqslant i,j \leqslant d-1}$. La matrice H est exactement la matrice G, où la i-ème ligne a été divisée par $\|\mathbf{b}_i\|^2$. On a :

$$G \cdot \begin{bmatrix} y_1 \\ \vdots \\ y_{d-1} \end{bmatrix} = \begin{bmatrix} \langle \mathbf{b}_1, \mathbf{t} \rangle \\ \vdots \\ \langle \mathbf{b}_{d-1}, \mathbf{t} \rangle \end{bmatrix}, \text{ et donc : } \begin{bmatrix} y_1 \\ \vdots \\ y_{d-1} \end{bmatrix} = H^{-1} \cdot \begin{bmatrix} \frac{\langle \mathbf{b}_1, \mathbf{t} \rangle}{\|\mathbf{b}_1\|^2} \\ \vdots \\ \frac{\langle \mathbf{b}_{d-1}, \mathbf{t} \rangle}{\|\mathbf{b}_{d-1}\|^2} \end{bmatrix}.$$

Nous utilisons cette dernière formule pour calculer les y_i avec une erreur absolue inférieure à $1/2$. Soit $r = \max_i \left\lceil \log \frac{|\langle \mathbf{b}_i, \mathbf{t} \rangle|}{\|\mathbf{b}_i\|^2} \right\rceil$. Remarquons que $r = O(1 + \log \|\mathbf{t}\| - \log \|\mathbf{b}_\alpha\|)$, ce qui peut être obtenu en bornant $\langle \mathbf{b}_i, \mathbf{t} \rangle$ suivant que $i \geqslant \alpha$ ou $i < \alpha$. En effet, si $i < \alpha$, puisque la base $[\mathbf{b}_1, \ldots, \mathbf{b}_{\alpha-1}, \mathbf{t}]$ est Minkowski-réduite, le vecteur $\mathbf{t}$ ne peut être raccourci à l'aide de multiples du vecteur $\mathbf{b}_i$ et donc $|\langle \mathbf{b}_i, \mathbf{t} \rangle| \leqslant \|\mathbf{b}_i\|^2/2$, pour $i < \alpha$. Remarquons aussi que les entrées de la matrice H sont toutes $\leqslant 1$ en valeur absolue : la base $[\mathbf{b}_1, \ldots, \mathbf{b}_{d-1}]$ est Minkowski-réduite et donc Lagrange-réduite par paires. De plus, $\det(H) = \frac{\det(G)}{\|\mathbf{b}_1\|^2 \ldots \|\mathbf{b}_{d-1}\|^2}$

est borné inférieurement par une constante ne dépendant que de la dimension, car le défaut d'orthogonalité de la base Minkowski-réduite $[\mathbf{b}_1, \ldots, \mathbf{b}_{d-1}]$ est borné.

Par conséquent, on peut calculer les entrées de H^{-1} avec une erreur absolue inférieure à $2^{-\Theta(r)}$, en $O(r^2)$ opérations élémentaires : on peut voir cela avec les formules de Cramer, car les valeurs absolues des entrées de la matrice H sont bornées supérieurement par 1 et son déterminant est borné inférieurement par une constante ne dépendant que de la dimension. On obtient alors les y_i par une simple multiplication matrice-vecteur, avec l'erreur absolue escomptée : cela provient du fait que les valeurs absolues des coefficients de la matrice H^{-1} sont bornés par une constante ne dépendant que de la dimension et que les valeurs absolues des coefficients vectoriels sont inférieures à 2^r. □

Le résultat reste valide si on remplace la réduction au sens de Minkowski par n'importe quelle réduction qui garantit que le défaut d'orthogonalité est borné, par exemple la LLL-réduction. Dans le chapitre II-3, nous montrerons un résultat similaire pour la résolution approchée du problème CVP avec l'algorithme de Babai. Ce dernier peut aussi être suivi d'une recherche exhaustive parmi $O(1)$ éléments (en dimension fixée) pour aboutir à une solution exacte au problème CVP, avec la même complexité $O(\log \|\mathbf{t}\| \cdot [1 + \log \|\mathbf{t}\| - \log \|\mathbf{b}_\alpha\|])$. Il n'est pas très clair quel serait le bon choix en pratique en petite dimension.

Le théorème 13 peut être rendu efficace en utilisant les bornes données au théorème 10 : elles permettent de limiter fortement la recherche exhaustive qui suit la phase d'algèbre linéaire.

2.5 L'approche globale.

Dans cette section, nous donnons l'approche globale permettant de montrer qu'il y a au plus un nombre linéaire dans l'algorithme glouton itératif de la figure 2.8. Le but de cette approche globale est ainsi de montrer le théorème 14. La preuve de ce théorème peut être remplacée par une preuve alternative et indépendante, décrite aux sections 2.7 et 2.8, qui constituent l'approche locale. Dans les deux cas, l'analyse de l'algorithme doit être complétée par la preuve de sa complexité quadratique, effectuée à la section 2.6. Cette dernière utilise les sections 2.5, 2.7 et 2.8 uniquement via le théorème 14.

Dans cette section, nous donnons l'analyse globale, qui permet de montrer que le nombre d'itérations de boucle de la version itérative de l'algorithme glouton de réduction de la figure 2.8 est borné par $O(\log \|\mathbf{b}_d\|)$, quand $d \leqslant 4$. Plus précisément, nous montrons que :

Théorème 14. *Soit $d \leqslant 4$. Soient $[\mathbf{b}_1, \ldots, \mathbf{b}_d]$ des vecteurs linéairement indépendants. Dans l'algorithme glouton itératif de réduction décrit à la figure 2.8, le nombre d'itérations de boucle est borné par $O(\log \|\mathbf{b}_d\| - \lambda_1)$, où λ_1 est le premier minimum du réseau engendré par les vecteurs $\mathbf{b}_1, \ldots, \mathbf{b}_d$.*

Pour arriver à ce résultat, nous montrons qu'en dimension inférieure ou égale à 4, dans l'algorithme glouton récursif de réduction décrit à la figure 2.7, il y a au plus $O(1)$ itérations de boucle successives sans décroissance significative des longueurs, c'est-à-dire sans que le

produit des longueurs des vecteurs de la base courante ne décroisse d'un facteur C pour une certaine constante $C > 1$. Ceci implique qu'il ne peut y avoir plus de $O(1)$ itérations de boucle de l'algorithme glouton itératif de réduction sans une décroissance d'un facteur plus grand que C du produit des longueurs des vecteurs. Cela donne immédiatement le théorème 14.

Notre démarche est la suivante. Nous commençons par définir deux phases dans l'algorithme récursif. Dans la première phase, quand un vecteur est raccourci, sa longueur diminue d'un facteur au moins $1 + \eta$ pour une certaine constante $\eta > 0$ qui sera définie plus tard, ce qui implique que toutes les itérations de boucle de cette phase sont de « bonnes » itérations de boucle. Dans la seconde phase, les longueurs des vecteurs peuvent ne pas diminuer de manière significative, mais nous allons montrer que lorsque nous entrons dans cette phase, la base courante est déjà quasiment réduite, et il ne reste que très peu d'itérations de boucle.

À la section 2.7, nous donnerons une preuve alternative du théorème 14.

2.5.1 Deux phases dans l'algorithme glouton récursif.

Nous généralisons dans ce paragraphe l'analyse globale que nous avions effectuée en dimension 2, au paragraphe 2.2.1. Nous divisons les itérations de boucle successives de l'algorithme glouton récursif en deux phases : la η-phase et la phase restante. L'exécution commence par la η-phase et demeure dans la η-phase tant que l'itération de boucle courante introduit un vecteur $\mathbf{b}'_d$ significativement plus court que le vecteur $\mathbf{b}_d$ qu'il remplace. Dès qu'une telle diminution de longueur n'est pas atteinte, l'exécution entre dans la phase restante, et elle y reste jusqu'à la fin. Plus précisément, la η-phase est constituée des itérations de boucle de l'algorithme η-glouton décrit à la figure 2.9, qui simule donc le début de l'algorithme glouton de la figure 2.7. Cet algorithme généralise l'algorithme η-Lagrange de la figure 2.6, page 58. La phase restante correspond aux itérations de boucle de l'algorithme glouton récursif quand on lui donne en entrée la base de sortie de l'algorithme η-glouton.

Il est clair que toutes les itérations de boucle de la η-phase sont bonnes, puisque le produit des longueurs décroît à chaque fois d'un facteur au moins $1+\eta$. De plus, si $d \leqslant 4$, quand l'exécution entre dans la phase restante, la base a un défaut d'orthogonalité borné :

Lemme 13. *Soient $d \leqslant 4$ et $\eta \in \left]0, \sqrt{\frac{4}{3}} - 1\right[$. Supposons que la base $[\mathbf{b}_1, \ldots, \mathbf{b}_d]$ soit invariante par l'algorithme η-**Glouton** de la figure 2.9. Alors pour tout $k \leqslant d$, on a $\|\mathbf{b}_k^*\|^2 \geqslant \left(\frac{1}{(1+\eta)^2} + \frac{1-k}{4}\right) \|\mathbf{b}_k\|^2$. Remarquons que pour $k \leqslant 4$, on a $\frac{1}{(1+\eta)^2} + \frac{1-k}{4} > 0$.*

Démonstration. Soient $k \leqslant d$ et $\mathbf{b}'_k = \mathbf{b}_k - \mathbf{c}$ où $\mathbf{c}$ est un vecteur le plus proche de $\mathbf{b}_k$ dans $L(\mathbf{b}_1, \ldots, \mathbf{b}_{k-1})$. Nous écrivons $\mathbf{b}'_k = \mathbf{b}_k^* + \mathbf{b}_k^-$, où $\mathbf{b}_k^-$ est dans l'espace engendré par les vecteurs $[\mathbf{b}_1, \ldots, \mathbf{b}_{k-1}]$. Puisque $\mathbf{b}'_k$ ne peut pas être raccourci en lui ajoutant une combinaison linéaire entière de $\mathbf{b}_1, \ldots, \mathbf{b}_{k-1}$, on sait que :

$$\|\mathbf{b}_k^-\|^2 \leqslant \rho(\mathbf{b}_1, \ldots, \mathbf{b}_{k-1})^2 \leqslant \frac{k-1}{4} \max_{i<k} \|\mathbf{b}_i^*\|^2 \leqslant \frac{k-1}{4} \|\mathbf{b}_{k-1}\|^2 \leqslant \frac{k-1}{4} \|\mathbf{b}_k\|^2.$$

Algorithme: η-**Glouton**$(\mathbf{b}_1, \ldots, \mathbf{b}_d)$.
Entrée : Une base $[\mathbf{b}_1, \ldots, \mathbf{b}_d]$ avec sa matrice de Gram.
Sortie : Une base ordonnée de $L(\mathbf{b}_1, \ldots, \mathbf{b}_d)$, avec sa matrice de Gram.

1. Si $d = 1$, renvoyer $\mathbf{b}_1$.
2. Répéter
3. Trier $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ par longueurs croissantes, et mettre à jour la matrice de Gram,
4. $[\mathbf{b}_1, \ldots, \mathbf{b}_{d-1}]$:=**Glouton**$(\mathbf{b}_1, \ldots, \mathbf{b}_{d-1})$,
5. Calculer un vecteur $\mathbf{c} \in L(\mathbf{b}_1, \ldots, \mathbf{b}_{d-1})$ le plus proche de $\mathbf{b}_d$,
6. $\mathbf{b}'_d$:=$\mathbf{b}_d - \mathbf{c}$,
7. Si $(1+\eta)\|\mathbf{b}'_d\| > \|\mathbf{b}_d\|$, aller à l'étape 9.
8. Sinon, $\mathbf{b}_d$:=$\mathbf{b}'_d$ et mettre à jour la matrice de Gram.
9. Renvoyer $[\mathbf{b}_1, \ldots, \mathbf{b}_d]$ et sa matrice de Gram.

FIG. 2.9 – L'algorithme η-**Glouton** de réduction en dimension d.

En utilisant le théorème de Pythagore, on obtient que :

$$\|\mathbf{b}_k\|^2 \leqslant (1+\eta)^2\|\mathbf{b}'_k\|^2 \leqslant (1+\eta)^2(\|\mathbf{b}^*_k\|^2 + \|\mathbf{b}^-_k\|^2) \leqslant (1+\eta)^2\left(\|\mathbf{b}^*_k\|^2 + \frac{k-1}{4}\|\mathbf{b}_k\|^2\right),$$

ce qui donne le résultat. □

2.5.2 L'algorithme glouton avec une base plutôt orthogonale.

Pour conclure en dimension inférieure à 4, il suffit maintenant de montrer que lorsque les $\|\mathbf{b}^*_i\|/\|\mathbf{b}_i\|$ sont bornés inférieurement (c'est-à-dire quand le défaut d'orthogonalité est borné) alors le nombre d'itérations de boucle de l'algorithme glouton récursif est borné par une constante.

Lemme 14. *Soient $d \leqslant 4$ et $C > 0$. Il existe une constante K telle que pour toute base $[\mathbf{b}_1, \ldots, \mathbf{b}_d]$ qui satisfait $\prod \frac{\|\mathbf{b}^*_i\|}{\|\mathbf{b}_i\|} \geqslant C$, l'algorithme glouton récursif finit sur l'entrée $[\mathbf{b}_1, \ldots, \mathbf{b}_d]$ en au plus K itérations de boucle.*

Démonstration. Puisque pour tout $i \leqslant d$ on a $\|\mathbf{b}^*_i\| \leqslant \|\mathbf{b}_i\|$, on a aussi $\frac{\|\mathbf{b}^*_i\|}{\|\mathbf{b}_i\|} \geqslant C$ pour tout $i \leqslant d$. Si initialement la base satisfait $\frac{\prod\|\mathbf{b}^*_i\|}{\prod\|\mathbf{b}_i\|} \geqslant C$, puisque le numérateur est constant (c'est le volume du réseau) et que le dénominateur décroît, toutes les bases qui apparaissent dans l'exécution de l'algorithme satisfont cette propriété. Ainsi, pour tout vecteur $\mathbf{b}_i$ apparaissant dans l'exécution de l'algorithme on a : $\frac{\|\mathbf{b}^*_i\|}{\|\mathbf{b}_i\|} \geqslant C$.

Nous définissons l'ensemble $X_i = \{(x_i, \ldots, x_d), \exists (x_1, \ldots, x_{i-1}) \in \mathbb{Z}^{i-1}, \|\sum_j x_j\mathbf{b}_j\| \leqslant \|\mathbf{b}_i\|\}$ pour tout $i \leqslant d$. Nous avons $|X_i| \leqslant (1 + 2/C)^{d-i+1}$. Nous montrons cela par récurrence décroissante sur i. Soit $\mathbf{b} = x_1\mathbf{b}_1 + \ldots + x_d\mathbf{b}_d$ avec $\|\mathbf{b}\| \leqslant \|\mathbf{b}_d\|$. En considérant la composante de $\mathbf{b}$ sur $\mathbf{b}^*_d$, on obtient $|x_d| \leqslant 1/C$. Supposons maintenant que $i < d$. Soit $\mathbf{b} = x_1\mathbf{b}_1 + \ldots + x_d\mathbf{b}_d$ avec $\|\mathbf{b}\| \leqslant \|\mathbf{b}_i\|$. Comme la base est ordonnée, nous avons $\|\mathbf{b}\| \leqslant \|\mathbf{b}_{i+1}\|$,

ce qui donne par hypothèse de récurrence que $(x_{i+1}, \ldots, x_d)$ appartient à un ensemble fini. De plus, en prenant la composante de $\mathbf{b}$ sur $\mathbf{b}_i^*$, on obtient que :

$$\left| x_i + \sum_{j=i+1}^{d} x_j \frac{\langle \mathbf{b}_j, \mathbf{b}_i^* \rangle}{\|\mathbf{b}_i^*\|^2} \right| \leqslant 1/C.$$

Ainsi, pour chaque choix de $(x_{i+1}, \ldots, x_d)$, il y a au plus $2/C+1$ valeurs possibles pour x_i.

Considérons maintenant l'exécution de l'algorithme glouton de réduction sur une telle base. Le nombre de fois qu'un vecteur plus court que $\mathbf{b}_1$ est créé est borné par $(1+2/C)^d$. Ainsi, on peut couper l'exécution de l'algorithme en phases pendant lesquelles le vecteur $\mathbf{b}_1$ ne change pas, et il y en a au plus $\left(1+\frac{2}{C}\right)^d$. Considérons une telle phase. Soit $\mathbf{b}_1, \ldots, \mathbf{b}_d$ la base au début de cette phase. Elle satisfait $\prod \frac{\|\mathbf{b}_i^*\|}{\|\mathbf{b}_i\|} \geqslant C$. Au plus $|X_2| \leqslant (1+2/C)^{d-1}$ fois dans cette phase un nouveau vecteur $\mathbf{b}_2$ est créé : pour tout $(x_2, \ldots, x_d) \in X_2$, il n'y a qu'au plus deux valeurs possibles de x_1 à cause du choix glouton de l'algorithme à l'étape 2 de la version itérative ou de l'étape 5 de la version récursive (il peut y avoir deux solutions à une instance du problème CVP en dimension 1). On peut donc découper l'exécution de l'algorithme en phases pendant lesquelles $\mathbf{b}_1$ et $\mathbf{b}_2$ sont constants, et il y en a moins de $2(1+2/C)^{2d-1}$. En utilisant la borne sur $|X_i|$ et la finitude des solutions à une instance du problème CVP en dimension $i-1$ (c'est le « kissing number », voir [42]), le même raisonnement peut être fait pour découper l'exécution de l'algorithme en phases (en nombre borné indépendamment de la base) pendant lesquelles les vecteurs $\mathbf{b}_1, \ldots, \mathbf{b}_i$ ne changent pas. Le cas $i = d$ donne le résultat. □

2.6 La preuve de la complexité quadratique.

Dans cette section, nous utilisons les théorèmes 13 et 14 des deux sections précédentes pour montrer la borne de complexité affirmée au théorème 12. Pour faire cela, nous généralisons le phénomène d'annulation utilisé dans l'algorithme d'Euclide et dans l'algorithme de Lagrange (voir la section 2.2). Nous étendrons cette technique au chapitre II-3 pour analyser l'algorithme LLL.

Supposons que $d \leqslant 4$. Nous considérons la description itérative de l'algorithme glouton de réduction en dimension d, et notons $[\mathbf{b}_1^{(t)}, \ldots, \mathbf{b}_d^{(t)}]$ la base courante au début de la t-ième itération de boucle. Au début, nous avons : $[\mathbf{b}_1^{(1)}, \ldots, \mathbf{b}_d^{(1)}] = [\mathbf{b}_1, \ldots, \mathbf{b}_d]$. Le théorème 13 nous donne que le coût de la t-ième itération de boucle est borné par $O(\log \|\mathbf{b}_d\| \cdot (1 + \log \|\mathbf{b}_{k(t)}^{(t)}\| - \log \|\mathbf{b}_{\alpha(t)}^{(t)}\|))$. Le théorème 14 donne que le nombre τ d'itérations de boucle de l'algorithme est borné par $O(\log \|\mathbf{b}_d\|)$. Nous avons deux indices intéressants pour l'analyse de complexité : l'indice $k(t)$ signifie qu'à la t-ième itération nous essayons de raccourcir le vecteur $\mathbf{b}_{k(t)}^{(t)}$, et l'indice $\alpha(t)$, qui est défini précisément dans le lemme suivant, et qui correspond au plus grand i tel qu'aucun des vecteurs $\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}$ n'a été changé depuis la dernière fois que l'indice k a été égal à $k(t)$. Intuitivement, la signification de l'indice α est la suivante : dans toute cette série d'itérations (depuis la dernière fois que l'on a changé le k-ième vecteur), les premiers vecteurs (jusqu'à $\mathbf{b}_{\alpha-1}$) n'ont jamais été

modifiés ; ainsi, pendant toutes ces itérations, les vecteurs créés sont toujours réduits avec les premiers vecteurs, et c'est quasiment comme si l'on travaillait dans l'orthogonal de ces premiers vecteurs ; ces premiers vecteurs n'interviennent pas dans le coût de ces itérations de boucle.

Lemme 15. *Soit t une itération de boucle. Soient $\phi(t) = \max(t' < t, k(t') \geqslant k(t))$ si cela existe, et 1 sinon, et $\alpha(t) = \min(k(t'), t' \in [|\phi(t), t[|) - 1$. Alors le coût de la t-ième itération de boucle est borné par $O(\log \|\mathbf{b}_d\| \cdot [1 + \log \|\mathbf{b}_{k(t)}^{(t)}\| - \log \|\mathbf{b}_{\alpha(t)}^{(t)}\|])$.*

Démonstration. Entre les itérations de boucle $\phi(t)$ et t, les vecteurs $\mathbf{b}_1, \ldots, \mathbf{b}_{\alpha(t)-1}$ sont inchangés, et à cause des choix gloutons faits aux diverses étapes 2 et 3, chaque vecteur $\mathbf{b}$ créé pendant ces itérations de boucle est tel que $[\mathbf{b}_1, \ldots, \mathbf{b}_{\alpha(t)-1}, \mathbf{b}]$ est G-réduit, et donc réduit au sens de Minkowski si $\alpha(t) \leqslant 4$ (la G-réduction et la Minkowski-réduction sont équivalentes jusqu'en dimension 4). Ces vecteurs $\mathbf{b}$ incluent le vecteur $\mathbf{b}_{k(t)}^{(t)}$. Le théorème 13 nous donne le résultat car tous les vecteurs apparaissant pendant ces itérations de boucle sont plus courts que $\mathbf{b}_d$. □

Nous allons séparer en $O(d)$ sous-sommes la somme des coûts des itérations de boucle successives, en fonction de la valeur de $k(t)$:

$$\sum_{t \leqslant \tau} \left[1 + \log \|\mathbf{b}_{k(t)}^{(t)}\| - \log \|\mathbf{b}_{\alpha(t)}^{(t)}\|\right] \leqslant \tau + \sum_{k=2}^{d} \sum_{t, k(t)=k} (\log \|\mathbf{b}_k^{(t)}\| - \log \|\mathbf{b}_{\alpha(t)}^{(t)}\|).$$

Pour chacune de ces sous-sommes, nous gardons $k-1$ termes positifs et $k-1$ termes négatifs, et « éliminons » les autres. Le point central de ce processus est le suivant :

Lemme 16. *Soient $k \in [|2, d|]$ et $t_1 < t_2 < \ldots < t_k$ des itérations de boucle de l'algorithme itératif glouton de réduction tels que pour tout $j < k$, on a $k(t_j) = k$. Alors il existe $j < k$ tel que $\|\mathbf{b}_{\alpha(t_j)}^{(t_j)}\| \geqslant \|\mathbf{b}_k^{(t_k)}\|$.*

Démonstration. Nous choisissons $j = \max(i \leqslant k, \alpha(t_i) \geqslant i)$. Cette définition est valable car l'ensemble maximisé n'est pas vide : il contient 1. Puisque $\alpha(t_k) < k$ et $k(t_k) = k(t_{k-1}) = k$, il existe une première itération de boucle $T_k \in [|t_{k-1}, t_k[|$ telle que $k(T_k) \geqslant k \geqslant k(T_k + 1)$. Puisque pour un indice donné les longueurs des vecteurs décroissent toujours, nous avons :

$$\|\mathbf{b}_k^{(t_k)}\| \leqslant \|\mathbf{b}_k^{(T_k+1)}\| \leqslant \|\mathbf{b}_{k-1}^{(T_k)}\|.$$

Par définition de T_k les vecteurs $\mathbf{b}_1, \ldots, \mathbf{b}_{k-1}$ ne changent pas entre les itérations de boucle t_{k-1} et T_k. Ainsi :

$$\|\mathbf{b}_k^{(t_k)}\| \leqslant \|\mathbf{b}_{k-1}^{(t_{k-1})}\|.$$

Si $j = k-1$, nous avons le résultat. Sinon, il existe une première itération de boucle $T_{k-1} \in [|t_{k-2}, t_{k-1}[|$ telle que $k(T_{k-1}) \geqslant k-1 \geqslant k(T_{k-1}+1)$. Nous avons :

$$\|\mathbf{b}_{k-1}^{(t_{k-1})}\| \leqslant \|\mathbf{b}_{k-1}^{(T_{k-1}+1)}\| \leqslant \|\mathbf{b}_{k-2}^{(T_{k-1})}\| \leqslant \|\mathbf{b}_{k-2}^{(t_{k-2})}\|.$$

Si $j = k-2$, nous avons le résultat escompté, et sinon nous continuons à construire des itérations de boucle T_i de la même manière pour aboutir au résultat. □

Il est maintenant possible d'achever l'analyse de complexité. Soient $k \in [|2, d|]$ et $t_1 < t_2 < \ldots < t_{\tau_k} = \{t \leqslant \tau, k(t) = k\}$. Nous avons :

$$\sum_{i=1}^{\tau_k} (\log \|\mathbf{b}_k^{(t_i)}\| - \log \|\mathbf{b}_{\alpha(t_i)}^{(t_i)}\|) \leqslant k(\log \|\mathbf{b}_d\| - \lambda_1) + \left(\sum_{i=k}^{\tau_k} \log \|\mathbf{b}_k^{(t_i)}\| - \sum_{i=1}^{\tau_k - k + 1} \log \|\mathbf{b}_{\alpha(t_i)}^{(t_i)}\| \right),$$

où λ_1 est le premier minimum du réseau que l'on réduit. Le lemme 16 sert à borner la partie droite de la quantité ci-dessus. On l'applique d'abord avec $t_1, \ldots, t_k$. Cela donne qu'il existe $j < k$ tel que $\|\mathbf{b}_k^{(t_k)}\| \leqslant \|\mathbf{b}_{\alpha(t_j)}^{(t_j)}\|$. Les termes d'indices « $i = k$ » dans la somme des termes positifs et « $i = j$ » dans la somme des termes négatifs s'annulent. Nous utilisons alors le lemme 16 avec t_{k+1} et les $k - 1$ premiers t_i qui restent dans la somme des termes négatifs. On peut voir que t_{k+1} est plus grand que n'importe lequel d'entre eux, et donc nous pouvons faire apparaître une nouvelle annulation « terme positif-terme négatif ». On effectue ce procédé $\tau_k - k + 1$ fois, pour obtenir :

$$\sum_{i=k}^{\tau_k} \log \|\mathbf{b}_k^{(t_i)}\| - \sum_{i=1}^{\tau_k - k + 1} \log \|\mathbf{b}_{\alpha(t_i)}^{(t_i)}\| \leqslant 0.$$

Le fait que $\sum_k \tau_k = \tau = O(\log \|\mathbf{b}_d\| - \log \lambda_1)$ (qui provient du théorème 14) permet de finir la preuve du théorème 12.

2.7 L'approche locale.

Cette section et la suivante offrent une preuve différente du théorème 14. L'approche locale donne une meilleure compréhension du comportement de l'algorithme, mais est moins rapide que l'approche globale.

Dans cette section, nous donnons une preuve alternative du théorème 14 pour $d \leqslant 4$. Celle-ci généralise l'analyse locale que nous avons faite pour l'algorithme de Lagrange. Nous montrons que l'algorithme glouton itératif de réduction de la figure 2.8 a un nombre d'itérations de boucle au plus linéaire en la taille de l'entrée. Pour prouver ce résultat, nous montrons qu'il ne peut y avoir plus de $O(1)$ itérations de boucle consécutives sans une décroissance significative du produit des longueurs des vecteurs courants. Plus précisément, nous montrons le théorème suivant, qui implique directement le théorème 14.

Théorème 15. *Soit $d \leqslant 4$. Il existe trois constantes $C > 1, I, F$ telles que pour n'importe quelles d itérations de boucle consécutives de l'algorithme glouton récursif de réduction de la figure 2.7, certaines d'entre elles font partie des I itérations initiales ou des F itérations finales, ou le produit des longueurs des vecteurs de la base courante décroît d'au moins un facteur C.*

Ce résultat implique bien le théorème 14 pour $d \leqslant 4$. En effet, le nombre d'itérations de boucle successives (de l'algorithme itératif) sans décroissance significative assurée du produit des longueurs est borné par $(I + F + C)^d = O(1)$. À partir de maintenant, nous nous intéresserons uniquement à la version récursive de l'algorithme glouton de réduction.

2.7.1 Une analyse géométrique unifiée jusqu'en dimension 4.

L'analyse locale de l'algorithme de Lagrange donnée à la section 2.2 reposait sur le fait que si $|x| \geq 2$, alors le vecteur $x\mathbf{u}$ était très éloigné de la cellule de Voronoï du réseau de dimension 1 engendré par $\mathbf{u}$. L'analyse du nombre d'itérations de boucle de l'algorithme glouton de réduction en dimensions 3 et 4 repose sur un phénomène similaire lié à des cellules de Voronoï en dimensions respectivement 2 et 3. Cependant, la situation est relativement plus complexe, comme le suggèrent les remarques suivantes :

- Pour $d = 2$, nous étudions la valeur de l'entier x, mais quand $d \geqslant 3$, nous disposons de plusieurs entiers x_i au lieu d'un seul, et il n'est pas facile a priori de dire celui qui va s'avérer utile.
- Pour $d = 2$, l'appel récursif (qui est inexistant) ne peut pas changer la base courante, puisque pour un réseau de dimension 1 il n'y a qu'une base possible (au signe près). Si $d \geq 3$, l'appel récursif peut changer complètement les vecteurs, et il paraît difficile dans ce cas de dire précisément ce qui se passe.

Pour prouver le théorème 15, nous introduisons quelques notations. Considérons la i-ème itération de boucle de l'algorithme. Soit $[\mathbf{a}_1^{(i)}, \ldots, \mathbf{a}_d^{(i)}]$ la base courante $[\mathbf{b}_1, \ldots, \mathbf{b}_d]$ au début de cette i-ème itération de boucle. La base $[\mathbf{a}_1^{(i)}, \ldots, \mathbf{a}_d^{(i)}]$ devient $[\mathbf{b}_1^{(i)}, \ldots, \mathbf{b}_{d-1}^{(i)}, \mathbf{a}_d^{(i)}]$ avec $\|\mathbf{b}_1^{(i)}\| \leqslant \cdots \leqslant \|\mathbf{b}_{d-1}^{(i)}\|$ après l'appel récursif, et $(\mathbf{b}_1^{(i)}, \ldots, \mathbf{b}_d^{(i)})$ après l'étape de translation, où $\mathbf{b}_d^{(i)} = \mathbf{a}_d^{(i)} - \mathbf{c}^{(i)}$ et $\mathbf{c}^{(i)}$ est le vecteur (parmi les plus proches) qui a été trouvé à l'étape 5. Soit p_i le nombre d'entiers $1 \leqslant j \leqslant d$ tels que $\|\mathbf{b}_j^{(i)}\| \leq \|\mathbf{b}_d^{(i)}\|$. Soit π_i le rang de $\mathbf{b}_d^{(i)}$ une fois que la base $(\mathbf{b}_1^{(i)}, \ldots, \mathbf{b}_d^{(i)})$ a été triée par ordre croissant des longueurs : par exemple, $\pi_i = 1$ si $\|\mathbf{b}_d^{(i)}\| < \|\mathbf{b}_1^{(i)}\|$. Remarquons que π_i peut être différent de p_i parce qu'il peut y avoir plusieurs possibilités pour le tri des vecteurs dans le cas où certains d'entre eux sont de normes égales. Clairement, on a $1 \leq \pi_i \leq p_i \leq d$, si $p_i = d$ alors la boucle s'achève, et on a $\|\mathbf{a}_{\pi_i}^{(i+1)}\| = \|\mathbf{a}_{p_i}^{(i+1)}\|$.

Nous considérons la $(i+1)$-ème itération de boucle pour un certain $i \geq 1$. Remarquons que la définition de π_i nous donne que $\mathbf{a}_{\pi_i}^{(i+1)} = \mathbf{b}_d^{(i)} = \mathbf{a}_d^{(i)} - \mathbf{c}^{(i)}$, alors que $\{\mathbf{a}_j^{(i+1)}\}_{j \neq \pi_i} = \{\mathbf{b}_j^{(i)}\}_{1 \leq j \leq d-1}$. Le vecteur $\mathbf{c}^{(i+1)}$ appartient à $L(\mathbf{b}_1^{(i+1)}, \ldots, \mathbf{b}_{d-1}^{(i+1)}) = L(\mathbf{a}_1^{(i+1)}, \ldots, \mathbf{a}_{d-1}^{(i+1)})$: il existe donc des entiers $x_1^{(i+1)}, \ldots, x_{d-1}^{(i+1)}$ tels que $\mathbf{c}^{(i+1)} = \sum_{j=1}^{d-1} x_j^{(i+1)} \mathbf{a}_j^{(i+1)}$.

Nous allons montrer qu'il existe une constante universelle $K > 1$ telle que pour n'importe quelle exécution de l'algorithme de réduction en dimension d avec $d \leqslant 4$, et pour n'importe quelles d itérations de boucle successives (sauf éventuellement les premières et les dernières), le produit des longueurs des vecteurs de la base courante décroît au moins d'un facteur K :

$$\frac{\|\mathbf{a}_1^{(i)}\| \ldots \|\mathbf{a}_d^{(i)}\|}{\|\mathbf{a}_1^{(i+d)}\| \ldots \|\mathbf{a}_d^{(i+d)}\|} \geqslant K. \tag{2.1}$$

Cela impliquera automatiquement que le nombre d'itérations de boucle est au plus proportionnel à $\log \|\mathbf{a}_d^{(1)}\|$.

Nous nous occupons maintenant de la première difficulté que nous avons mentionnée : quel coefficient entier va nous servir ? L'astuce est de s'intéresser à la valeur de $x_{\pi_i}^{(i+1)}$,

c'est-à-dire au coefficient multiplicatif de $\mathbf{a}_{\pi_i}^{(i+1)} = \mathbf{a}_d^{(i)} - \mathbf{c}^{(i)}$ dans $\mathbf{c}^{(i+1)}$, et d'utiliser les propriétés gloutonnes de l'algorithme. Ce coefficient est particulier car il correspond au vecteur qui a été créé à l'itération précédente. Puisque ce vecteur a été créé de telle sorte qu'il ne puisse plus être raccourci à l'aide des autres, on n'a que deux alternatives pour créer un vecteur à l'itération courante : soit il est plus long que $\mathbf{a}_{\pi_i}^{(i+1)}$, auquel cas p_i augmente (ce qui ne peut pas se reproduire très longtemps de façon consécutive), soit il est plus court, et on a forcément $|x_{\pi_i}| \neq 1$ car sinon cela serait en contradiction avec la manière dont on a créé le vecteur $\mathbf{a}_{\pi_i}^{(i+1)}$ (on aurait pu faire un meilleur choix pour ce vecteur à l'itération précédente).

Lemme 17 (Lemme glouton). *Parmi d itérations de boucle consécutives de l'algorithme glouton de réduction de la figure 2.7, il y en a au moins une d'indice $i+1$ pour un certain $i \geqslant 1$ telle que $p_{i+1} \leqslant p_i$. De plus, pour une telle itération de boucle, $|x_{\pi_i}^{(i+1)}| \geqslant 2$, ou il s'agit de la dernière itération de boucle.*

Démonstration. La première partie du lemme est immédiate car pour tout i, on a $p_i \in [|1, d|]$, et si l'on avait $p_i = d$, alors il n'y aurait qu'une seule itération de boucle. Considérons maintenant une telle itération de boucle, et supposons que nous ayons un petit entier $x_{\pi_i}^{(i+1)}$, c'est-à-dire $x_{\pi_i}^{(i+1)} = 0$ ou $|x_{\pi_i}^{(i+1)}| = 1$.

- Si $x_{\pi_i}^{(i+1)} = 0$, alors $\mathbf{c}^{(i+1)} \in L(\mathbf{a}_j^{(i+1)})_{j \neq \pi_i, j \leqslant d-1} = L(\mathbf{b}_1^{(i)}, \ldots, \mathbf{b}_{d-2}^{(i)})$. Nous affirmons que dans ce cas, la $(i+1)$-ème itération de boucle est la dernière dans l'exécution de l'algorithme. Puisque la i-ème itération de boucle n'est pas la dernière, nous avons l'égalité $\mathbf{a}_d^{(i+1)} = \mathbf{b}_{d-1}^{(i)}$. De plus, la base $[\mathbf{b}_1^{(i)}, \ldots, \mathbf{b}_{d-1}^{(i)}]$ est déjà G-réduite à cause de l'appel récursif de la i-ème itération de boucle. Tout cela implique que $\mathbf{c}^{(i+1)}$ est le vecteur nul (ou en tout cas qu'il ne permet pas de raccourcir $\mathbf{a}_d^{(i)}$ s'il y a plusieurs plus proches vecteurs), et donc que la $(i+1)$-ème itération de boucle est la dernière.
- Si $|x_{\pi_i}^{(i+1)}| = 1$, alors on a $p_{i+1} > p_i$, ce qui est contradictoire avec notre hypothèse. En effet, on a $\mathbf{c}^{(i+1)} = \sum_{j=1}^{d-1} x_j^{(i+1)} \mathbf{a}_j^{(i+1)}$ où $\mathbf{a}_{\pi_i}^{(i+1)} = \mathbf{a}_d^{(i)} - \mathbf{c}^{(i)}$ et $\{\mathbf{a}_j^{(i+1)}\}_{j \neq \pi_i} = \{\mathbf{b}_j^{(i)}\}_{1 \leqslant j \leqslant d-1}$. Ainsi, le vecteur $\mathbf{c}^{(i+1)}$ peut s'écrire $\mathbf{c}^{(i+1)} = \mp(\mathbf{a}_d^{(i)} - \mathbf{c}^{(i)}) - \mathbf{e}$ avec $\mathbf{e} \in L(\mathbf{b}_1^{(i)}, \ldots, \mathbf{b}_{d-1}^{(i)})$. Par conséquent, $\mathbf{a}_d^{(i+1)} - \mathbf{c}^{(i+1)} = \mathbf{b}_{d-1}^{(i)} \pm (\mathbf{a}_d^{(i)} - \mathbf{c}^{(i)}) + \mathbf{e}$. Autrement dit, $\|\mathbf{a}_d^{(i+1)} - \mathbf{c}^{(i+1)}\| = \|\mathbf{a}_d^{(i)} - \mathbf{f}\|$ pour un vecteur $\mathbf{f}$ dans $L(\mathbf{b}_1^{(i)}, \ldots, \mathbf{b}_{d-1}^{(i)})$. Le choix glouton de $\mathbf{b}_d(i)$ à la i-ème itération de boucle implique que $p_{i+1} \geq 1 + p_i$, ce qui achève la preuve du lemme.

□

Nous allons montrer qu'en dimension 3, lors d'une telle itération de boucle $(i+1)$, au moins une des longueurs des vecteurs de la base décroît de manière significative, ou qui a décru de manière significative à la i-ème itération de boucle. Cela n'est plus vrai en dimension 4, mais n'est pas très loin de l'être : nous isolerons les mauvais cas et montrerons que lorsqu'un mauvais cas apparaît, le nombre d'itérations de boucle restantes est borné par une constante. Plus précisément, les mauvais cas sont ceux pour lesquels les x_i calculés satisfont $|x_{\sigma(1)}| = |x_{\sigma(2)}| = 1$ et $|x_{\sigma(3)}| = 2$, pour une certaine permutation σ de $\{1, 2, 3\}$.

Cette situation sera considérée au paragraphe 2.7.2.

Nous nous intéressons désormais à la deuxième difficulté soulevée plus haut, c'est-à-dire au problème soulevé par le changement éventuel des premiers vecteurs lors de l'appel récursif. Nous avons l'égalité $\mathbf{c}^{(i+1)} = \sum_{j=1}^{d-1} x_j^{(i+1)} \mathbf{a}_j^{(i+1)}$ mais la base $[\mathbf{a}_1^{(i+1)}, \ldots, \mathbf{a}_{d-1}^{(i+1)}]$ n'est pas nécessairement G-réduite. Nous distinguons deux situations :

1. La base $[\mathbf{a}_1^{(i+1)}, \ldots, \mathbf{a}_{d-1}^{(i+1)}]$ est, en un certain sens qui sera précisé plus bas, loin d'être G-réduite. Alors le vecteur $\mathbf{b}_d^{(i)}$ calculé à la i-ème itération de boucle est nettement plus court que le vecteur $\mathbf{a}_d^{(i)}$ qu'il remplace. Remarquons que cette décroissance des longueurs concerne la i-ème itération de boucle et non la $(i+1)$-ème.
2. Sinon, la base $[\mathbf{a}_1^{(i+1)}, \ldots, \mathbf{a}_{d-1}^{(i+1)}]$ est presque G-réduite. L'inégalité $|x_{\pi_i}^{(i+1)}| \geqslant 2$ implique en quelque sorte que le vecteur $\mathbf{c}^{(i+1)}$ est très loin de la cellule de Voronoï $\mathrm{Vor}(\mathbf{a}_1^{(i+1)}, \ldots, \mathbf{a}_{d-1}^{(i+1)})$: ce phénomène sera pris en compte de façon précise par le lemme du vide (théorème 16, page 75). Quand cette situation se présente, le nouveau vecteur $\mathbf{b}_d^{(i+1)}$ est nettement plus court que le vecteur $\mathbf{a}_d^{(i+1)}$ qu'il remplace. Intuitivement, le lemme du vide a la signification suivante : si l'on considère un vecteur éloigné de la cellule de Voronoï d'un réseau (c'est-à-dire qui est à l'extérieur d'une couronne encerclant cette cellule de Voronoï), et un vecteur de la cellule de Voronoï, alors le deuxième est significativement plus court que le premier. L'existence de la couronne proviendra du fait que la base $[\mathbf{a}_1^{(i+1)}, \ldots, \mathbf{a}_{d-1}^{(i+1)}]$ est presque réduite et que l'on a $|x_{\pi_i}^{(i+1)}| \geqslant 2$.

Pour formaliser l'idée qu'un ensemble de vecteurs est quasiment G-réduit, nous introduisons la ε-G-réduction.

Définition 26 (ε-G-réduction). *Soit $\varepsilon \geqslant 0$. Un vecteur isolé $[\mathbf{b}_1]$ est toujours ε-G-réduit ; pour $d \geqslant 2$, un d-uplet $[\mathbf{b}_1, \ldots, \mathbf{b}_d]$ est ε-G-réduit si $[\mathbf{b}_1, \ldots, \mathbf{b}_{d-1}]$ est ε-G-réduit et la projection orthogonale de $\mathbf{b}_d$ sur l'espace engendré par $[\mathbf{b}_1, \ldots, \mathbf{b}_{d-1}]$ appartient à l'ensemble $(1+\varepsilon)\mathrm{Vor}(\mathbf{b}_1, \ldots, \mathbf{b}_{d-1})$.*

On a immédiatement qu'une base G-réduite est aussi ε-G-réduite pour n'importe quel $\varepsilon \geqslant 0$. Dans la définition, nous n'avons pas supposé que les vecteurs $\mathbf{b}_i$ ne sont pas nuls ni linéairement indépendants. La raison est que le lemme du vide repose essentiellement sur des propriétés de compacité : l'ensemble des d-uplets ε-G-réduits doit être fermé (d'un point de vue topologique), alors qu'une limite de bases peut ne pas être une base (les vecteurs peuvent devenir linéairement dépendants). Cela a pour conséquence que la matrice de Gram d'une telle base peut ne pas être inversible.

Nous pouvons maintenant donner un énoncé précis des deux cas que nous venons de décrire. Le lemme 18 correspond à la première situation et le lemme 19 à la seconde.

Lemme 18. *Soit $2 \leqslant d \leqslant 4$. Il existe une constante $\varepsilon_1 > 0$ telle que pour tout $\varepsilon \in]0, \varepsilon_1]$, il existe $C_\varepsilon > 1$ tel que la propriété suivante est valide. Considérons la $(i+1)$-ème itération de boucle d'une exécution de l'algorithme glouton de la figure 2.7 en dimension d. Si $[\mathbf{a}_1^{(i+1)}, \ldots, \mathbf{a}_{d-1}^{(i+1)}]$ n'est pas ε-G-réduite, alors $\|\mathbf{a}_d^{(i)}\| \geqslant C_\varepsilon \|\mathbf{b}_d^{(i)}\|$.*

Démonstration. Le résultat est évident pour $d = 2$ puisqu'un vecteur isolé est toujours ε-G-réduit. Supposons que $d = 3$ et que $[\mathbf{a}_1^{(i+1)}, \mathbf{a}_2^{(i+1)}]$ ne soit pas ε-G-réduite. Nous avons $|\langle \mathbf{a}_2^{(i+1)}, \mathbf{a}_1^{(i+1)} \rangle| \geqslant \frac{1+\varepsilon}{2} \|\mathbf{a}_1^{(i+1)}\|^2$. Par ailleurs, nous avons forcément $\pi_i = 1$ (sinon la base $[\mathbf{a}_1^{(i+1)}, \mathbf{a}_2^{(i+1)}] = [\mathbf{b}_1^{(i)}, \mathbf{b}_2^{(i)}]$ serait Minkowski-réduite), ce qui implique que le vecteur $\mathbf{a}_1^{(i+1)} = \mathbf{b}_3^{(i)}$ ne peut pas être raccourci en lui additionnant des multiples entiers du vecteur $\mathbf{a}_2^{(i+1)} = \mathbf{b}_1^{(i)}$. Ceci signifie que $|\langle \mathbf{a}_2^{(i+1)}, \mathbf{a}_1^{(i+1)} \rangle| \leqslant \|\mathbf{a}_2^{(i+1)}\|^2/2$. Les deux inégalités prises simultanément donnent :

$$(1+\varepsilon)\|\mathbf{a}_1^{(i+1)}\|^2 \leqslant \|\mathbf{a}_2^{(i+1)}\|^2.$$

Les faits que $\mathbf{a}_1^{(i+1)} = \mathbf{b}_3^{(i)}$ et que $\|\mathbf{a}_2^{(i+1)}\| \leqslant \|\mathbf{a}_3^{(i)}\|$ achèvent la preuve.

Supposons désormais que $d = 4$ et que $[\mathbf{a}_1^{(i+1)}, \mathbf{a}_2^{(i+1)}, \mathbf{a}_3^{(i+1)}]$ ne soit pas ε-G-réduite. Cela signifie que la projection orthogonale de $\mathbf{a}_2^{(i+1)}$ sur l'espace engendré par $\mathbf{a}_1^{(i+1)}$ n'est pas dans $(1+\varepsilon)\mathrm{Vor}(\mathbf{a}_1^{(i+1)})$, ou que la projection de $\mathbf{a}_3^{(i+1)}$ sur l'espace engendré par $[\mathbf{a}_1^{(i+1)}, \mathbf{a}_2^{(i+1)}]$ n'est pas dans $(1+\varepsilon)\mathrm{Vor}(\mathbf{a}_1^{(i+1)}, \mathbf{a}_2^{(i+1)})$. Nous considérons ces deux situations séparément. Dans un premier temps, supposons que la projection orthogonale du vecteur $\mathbf{a}_2^{(i+1)}$ sur l'espace engendré par le vecteur $\mathbf{a}_1^{(i+1)}$ ne soit pas dans $(1+\varepsilon)\mathrm{Vor}(\mathbf{a}_1^{(i+1)})$. Alors $|\langle \mathbf{a}_2^{(i+1)}, \mathbf{a}_1^{(i+1)} \rangle| \geqslant \frac{1+\varepsilon}{2} \|\mathbf{a}_1^{(i+1)}\|^2$. De plus, nous avons forcément $\pi_i = 1$ (sinon la base $[\mathbf{a}_1^{(i+1)}, \mathbf{a}_2^{(i+1)}]' = [\mathbf{b}_1^{(i)}, \mathbf{b}_2^{(i)}]$ est réduite au sens de Minkowski). Ainsi, le vecteur $\mathbf{a}_1^{(i+1)}$ ne peut pas être raccourci en lui ajoutant un multiple entier du vecteur $\mathbf{a}_2^{(i+1)}$, ce qui implique en particulier que $|\langle \mathbf{a}_2^{(i+1)}, \mathbf{a}_1^{(i+1)} \rangle| \leqslant \|\mathbf{a}_2^{(i+1)}\|^2/2$. Ainsi :

$$(1+\varepsilon)\|\mathbf{b}_4^{(i)}\|^2 = (1+\varepsilon)\|\mathbf{a}_1^{(i+1)}\|^2 \leqslant \|\mathbf{a}_2^{(i+1)}\|^2 \leqslant \|\mathbf{a}_4^{(i)}\|^2.$$

Pour finir, supposons maintenant que les vecteurs $[\mathbf{a}_1^{(i+1)}, \mathbf{a}_2^{(i+1)}]$ soient ε-G-réduits, et que la projection orthogonale de $\mathbf{a}_3^{(i+1)}$ sur l'espace engendré par $[\mathbf{a}_1^{(i+1)}, \mathbf{a}_2^{(i+1)}]$ ne soit pas dans $(1+\varepsilon)\mathrm{Vor}(\mathbf{a}_1^{(i+1)}, \mathbf{a}_2^{(i+1)})$. Nous distinguons deux sous-cas : $\pi_i = 1$ ou $\pi_i = 2$. Supposons d'abord que $\pi_i = 2$. Alors la base $[\mathbf{a}_1^{(i+1)}, \mathbf{a}_2^{(i+1)}]$ est réduite au sens de Minkowski, et les coordonnées de Voronoï possibles de $L(\mathbf{a}_1^{(i+1)}, \mathbf{a}_2^{(i+1)})$ sont les couples $(x_1, x_2) \in \{-1, 0, 1\}^2$ (d'après le lemme 23). Ainsi un vecteur $\mathbf{u}$ a sa projection orthogonale sur l'espace engendré par $[\mathbf{a}_1^{(i+1)}, \mathbf{a}_2^{(i+1)}]$ dans $\mathrm{Vor}(\mathbf{a}_1^{(i+1)}, \mathbf{a}_2^{(i+1)})$ si et seulement si :

$$\forall (x_1, x_2) \in \{-1, 0, 1\}^2, \|\mathbf{u}\| \leqslant \|\mathbf{u} + x_1\mathbf{a}_1^{(i+1)} + x_2\mathbf{a}_2^{(i+1)}\|.$$

Cela implique qu'il existe un couple $(x_1, x_2) \in \{-1, 0, 1\}^2$ tel que $|\langle \mathbf{a}_3^{(i+1)}, x_1\mathbf{a}_1^{(i+1)} + x_2\mathbf{a}_2^{(i+1)} \rangle| \geqslant \frac{1+\varepsilon}{2} \|x_1\mathbf{a}_1^{(i+1)} + x_2\mathbf{a}_2^{(i+1)}\|^2$. Dans le cas où $\pi_i = 1$, nous pouvons supposer que $\|\mathbf{a}_1^{(i+1)}\| \geqslant (1-\varepsilon)\|\mathbf{a}_2^{(i+1)}\|$, puisque si cette inégalité n'est pas satisfaite, $\|\mathbf{b}_4^{(i)}\| = \|\mathbf{a}_1^{(i+1)}\| \leqslant (1-\varepsilon)\|\mathbf{a}_4^{(i)}\|$. Nous verrons au paragraphe 2.8.2 (plus précisément au lemme 30) que pour un $\varepsilon > 0$ suffisamment petit, les coordonnées de Voronoï possibles d'une telle base ε-G-réduite sont les mêmes que pour une base réduite au sens de Minkowski. Par conséquent, comme dans le sous-cas précédent, il existe un couple $(x_1, x_2) \in \{-1, 0, 1\}^2$ tel que $|\langle \mathbf{a}_3^{(i+1)}, x_1\mathbf{a}_1^{(i+1)} + x_2\mathbf{a}_2^{(i+1)} \rangle| \geqslant \frac{1+\varepsilon}{2} \|x_1\mathbf{a}_1^{(i+1)} + x_2\mathbf{a}_2^{(i+1)}\|^2$. À partir de maintenant,

nous considérons les deux sous-cas simultanément. Supposons d'abord que $x_2 = 0$, ce qui implique que $|x_1| = 1$. Comme pour la dimension 3, le fait que le vecteur $\mathbf{a}_1^{(i+1)}$ ne puisse pas être raccourci en lui ajoutant un multiple entier du vecteur $\mathbf{a}_3^{(i+1)}$ donne le résultat (si $\pi_i = 1$ c'est automatique, et si $\pi_i = 2$, alors la base $[\mathbf{a}_1^{(i+1)}, \mathbf{a}_3^{(i+1)}] = [\mathbf{b}_1^{(i)}, \mathbf{b}_2^{(i)}]$ est réduite au sens de Minkowski). Le cas où $x_1 = 0$ se gère de la même manière. Il reste donc à analyser la situation où $|x_1| = |x_2| = 1$. Sans perte de généralité, on suppose $x_1 = x_2 = 1$. Nous avons :

$$|\langle \mathbf{a}_3^{(i+1)}, \mathbf{a}_1^{(i+1)} + \mathbf{a}_2^{(i+1)} \rangle| \geqslant \frac{1+\varepsilon}{2} \|\mathbf{a}_1^{(i+1)} + \mathbf{a}_2^{(i+1)}\|^2.$$

Puisque le vecteur $\mathbf{a}_{\pi_i}^{(i+1)}$ ne peut pas être raccourci en lui ajoutant une combinaison linéaire entière des deux autres vecteurs, nous avons $\|\mathbf{a}_3^{(i+1)} \pm (\mathbf{a}_1^{(i+1)} + \mathbf{a}_2^{(i+1)})\| \geqslant \|\mathbf{a}_{\pi_i}^{(i+1)}\| = \|\mathbf{b}_4^{(i)}\|$. En considérant le bon choix du « plus ou moins », et en utilisant l'inégalité $\|\mathbf{a}_3^{(i+1)}\| \leqslant \|\mathbf{a}_4^{(i)}\|$, on obtient :

$$\|\mathbf{b}_4^{(i)}\|^2 \leqslant \|\mathbf{a}_4^{(i)}\|^2 - 2|\langle \mathbf{a}_3^{(i+1)}, \mathbf{a}_1^{(i+1)} + \mathbf{a}_2^{(i+1)} \rangle| + \|\mathbf{a}_1^{(i+1)} + \mathbf{a}_2^{(i+1)}\|^2 \leqslant \|\mathbf{a}_4^{(i)}\|^2 - \varepsilon \|\mathbf{a}_1^{(i+1)} + \mathbf{a}_2^{(i+1)}\|^2.$$

Puisque la base $[\mathbf{a}_1^{(i+1)}, \mathbf{a}_2^{(i+1)}]$ est ε-G-réduite, nous avons aussi les inégalités :

$$\|\mathbf{a}_1^{(i+1)} + \mathbf{a}_2^{(i+1)}\|^2 \geqslant \|\mathbf{a}_1^{(i+1)}\|^2 + \|\mathbf{a}_2^{(i+1)}\|^2 - (1+\varepsilon)\|\mathbf{a}_1^{(i+1)}\|^2 \geqslant (1-\varepsilon)\|\mathbf{a}_2^{(i+1)}\|^2,$$

à partir desquelles on déduit : $(1 + \varepsilon(1-\varepsilon))\|\mathbf{b}_4^{(i)}\|^2 \leqslant \|\mathbf{a}_4^{(i)}\|^2$. □

Lemme 19. *Soit $2 \leqslant d \leqslant 4$. Il existe deux constantes $\varepsilon_2 > 0$ et $D > 0$ telles que la propriété suivante soit valide. Considérons la $(i+1)$-ème itération de boucle d'une exécution de l'algorithme glouton de réduction en dimension d. Supposons que $[\mathbf{a}_1^{(i+1)}, \ldots, \mathbf{a}_{d-1}^{(i+1)}]$ soit ε_2-G-réduite, et que $\|\mathbf{a}_k^{(i+1)}\| \geqslant (1-\varepsilon_2)\|\mathbf{a}_d^{(i+1)}\|$ pour un certain $k \in [|1, d-1|]$. Alors si $|x_k| \geqslant 2$ et si nous ne sommes pas dans le cas « 211 », on a :*

$$\|\mathbf{b}_d^{(i+1)}\|^2 + D\|\mathbf{b}_k^{(i+1)}\|^2 \leqslant \|\mathbf{a}_d^{(i+1)}\|^2,$$

où le cas « 211 » est la situation où $d = 4$, $|x_k| = 2$ et les deux autres $|x_j|$ valent 1. En particulier, si $\|\mathbf{b}_d^{(i+1)}\| \leqslant \|\mathbf{b}_k^{(i+1)}\|$, alors le vecteur $\mathbf{b}_d^{(i+1)}$ est significativement plus court que le vecteur $\mathbf{a}_d^{(i+1)}$:

$$\|\mathbf{b}_d^{(i+1)}\|^2 \leqslant \frac{1}{1+D}\|\mathbf{a}_d^{(i+1)}\|^2.$$

Ce dernier lemme est une conséquence directe du théorème de Pythagore et du lemme du vide, auquel la section 2.8 est dédiée. Le lemme du vide est illustré par la figure 2.10 : un vecteur de la partie non hachurée externe qui est transformé en un vecteur dans la zone non hachurée interne voit sa norme diminuer significativement.

Théorème 16 (Lemme du vide). *Soit $2 \leqslant d \leqslant 4$. Il existe deux constantes $\varepsilon > 0$ et $D > 0$ telles que la propriété suivante soit valide. Soient $[\mathbf{a}_1, \ldots, \mathbf{a}_{d-1}]$ des vecteurs ε-G-réduits, $\mathbf{u}$ un vecteur de $\mathrm{Vor}(\mathbf{a}_1, \ldots, \mathbf{a}_{d-1})$ et $x_1, \ldots, x_{d-1}$ des entiers. Si $\|\mathbf{a}_k\| \geqslant (1-\varepsilon)\|\mathbf{a}_{d-1}\|$ pour un certain $k \leqslant d-1$, alors :*

$$\|\mathbf{u}\|^2 + D\|\mathbf{a}_k\|^2 \leqslant \|\mathbf{u} + \sum_{j=1}^{d-1} x_j \mathbf{a}_j\|^2,$$

où $|x_k| \geqslant 2$, et si $d = 4$ et $|x_k| = 2$, alors les deux autres $|x_j|$ ne valent pas tous les deux 1.

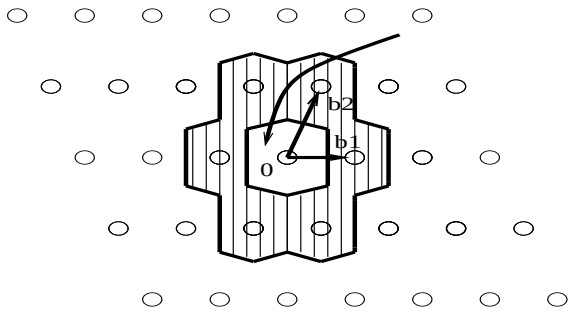


FIG. 2.10 – Le lemme du vide en dimension 2. Un vecteur de l'extérieur de la couronne qui est transformé en un vecteur de la cellule de Voronoï voit sa norme diminuer significativement.

Cela achève la description générale de la preuve de l'équation 2.1, page 71, et par conséquent du théorème 14. En effet, choisissons trois constantes $\varepsilon, D > 0$ et $C > 1$ telles que l'on puisse appliquer les lemmes 18 et 19. Nous montrons que l'équation (2.1) est vérifiée pour $K = \min(C, \sqrt{1+D}, \frac{1}{1-\varepsilon}) > 1$. Considérons une itération de boucle $(i+1)$ qui n'est pas la dernière et telle que $p_{i+1} \leqslant p_i$. Nous rappelons que parmi n'importe quelles d itérations de boucle successives, il y a au moins une telle itération de boucle. Pour cette itération, nous avons $|x_{\pi_i}^{(i+1)}| \geqslant 2$. Nous distinguons quatre situations :

- La base $[\mathbf{a}_1^{(i+1)}, \ldots, \mathbf{a}_{d-1}^{(i+1)}]$ n'est pas ε-G-réduite : alors le lemme 18 donne le résultat escompté grâce à la i-ème itération de boucle.
- On a $\|\mathbf{a}_{\pi_i}^{(i+1)}\| < (1-\varepsilon)\|\mathbf{a}_d^{(i+1)}\|$: puisque $p_{i+1} \leqslant p_i$, nous avons la suite d'inégalités $\|\mathbf{b}_d^{(i+1)}\| < \|\mathbf{a}_{p_i}^{(i+1)}\| = \|\mathbf{a}_{\pi_i}^{(i+1)}\| < (1-\varepsilon)\|\mathbf{a}_d^{(i+1)}\|$.
- Nous sommes dans le cas « 211 », c'est-à-dire $d = 4$, $|x_{\pi_i}^{(i+1)}| = 2$ et les deux autres $|x_j^{(i+1)}|$ valent 1, alors nous utilisons l'étude qui sera effectuée au paragraphe 2.7.2.
- Sinon, nous utilisons le lemme 19, qui donne le résultat escompté grâce à la $(i+1)$-ème itération de boucle.

2.7.2 Fin de l'analyse en dimension 4.

Dans les paragraphes qui précèdent, nous avons montré que l'algorithme glouton itératif de réduction a un nombre au plus linéaire d'itérations de boucle en dimensions 2 et 3, mais nous avons mis en évidence l'apparition d'une difficulté supplémentaire en dimension 4 : le lemme du vide ne permet pas de gérer le cas « 211 ». Cela vient du fait qu'en dimension 3 il existe des bases $[\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3]$ réduites au sens de Minkowski pour lesquelles $2\mathbf{b}_i + s_1\mathbf{b}_j + s_2\mathbf{b}_k$ est un vecteur de Voronoï, avec $\{i,j,k\} = \{1,2,3\}$ et $|s_1| = |s_2| = 1$. En effet, considérons le réseau engendré par les colonnes $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3$ de la matrice suivante :

$$M = \begin{bmatrix} 1 & 1 & -1 \\ 1 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Cette base est Minkowski-réduite et pour tous $k_1, k_2, k_3 \in \mathbb{Z}$ on a $\|\mathbf{b}_1 + \mathbf{b}_2 + 2\mathbf{b}_3\| = \|\mathbf{b}_1 + \mathbf{b}_2\| \leqslant \|(2k_1+1)\mathbf{b}_1 + (2k_2+1)\mathbf{b}_1 + 2k_3\mathbf{b}_3\|$. Ainsi, un vecteur du translaté de la

cellule de Voronoï centrée en $\mathbf{b}_1 + \mathbf{b}_2 + 2\mathbf{b}_3$ peut éventuellement ne pas être raccourci de manière significative quand il est « ramené » dans la cellule de Voronoï centrée en $\mathbf{0}$.

Le lemme du vide n'aide pas à résoudre cette difficulté. Néanmoins, nous pouvons remarquer que le triplet $(1,1,2)$ est très peu souvent une coordonnée de Voronoï (vis-à-vis d'une base réduite au sens de Minkowski), et quand c'est le cas, il ne peut pas être une coordonnée de Voronoï stricte : on peut prouver assez simplement que si $(1,1,2)$ est une coordonnée de Voronoï, alors $\|\mathbf{b}_1 + \mathbf{b}_2\| = \|\mathbf{b}_1 + \mathbf{b}_2 + 2\mathbf{b}_3\|$, ce qui signifie que le vecteur $\mathbf{b}_1 + \mathbf{b}_2 + 2\mathbf{b}_3$ n'est pas le seul dans son orbite de $L/2L$ à être de longueur minimale. En fait, il se trouve que le réseau engendré par les colonnes de la matrice M est à peu près le seul pour lequel $(1,1,2)$ — à tout changement de signes et permutation de colonnes près — peut être une coordonnée de Voronoï. Plus précisément, si $(1,1,2)$ — à tout changement de signes et permutation de colonnes près — est une coordonnée de Voronoï d'une base d'un réseau, alors la matrice de la base peut être écrite rUM où r est un réel strictement positif et U est une matrice orthogonale. Comme une base peut être arbitrairement proche d'une de ces bases critiques sans être l'une d'entre elles, nous avons besoin de considérer un petit ensemble compact de bases normalisées autour de ces bases gênantes. Plus précisément, cet ensemble compact est :

$$\left\{ [\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3] \text{ } \varepsilon\text{-G-réduits}, \exists \sigma \in \mathcal{S}_3, \left\| \frac{1}{\|\mathbf{b}_3\|^2} \left| G(\mathbf{b}_{\sigma(1)}, \mathbf{b}_{\sigma(2)}, \mathbf{b}_{\sigma(3)}) \right| - \left| M^t M \right| \right\|_\infty \leqslant \varepsilon \right\},$$

pour une constante $\varepsilon > 0$ suffisamment petite et où $\|M\|_\infty$ est le maximum des valeurs absolues de la matrice M et $|M|$ est la matrice dont chaque entrée de M a été remplacée par sa valeur absolue.

Désormais, considérons que nous sommes dans le cas « 211 » pour une certaine itération de boucle $i+1$. Nous distinguons trois cas :

- La base $[\mathbf{a}_1^{(i+1)}, \mathbf{a}_2^{(i+1)}, \mathbf{a}_3^{(i+1)}]$ est à l'extérieur du compact. Dans ce cas une variante du lemme du vide (le lemme 36), prouvé à la section 2.8, peut être utilisée pour montrer que $\mathbf{b}_4^{(i+1)}$ est significativement plus court que le vecteur $\mathbf{a}_4^{(i+1)}$.
- La base $[\mathbf{a}_1^{(i+1)}, \mathbf{a}_2^{(i+1)}, \mathbf{a}_3^{(i+1)}]$ est à l'intérieur du compact, mais la projection orthogonale de $\mathbf{a}_4^{(i+1)}$ sur l'espace engendré par $[\mathbf{a}_1^{(i+1)}, \mathbf{a}_2^{(i+1)}, \mathbf{a}_3^{(i+1)}]$ est loin de la cellule de Voronoï $\mathrm{Vor}(\mathbf{a}_1^{(i+1)}, \mathbf{a}_2^{(i+1)}, \mathbf{a}_3^{(i+1)})$. Dans ce cas, en utilisant le lemme 36, on obtient que le vecteur $\mathbf{b}_4^{(i+1)}$ est significativement plus court que le vecteur $\mathbf{a}_4^{(i+1)}$.
- Sinon, la géométrie de la base $[\mathbf{a}_1^{(i+1)}, \mathbf{a}_2^{(i+1)}, \mathbf{a}_3^{(i+1)}, \mathbf{a}_4^{(i+1)}]$ est connue de façon très précise et on peut montrer qu'il reste $O(1)$ itérations de boucle.

Plus précisément, en utilisant le lemme 36, on montre que :

Lemme 20. *Il existe deux constantes $K, \varepsilon > 0$ telles que la proposition suivante soit valide. Considérons une exécution de l'algorithme glouton de réduction en dimension 4, et une itération de boucle $i+1$ pour laquelle on a :*

- $p_{i+1} \leqslant p_i$,
- $|x_{\pi_i}| = 2$ *et* $(|x_{\sigma(1)}|, |x_{\sigma(2)}|, |x_{\sigma(3)}|) = (1,1,2)$ *pour un certain* $\sigma \in \mathcal{S}_3$,
- $[\mathbf{a}_1^{(i+1)}, \mathbf{a}_2^{(i+1)}, \mathbf{a}_3^{(i+1)}]$ *est ε-G-réduite,*
- $\|\mathbf{a}_{\pi_i}^{(i+1)}\| \geqslant (1-\varepsilon)\|\mathbf{a}_4^{(i+1)}\|$.

Alors, on a $\|\mathbf{a}_4^{(i+1)}\| \geqslant (1+K)\|\mathbf{b}_4^{(i+1)}\|$ *ou :*

$$\left\| \frac{1}{\|\mathbf{a}_4^{(i+1)}\|^2} \left| G(\mathbf{a}_{\sigma(1)}^{(i+1)}, \mathbf{a}_{\sigma(2)}^{(i+1)}, \mathbf{a}_{\sigma(3)}^{(i+1)}, \mathbf{a}_4^{(i+1)}) \right| - A \right\|_\infty \leqslant \varepsilon, \text{ avec } A = \begin{bmatrix} 1 & 0 & 1/2 & 0 \\ 0 & 1 & 1/2 & 0 \\ 1/2 & 1/2 & 1 & 1/2 \\ 0 & 0 & 1/2 & 1 \end{bmatrix}.$$

Pour prouver ce résultat, nous allons restreindre progressivement les configurations géométriques possibles de la base $[\mathbf{a}_1^{(i)}, \mathbf{a}_2^{(i)}, \mathbf{a}_3^{(i)}, \mathbf{a}_4^{(i)}]$. Remarquons que la configuration restante est celle du « root lattice » D_4. Ce dernier cas sera considéré au lemme 21.

Démonstration. La preuve s'appuie sur le lemme 36. Nous choisissons $\varepsilon, C > 0$ pour que ce lemme soit valide. La constante K dépendra de C et de ε. Nous montrons dans un premier temps que nous pouvons supposer sans perte de généralité que les vecteurs de la base ont des longueurs similaires, c'est-à-dire : $(1-\varepsilon)^2\|\mathbf{a}_4^{(i+1)}\| \leqslant \|\mathbf{a}_1^{(i+1)}\|$. Nous savons par hypothèse que $|x_{\pi_i}| = 2$ et que les deux autres $|x_j|$ valent 1. Nous savons aussi que : $\|\mathbf{a}_{\pi_i}^{(i+1)}\| \geqslant (1-\varepsilon)\|\mathbf{a}_4^{(i+1)}\|$. Si $\pi_i = 1$, c'est évident. Si $\pi_i = 2$, alors nous utilisons le lemme 36, 2), et si $\pi_i = 3$ nous utilisons le lemme 36, 1), dans les deux cas en utilisant aussi le théorème de Pythagore. Jusqu'ici, nous avons montré que nous sommes dans l'une des deux situations suivantes :

- $\|\mathbf{a}_4^{(i+1)}\|^2 \geqslant (1+C)\|\mathbf{b}_4^{(i+1)}\|^2$,
- $(1-\varepsilon)^2\|\mathbf{a}_4^{(i+1)}\| \leqslant \|\mathbf{a}_1^{(i+1)}\| \leqslant \|\mathbf{a}_4^{(i+1)}\|$.

Le reste de la preuve est le même quelle que soit la permutation appliquée au triplet $(2,1,1)$. Pour simplifier, nous supposerons donc que $(x_1, x_2, x_3) = (2,1,1)$. L'étape suivante de la preuve consiste à montrer que nous pouvons supposer que la matrice de Gram de $[\mathbf{a}_1^{(i+1)}, \mathbf{a}_2^{(i+1)}, \mathbf{a}_3^{(i+1)}]$ est proche de :

$$\|\mathbf{a}_4^{(i+1)}\|^2 \cdot \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & 1 & 0 \\ -\frac{1}{2} & 0 & 1 \end{bmatrix}.$$

Cela provient directement du lemme 36 et du théorème de Pythagore, qui impliquent que nous sommes dans au moins une des situations suivantes :

- $\|\mathbf{a}_4^{(i+1)}\|^2 \geqslant (1+C)\|\mathbf{b}_4^{(i+1)}\|^2$,
- $|\langle \mathbf{a}_2^{(i+1)}, \mathbf{a}_3^{(i+1)} \rangle| \leqslant \varepsilon\|\mathbf{a}_3^{(i+1)}\|^2$ et $\left| \langle \mathbf{a}_1^{(i+1)}, \mathbf{a}_j^{(i+1)} \rangle + \frac{1}{2}\|\mathbf{a}_3^{(i+1)}\|^2 \right| \leqslant \varepsilon\|\mathbf{a}_3^{(i+1)}\|^2$ pour $j = 2$ et $j = 3$.

Il ne reste donc qu'à s'intéresser aux produits scalaires $\langle \mathbf{a}_4^{(i+1)}, \mathbf{a}_k^{(i+1)} \rangle$ pour $k \in \{1,2,3\}$. Rappelons que la projection orthogonale de $\mathbf{b}_4^{(i+1)} = \mathbf{a}_4^{(i+1)} - \mathbf{a}_3^{(i+1)} - \mathbf{a}_2^{(i+1)} - 2\mathbf{a}_1^{(i+1)}$ sur l'espace engendré par la base $[\mathbf{a}_1^{(i+1)}, \mathbf{a}_2^{(i+1)}, \mathbf{a}_3^{(i+1)}]$ appartient à la cellule de Voronoï $\mathrm{Vor}(\mathbf{a}_1^{(i+1)}, \mathbf{a}_2^{(i+1)}, \mathbf{a}_3^{(i+1)})$. Sans perte de généralité, nous supposons que $\|\mathbf{a}_4^{(i+1)}\| \geqslant \|\mathbf{b}_4^{(i+1)}\| \geqslant (1-\varepsilon)\|\mathbf{a}_4^{(i+1)}\|$, puisque sinon nous avons le résultat pour $K = \frac{1}{1-\varepsilon}$. En développant $\|\mathbf{b}_4\|^2$, nous obtenons :

$$(1+19\varepsilon)\|\mathbf{a}_4^{(i+1)}\|^2 \geqslant 3\|\mathbf{a}_4^{(i+1)}\|^2 - 2\langle \mathbf{a}_4^{(i+1)}, 2\mathbf{a}_1^{(i+1)} + \mathbf{a}_2^{(i+1)} + \mathbf{a}_3^{(i+1)} \rangle \geqslant (1-19\varepsilon)\|\mathbf{a}_4^{(i+1)}\|^2,$$

où l'on a utilisé notre connaissance de la matrice de Gram de $[\mathbf{a}_1^{(i+1)}, \mathbf{a}_2^{(i+1)}, \mathbf{a}_3^{(i+1)}]$. Nous obtenons donc :

$$| \langle \mathbf{a}_4^{(i+1)}, 2\mathbf{a}_1^{(i+1)} + \mathbf{a}_2^{(i+1)} + \mathbf{a}_3^{(i+1)} \rangle - \|\mathbf{a}_4^{(i+1)}\| | \leqslant 19\varepsilon \|\mathbf{a}_4^{(i+1)}\|.$$

Cette dernière équation indique que, pour finir la preuve, il suffit de montrer que les produits scalaires $\langle \mathbf{a}_4^{(i+1)}, \mathbf{a}_2^{(i+1)} \rangle$ et $\langle \mathbf{a}_4^{(i+1)}, \mathbf{a}_3^{(i+1)} \rangle$ sont petits. Soit $j \in \{2, 3\}$. Par hypothèse, $\|\mathbf{a}_4^{(i+1)} - 2\mathbf{a}_1^{(i+1)} - x\mathbf{a}_j^{(i+1)}\| \geqslant \|\mathbf{a}_4^{(i+1)} - 2\mathbf{a}_1^{(i+1)} - \mathbf{a}_2^{(i+1)} - \mathbf{a}_3^{(i+1)}\|$ pour tout $x \in \mathbb{Z}$. En particulier, en choisissant $x = 0$ puis $x = 2$, en développant les normes et en utilisant notre connaissance de la matrice de Gram de $[\mathbf{a}_1^{(i+1)}, \mathbf{a}_2^{(i+1)}, \mathbf{a}_3^{(i+1)}]$, on peut obtenir explicitement un entier k tel que $|\langle \mathbf{a}_4^{(i+1)}, \mathbf{a}_1^{(i+1)} \rangle| \leqslant k\varepsilon \|\mathbf{a}_4^{(i+1)}\|$.

Soit $K = \min(1 + C, \frac{1}{(1-\varepsilon)^2})$. Nous avons montré qu'il existe une constante $K' = \max(K, 19, k) > 0$ telle que :

$$\left\| \frac{1}{\|\mathbf{a}_4^{(i+1)}\|^2} G(\mathbf{a}_1^{(i+1)}, \mathbf{a}_2^{(i+1)}, \mathbf{a}_3^{(i+1)}, \mathbf{a}_4^{(i+1)}) - A \right\|_\infty \leqslant K'\varepsilon.$$

Cela finit la preuve du lemme. □

À ce moment de l'analyse du cas « 211 », nous avons prouvé que l'on peut supposer que la géométrie de la base $[\mathbf{a}_1^{(i+1)}, \mathbf{a}_2^{(i+1)}, \mathbf{a}_3^{(i+1)}, \mathbf{a}_4^{(i+1)}]$ est très particulière : sa matrice de Gram est extrêmement proche de la matrice A. Nous nous attaquons à ce dernier cas avec le lemme suivant, qui explique que si la matrice de Gram d'une base est suffisamment proche d'une matrice inversible donnée, alors le nombre de vecteurs courts dans le réseau engendré par la base est borné. Puisque l'algorithme glouton de réduction produit des bases toujours plus petites pour l'ordre lexicographique des longueurs, si la matrice de Gram de la base courante est proche de A, alors il ne reste que $O(1)$ itérations de boucle.

Lemme 21. *Soient A une matrice $d \times d$ inversible, et $B > 0$. Alors il existe $\varepsilon, N > 0$ tels que pour n'importe quelle base $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$, si $\left\| \frac{1}{\|\mathbf{b}_d\|^2} G(\mathbf{b}_1, \ldots, \mathbf{b}_d) - A \right\|_\infty \leqslant \varepsilon$, alors :*

$$|\{(x_1, \ldots, x_d), \|x_1\mathbf{b}_1 + \cdots + x_d\mathbf{b}_d\| \leqslant B\|\mathbf{b}_d\|\}| \leqslant N.$$

Démonstration. Soit $\varepsilon > 0$ tel que pour toute matrice G vérifiant $\|G - A\|_\infty \leqslant \varepsilon$, la matrice G est inversible (un tel ε existe puisque l'ensemble des matrices inversibles est ouvert). Dans ce cas, si $X = (x_1, \ldots, x_d)$, alors :

$$\left| \frac{1}{\|\mathbf{b}_d\|^2} \|x_1\mathbf{b}_1 + \cdots + x_d\mathbf{b}_d\|^2 - XAX^t \right| = |X(G - A)X^t| \leqslant d^2\varepsilon(XX^t),$$

où $G = \frac{1}{\|\mathbf{b}_d\|^2} G(\mathbf{b}_1, \ldots, \mathbf{b}_d)$. Par conséquent, si $\|x_1\mathbf{b}_1 + \cdots + x_d\mathbf{b}_d\| \leqslant B\|\mathbf{b}_d\|$, alors, en utilisant l'inégalité triangulaire, on obtient $|XAX^t| \leqslant B^2 + d^2\varepsilon(XX^t)$. Mais $|XAX^t| \geqslant \frac{1}{\|A^{-1}\|}(XX^t)$, où $\|A\|$ est défini par $\|B\| = \max(YBY^t, Y \in \mathbb{R}^n \text{ et } \|Y\| = 1)$, ce qui est strictement positif. Ainsi :

$$(XX^t) \cdot \left(\frac{1}{\|A^{-1}\|} - d^2\varepsilon \right) \leqslant B^2.$$

On choisit $\varepsilon < \frac{1}{d^2\|A^{-1}\|}$. Les x_i sont des entiers tels que (XX^t) est borné, on est donc en train de considérer les points à coordonnées entières à l'intérieur d'une hypersphère de dimension d. Il n'y a qu'un nombre fini de tels points. □

2.8 Résultats sur la géométrie des réseaux en petite dimension.

Dans cette section, nous donnons quelques résultats à propos des cellules de Voronoï en dimensions 2 et 3, qui sont cruciaux dans l'analyse de complexité de l'algorithme glouton de réduction décrit plus haut. Plus précisément, l'analyse de complexité s'appuie fortement sur le lemme du vide, qui provient de l'étude des cellules de Voronoï pour des vecteurs ε-G-réduits (faite au paragraphe 2.8.2), qui s'appuie elle-même sur l'étude des cellules de Voronoï des base réduites au sens de Minkowski (faite au paragraphe 2.8.1).

2.8.1 Les cellules de Voronoï pour des bases réduites au sens de Minkowski.

Nous commençons en donnant des bornes simples sur le diamètre d'une cellule de Voronoï et sur l'orthogonalisation de Gram-Schmidt d'une base réduite au sens de Minkowski :

Lemme 22. *Soit $d \in [|1,5|]$. Soit $[\mathbf{b}_1,\ldots,\mathbf{b}_d]$ une base d'un réseau L. Alors $\rho(L) \leqslant \frac{\sqrt{d}}{2}\|\mathbf{b}_d\|$. Ainsi, si $[\mathbf{b}_1,\ldots,\mathbf{b}_d]$ est une base Minkowski-réduite, on a $\|\mathbf{b}_d^*\| \geqslant \frac{\sqrt{5-d}}{2}\|\mathbf{b}_d\|$.*

Démonstration. La première partie du lemme est très classique et diverses preuves existent dans la littérature. Ici nous en donnons une qui repose sur les bornes de transfert (« transferance bounds »). Sans perte de généralité nous supposons que $\|\mathbf{b}_d\| = 1$. Nous avons la borne de transfert [99] : $\rho(L)\cdot\lambda_1(L^*) \leq \sqrt{d}/2$, où L^* est le réseau dual de L, c'est-à-dire $(\forall \mathbf{b}\in L^*)(\forall i \leqslant d)(\langle \mathbf{b}, \mathbf{b}_i\rangle \in \mathbb{Z})$. Par définition de L^* et à cause du fait que $\mathbf{b}_1,\ldots,\mathbf{b}_d$ sont de normes plus petites que 1, nous avons $\lambda_1(L^*) \geqslant 1$, ce qui donne la borne sur $\rho(L)$.

Supposons maintenant que $[\mathbf{b}_1,\ldots,\mathbf{b}_d]$ soit une base réduite au sens de Minkowski. Alors la projection orthogonale de $\mathbf{b}_d$ sur l'espace engendré par $[\mathbf{b}_1,\ldots,\mathbf{b}_{d-1}]$ est dans la cellule de Voronoï $\mathrm{Vor}(\mathbf{b}_1,\ldots,\mathbf{b}_{d-1})$. En appliquant le théorème de Pythagore, on obtient que : $\|\mathbf{b}_d^*\|^2 \geqslant \|\mathbf{b}_d\|^2 - \frac{d-1}{4}\|\mathbf{b}_{d-1}\|^2$. Le fait que $\|\mathbf{b}_{d-1}\| \leqslant \|\mathbf{b}_d\|$ finit la preuve. □

Le lemme suivant donne la liste des coordonnées de Voronoï possibles d'un réseau de dimension 2 donné par une base Minkowski-réduite[10] Une telle base a des vecteurs de Voronoï dont les coordonnées sont petites :

Lemme 23. *En dimension 2, les coordonnées de Voronoï possibles sont $(1,0)$ et $(1,1)$, auxquelles on peut appliquer tout changement de signes et toute permutation de coordonnées. Ainsi, les coordonnées possibles sont les couples $(\varepsilon_1,\varepsilon_2)$ non nuls avec $|\varepsilon_1|,|\varepsilon_2| \leqslant 1$.*

La preuve repose sur une analyse précise de l'expression $\|(2x_1+\varepsilon_1)\cdot\mathbf{b}_1 + (2x_2+\varepsilon_2)\cdot\mathbf{b}_2\|^2 - \|\varepsilon_1\mathbf{b}_1+\varepsilon_2\mathbf{b}_2\|^2$, où $[\mathbf{b}_1,\mathbf{b}_2]$ est Minkowski-réduite, $\varepsilon_1,\varepsilon_2 \in \{0,1\}$ et $x_1,x_2\in\mathbb{Z}$. En effet, puisque les coordonnées de Voronoï d'un réseau L sont données par les minima des orbites non nulles de $L/2L$, il suffit de montrer que si $x_1 \neq 0$ ou $x_2 \neq 0$, alors cette expression est strictement positive.

[10] Nous rappelons que nous ne pouvons pas utiliser directement le théorème 9, page 55, car celui-ci ne concerne que les coordonnées de Voronoï strictes.

Démonstration. Rappelons que les coordonnées de Voronoï possibles peuvent être obtenues en considérant les représentants courts des éléments de $L/2L$, quand on dispose d'une base Minkowski-réduite de L. Soit $[\mathbf{b}_1, \mathbf{b}_2]$ une base réduite au sens de Minkowski. En remplaçant $\mathbf{b}_i$ par son opposé $-\mathbf{b}_i$ pour $i \in \{1, 2\}$, il suffit clairement de montrer que pour tout $x_1, x_2 \geqslant 0$, et pour tout $\varepsilon_1, \varepsilon_2 \in \{0, 1\}$, si $x_1 \geqslant 1$ ou $x_2 \geqslant 1$, alors :

$$\|(2x_1 + \varepsilon_1)\mathbf{b}_1 + (2x_2 + \varepsilon_2)\mathbf{b}_2\|^2 > \|\varepsilon_1\mathbf{b}_1 + \varepsilon_2\mathbf{b}_2\|^2.$$

Avant tout,

$$\begin{aligned}\|(2x_1 + \varepsilon_1)\mathbf{b}_1 + (2x_2 + \varepsilon_2)\mathbf{b}_2\|^2 - \|\varepsilon_1\mathbf{b}_1 + \varepsilon_2\mathbf{b}_2\|^2 \\ = ((2x_1 + \varepsilon_1)^2 - \varepsilon_1^2)\|\mathbf{b}_1\|^2 + ((2x_2 + \varepsilon_2)^2 - \varepsilon_2^2)\|\mathbf{b}_2\|^2 \\ +2((2x_1 + \varepsilon_1)(2x_2 + \varepsilon_2) - \varepsilon_1\varepsilon_2)\langle\mathbf{b}_1, \mathbf{b}_2\rangle.\end{aligned}$$

Puisque $x_1, x_2 \geqslant 0$, nous avons $(2x_2 + \varepsilon_2)^2 - \varepsilon_2^2 \geqslant 0$ et $(2x_1 + \varepsilon_1)(2x_2 + \varepsilon_2) - \varepsilon_1\varepsilon_2 \geqslant 0$. De plus, $[\mathbf{b}_1, \mathbf{b}_2]$ est réduite et donc $\|\mathbf{b}_2\| \geqslant \|\mathbf{b}_1\|$ et $2\langle\mathbf{b}_1, \mathbf{b}_2\rangle \geqslant -\|\mathbf{b}_1\|^2$. À partir de ces faits, nous obtenons :

$$\begin{aligned}\|(2x_1 + \varepsilon_1)\mathbf{b}_1 + (2x_2 + \varepsilon_2)\mathbf{b}_2\|^2 - \|\varepsilon_1\mathbf{b}_1 + \varepsilon_2\mathbf{b}_2\|^2 \\ \geqslant 2[2x_1^2 + 2x_2^2 - 2x_1x_2 + x_1(2\varepsilon_1 - \varepsilon_2) + x_2(2\varepsilon_2 - \varepsilon_1)]\|\mathbf{b}_1\|^2.\end{aligned}$$

Cette dernière expression est strictement positive à partir du moment où $(x_1, x_2) \neq (0, 0)$. En effet :

- si $\varepsilon_1 = \varepsilon_2 = 0$, le facteur est $4((x_1 - x_2)^2 + x_1x_2)$,
- si $\varepsilon_1 = 0$ et $\varepsilon_2 = 1$, le facteur est $2(x_2^2 + 2x_2 + (x_2 - x_1)^2 + (x_1^2 - x_1))$,
- le cas $\varepsilon_1 = 1$ et $\varepsilon_2 = 0$ est symétrique,
- si $\varepsilon_1 = \varepsilon_2 = 1$, le facteur est $2((x_2 - x_1)^2 + (x_2^2 + x_2) + (x_1^2 + x_1))$.

□

Nous généralisons cette analyse en dimension 3. Les idées sous-jacentes de la preuve sont les mêmes, mais comme le nombre de variables augmente, l'analyse devient plus technique.

Lemme 24. *En dimension 3, les coordonnées de Voronoï possibles sont* $(1, 0, 0)$, $(1, 1, 0)$, $(1, 1, 1)$ *et* $(2, 1, 1)$, *auxquelles on peut appliquer tout changement de signes et toute permutation de coordonnées.*

Démonstration. Nous généralisons la preuve du lemme 23. En effet, nous montrons que pour tous $x_1, x_2, x_3 \in \mathbb{Z}$ et tous $\varepsilon_1, \varepsilon_2, \varepsilon_3 \in \{0, 1\}$, si $(2x_1 + \varepsilon_1, 2x_2 + \varepsilon_2, 2x_3 + \varepsilon_3)$ n'est pas dans la liste attendue des coordonnées de Voronoï, alors :

$$\|(2x_1 + \varepsilon_1)\mathbf{b}_1 + (2x_2 + \varepsilon_2)\mathbf{b}_2 + (2x_3 + \varepsilon_3)\mathbf{b}_3\|^2 - \|\varepsilon_1\mathbf{b}_1 + \varepsilon_2\mathbf{b}_2 + \varepsilon_3\mathbf{b}_3\|^2 > 0.$$

En remplaçant éventuellement le vecteur $\mathbf{b}_i$ par le vecteur $-\mathbf{b}_i$, la preuve peut être restreinte au cas où $x_1, x_2, x_3 \geqslant 0$. De plus, comme nous avons déjà considéré le cas de la dimension 2 dans le lemme 23, nous pouvons supposer que pour tout $i \in \{1, 2, 3\}$, nous ayons $(x_i, \varepsilon_i) \neq (0, 0)$.

Grâce au lemme 22, nous savons que puisque $[\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3]$ est réduite au sens de Minkowski, $\|\mathbf{b}_3^*\| \geqslant \|\mathbf{b}_3\|/\sqrt{2}$. Ainsi, si $2x_3 + \varepsilon_3 \geqslant 5$, alors :

$$\|(2x_1 + \varepsilon_1)\mathbf{b}_1 + (2x_2 + \varepsilon_2)\mathbf{b}_2 + (2x_3 + \varepsilon_3)\mathbf{b}_3\|^2 \geqslant 25\|\mathbf{b}_3^*\|^2 \geqslant \frac{25}{2}\|\mathbf{b}_3\|^2,$$

et l'inégalité triangulaire donne que $\|\varepsilon_1\mathbf{b}_1+\varepsilon_2\mathbf{b}_2+\varepsilon_3\mathbf{b}_3\|^2 \leqslant 9\|\mathbf{b}_3\|^2$. Cela fournit le résultat quand $2x_3+\varepsilon_3 \geqslant 5$. Le même argument tient pour $(x_3,\varepsilon_3)=(2,0)$, et pour $(x_3,\varepsilon_3) \in \{(1,1),(1,0)\}$ avec $\varepsilon_1 \cdot \varepsilon_2 = 0$. Par conséquent, il ne reste plus qu'à considérer les trois cas suivants : $(x_3,\varepsilon_3)=(1,1)$ avec $\varepsilon_1=\varepsilon_2=1$; $(x_3,\varepsilon_3)=(1,0)$ avec $\varepsilon_1=\varepsilon_2=1$; et $(x_3,\varepsilon_3)=(0,1)$.

Cas 1 : Supposons que $(x_3,\varepsilon_3)=(1,1)$ et que $\varepsilon_1=\varepsilon_2=1$. Comme $[\mathbf{b}_1,\mathbf{b}_2,\mathbf{b}_3]$ est réduite au sens de Minkowski, nous avons $\langle \mathbf{b}_i,\mathbf{b}_j\rangle \geqslant -\|\mathbf{b}_i\|^2/2$ pour tout $1 \leqslant i < j \leqslant 3$, ce qui donne :

$$\begin{aligned}\|(2x_1+1)\mathbf{b}_1+(2x_2+1)\mathbf{b}_2+3\mathbf{b}_3\|^2 &- \|\mathbf{b}_1+\mathbf{b}_2+\mathbf{b}_3\|^2\\ &= 4(x_1^2+x_1)\|\mathbf{b}_1\|^2+4(x_2^2+x_2)\|\mathbf{b}_2\|^2+8\|\mathbf{b}_3\|^2\\ &\quad +4(3x_1+1)\langle\mathbf{b}_1,\mathbf{b}_3\rangle+4(3x_2+1)\langle\mathbf{b}_2,\mathbf{b}_3\rangle+4(2x_1x_2+x_1+x_2)\langle\mathbf{b}_1,\mathbf{b}_2\rangle\\ &\geqslant (4x_1^2-4x_1x_2-4x_1-2x_2-2)\|\mathbf{b}_1\|^2+(4x_2^2-2x_2-2)\|\mathbf{b}_2\|^2+8\|\mathbf{b}_3\|^2.\end{aligned}$$

Si $x_2=0$, il suffit de trouver une borne inférieure pour la quantité $(x_1^2-x_1)\|\mathbf{b}_1\|^2+\|\mathbf{b}_3\|^2$, qui est toujours plus grande que $\|\mathbf{b}_3\|^2$, et est donc strictement positive. Supposons maintenant que $x_2 \geqslant 1$. Alors $4x_2^2-2x_2-2 \geqslant 0$ et nous obtenons :

$$\begin{aligned}\|(2x_1+1)\mathbf{b}_1+(2x_2+1)\mathbf{b}_2+3\mathbf{b}_3\|^2 &- \|\mathbf{b}_1+\mathbf{b}_2+\mathbf{b}_3\|^2\\ &\geqslant 4[x_1^2+x_2^2-x_1x_2-x_1-x_2+1]\|\mathbf{b}_1\|^2\\ &\geqslant 4[(x_1-x_2)^2+(x_1x_2-x_1-x_2)+1]\|\mathbf{b}_1\|^2.\end{aligned}$$

Il est évident que cette dernière expression est strictement positive pour tout $x_1,x_2 \geqslant 0$ sauf quand $x_1=x_2=1$. Dans cette dernière situation, nous utilisons le fait que $\|3\mathbf{b}_1+3\mathbf{b}_2+3\mathbf{b}_3\|^2-\|\mathbf{b}_1+\mathbf{b}_2+\mathbf{b}_3\|^2=8\|\mathbf{b}_1+\mathbf{b}_2+\mathbf{b}_3\|^2$.

Cas 2 : Supposons maintenant que $(x_3,\varepsilon_3)=(1,0)$ et que $\varepsilon_1=\varepsilon_2=1$. De manière similaire, nous avons :

$$\begin{aligned}\|(2x_1+1)\mathbf{b}_1+(2x_2+1)\mathbf{b}_2+2\mathbf{b}_3\|^2 &- \|\mathbf{b}_1+\mathbf{b}_2\|^2\\ &= 4(x_1^2+x_1)\|\mathbf{b}_1\|^2+4(x_2^2+x_2)\|\mathbf{b}_2\|^2+4\|\mathbf{b}_3\|^2\\ &\quad +4(2x_1+1)\langle\mathbf{b}_1,\mathbf{b}_3\rangle+4(2x_2+1)\langle\mathbf{b}_2,\mathbf{b}_3\rangle+4(2x_1x_2+x_1+x_2)\langle\mathbf{b}_1,\mathbf{b}_2\rangle\\ &\geqslant (4x_1^2-4x_1x_2-2x_1-2x_2-2)\|\mathbf{b}_1\|^2+(4x_2^2-2)\|\mathbf{b}_2\|^2+4\|\mathbf{b}_3\|^2.\end{aligned}$$

Si $x_2=0$, il suffit de trouver une borne inférieure pour la quantité $(2x_1^2-x_1-1)\|\mathbf{b}_1\|^2+\|\mathbf{b}_3\|^2$, qui est strictement positive si $x_1 \geqslant 1$. Si $x_1=0$, alors nous obtenons l'une des coordonnées possibles de Voronoï listées. Supposons maintenant que $x_2 \geqslant 1$. Dans ce cas nous avons $4x_2^2-2 \geqslant 0$, ce qui donne :

$$\begin{aligned}\|(2x_1+1)\mathbf{b}_1+(2x_2+1)\mathbf{b}_2+2\mathbf{b}_3\|^2 &- \|\mathbf{b}_1+\mathbf{b}_2\|^2\\ &\geqslant 2[2x_1^2+2x_2^2-2x_1x_2-x_1-x_2]\|\mathbf{b}_1\|^2\\ &\geqslant 2[(x_1-x_2)^2+(x_1^2-x_1)+(x_2^2-x_2)]\|\mathbf{b}_1\|^2.\end{aligned}$$

Si $x_1 \geqslant 2$ ou $x_2 \geqslant 2$ ou $x_1 \neq x_2$, cette dernière quantité est strictement positive. Par conséquent il reste à considérer le cas $x_1=x_2=1$. Nous avons :

$$\|3\mathbf{b}_1+3\mathbf{b}_2+2\mathbf{b}_3\|^2-\|\mathbf{b}_1+\mathbf{b}_2\|^2=4[\|\mathbf{b}_3\|^2+3\langle\mathbf{b}_3,\mathbf{b}_1+\mathbf{b}_2\rangle+2\|\mathbf{b}_1+\mathbf{b}_2\|^2].$$

Comme $[\mathbf{b}_1,\mathbf{b}_2,\mathbf{b}_3]$ est réduite, nous avons $\langle\mathbf{b}_3,\mathbf{b}_1+\mathbf{b}_2\rangle \geqslant -\|\mathbf{b}_1+\mathbf{b}_2\|^2/2$, ce qui donne le résultat escompté.

Cas 3 : Supposons désormais que $(x_3,\varepsilon_3)=(0,1)$. De manière similaire, nous avons les

inégalités :

$$\begin{aligned}\|(2x_1+\varepsilon_1)\mathbf{b}_1+(2x_2+\varepsilon_2)\mathbf{b}_2+\mathbf{b}_3\|^2-\|\varepsilon_1\mathbf{b}_1+\varepsilon_2\mathbf{b}_2+\mathbf{b}_3\|^2\\ =4(x_1^2+\varepsilon_1x_1)\|\mathbf{b}_1\|^2+4(x_2^2+\varepsilon_2x_2)\|\mathbf{b}_2\|^2+4x_1\langle\mathbf{b}_1,\mathbf{b}_3\rangle\\ +4x_2\langle\mathbf{b}_2,\mathbf{b}_3\rangle+4(2x_1x_2+x_1\varepsilon_2+x_2\varepsilon_1)\langle\mathbf{b}_1,\mathbf{b}_2\rangle\\ \geqslant[4x_1^2-4x_1x_2+(4\varepsilon_1-2-2\varepsilon_2)x_1-2x_2\varepsilon_1]\|\mathbf{b}_1\|^2+[4x_2^2+(4\varepsilon_2-2)x_2]\|\mathbf{b}_2\|^2.\end{aligned}$$

Si $x_2=0$, il suffit de trouver une borne inférieure pour la quantité $4x_1^2+(4\varepsilon_1-2-2\varepsilon_2)x_1$. Celle-ci est strictement positive dès que $x_1\geqslant 1$ sauf dans le cas $(x_1,\varepsilon_1,\varepsilon_2)=(1,0,1)$, ce qui correspond à l'une des coordonnées de Voronoï possibles. Si $x_1=0$, nous avons des coordonnées de Voronoï possibles. Supposons maintenant que $x_2\geqslant 1$. Dans ce cas $4x_2^2+(4\varepsilon_2-2)x_2\geqslant 0$ et :

$$\begin{aligned}\|(2x_1+\varepsilon_1)\mathbf{b}_1+(2x_2+\varepsilon_2)\mathbf{b}_2+\mathbf{b}_3\|^2-\|\varepsilon_1\mathbf{b}_1+\varepsilon_2\mathbf{b}_2+\mathbf{b}_3\|^2\\ \geqslant[4x_1^2+4x_2^2-4x_1x_2+(4\varepsilon_1-2-2\varepsilon_2)x_1+(4\varepsilon_2-2-2\varepsilon_1)x_2]\|\mathbf{b}_1\|^2.\end{aligned}$$

Pour $(\varepsilon_1,\varepsilon_2)=(0,0)$, nous obtenons $2[(x_1-x_2)^2+(x_1^2-x_1)+(x_2^2-x_2)]$, ce qui est strictement positif dès que $x_1\neq x_2$ ou $x_1\geqslant 2$ ou $x_2\geqslant 2$. Le seul cas restant qui ne donne pas une des coordonnées listées est $x_1=x_2=1$. Remarquons que $\|2\mathbf{b}_1+2\mathbf{b}_2+\mathbf{b}_3\|^2>\|\mathbf{b}_3\|^2$ est équivalent à $\|\mathbf{b}_1+\mathbf{b}_2\|^2+\langle\mathbf{b}_1+\mathbf{b}_2,\mathbf{b}_3\rangle>0$, qui est impliqué par $\langle\mathbf{b}_1+\mathbf{b}_2,\mathbf{b}_3\rangle\geqslant-\|\mathbf{b}_1+\mathbf{b}_2\|^2/2$ (le vecteur $\mathbf{b}_3$ a sa projection orthogonale sur l'espace engendré par $[\mathbf{b}_1,\mathbf{b}_2]$ dans la cellule de Voronoï $\mathrm{Vor}(\mathbf{b}_1,\mathbf{b}_2)$).

Pour $(\varepsilon_1,\varepsilon_2)=(1,0)$, nous obtenons $2(x_1-x_2)^2+2(x_2^2-2x_2)+2x_1^2+2x_1$. Si $x_2\geqslant 2$, ceci est strictement positif. Si $x_2=1$, nous obtenons $4x_1^2-2x_1$, qui est strictement positif sauf si $x_1=0$, qui donne l'une des coordonnées de Voronoï possibles. Nous avons déjà considéré le cas $x_2=0$. Le cas $(\varepsilon_1,\varepsilon_2)=(0,1)$ est symétrique.

Finalement, si $(\varepsilon_1,\varepsilon_2)=(1,1)$, nous obtenons $2[(x_1-x_2)^2+x_1^2+x_2^2]$, ce qui est strictement positif à moins que $x_1=x_2=0$ (qui donne l'une des coordonnées de Voronoï possibles). Ceci achève la preuve du lemme. □

Comme nous l'avons vu, la coordonnée de Voronoï possible $(2,1,1)$ provoque des difficultés pour l'analyse de l'algorithme glouton de réduction en dimension 4 parce qu'elle contient un « 2 », ce qui ne peut pas être géré par l'argument que l'algorithme est glouton, comme pour les « 1 ». Nous résolvons ce problème de la manière suivante : nous montrons que lorsque $(2,1,1)$ est une coordonnée de Voronoï, le réseau a une forme très particulière, pour laquelle le comportement de l'algorithme est bien compris.

Lemme 25. *Supposons que* $[\mathbf{b}_1,\mathbf{b}_2,\mathbf{b}_3]$ *soit réduite au sens de Minkowski.*

1. *Si* $(s_1,s_2,2)$ *est une coordonnée de Voronoï avec* $s_i=\pm1$ *pour* $i\in\{1,2\}$, *alors* $\|\mathbf{b}_1\|=\|\mathbf{b}_2\|=\|\mathbf{b}_3\|$, $\langle\mathbf{b}_1,\mathbf{b}_2\rangle=0$ *et* $\langle\mathbf{b}_i,\mathbf{b}_3\rangle=-\frac{s_i}{2}\|\mathbf{b}_1\|^2$ *pour* $i\in\{1,2\}$.
2. *Si* $(s_1,2,s_3)$ *est une coordonnée de Voronoï avec* $s_i=\pm1$ *pour* $i\in\{1,3\}$, *alors* $\|\mathbf{b}_1\|=\|\mathbf{b}_2\|$. *De plus, si* $\|\mathbf{b}_1\|=\|\mathbf{b}_2\|=\|\mathbf{b}_3\|$, *alors* $\langle\mathbf{b}_1,\mathbf{b}_3\rangle=0$ *et* $\langle\mathbf{b}_i,\mathbf{b}_2\rangle=-\frac{s_i}{2}\|\mathbf{b}_1\|^2/2$ *pour* $i\in\{1,3\}$.
3. *Si* $(2,s_2,s_3)$ *est une coordonnée de Voronoï avec* $s_i=\pm1$ *pour* $i\in\{2,3\}$ *et si* $\|\mathbf{b}_1\|=\|\mathbf{b}_2\|=\|\mathbf{b}_3\|$, *alors* $\langle\mathbf{b}_2,\mathbf{b}_3\rangle=0$ *et* $\langle\mathbf{b}_i,\mathbf{b}_1\rangle=-\frac{s_i}{2}\|\mathbf{b}_1\|^2$ *pour* $i\in\{2,3\}$.

Démonstration. Sans perte de généralité nous supposons que pour tout i, on ait $s_i=1$. Dans la première situation, nous considérons l'inégalité $\|\mathbf{b}_1+\mathbf{b}_2+2\mathbf{b}_3\|^2\leqslant\|\mathbf{b}_1+\mathbf{b}_2\|^2$: c'est équivalent à $\|\mathbf{b}_3\|^2+\langle\mathbf{b}_1,\mathbf{b}_3\rangle+\langle\mathbf{b}_2,\mathbf{b}_3\rangle\leqslant 0$. Puisque la base est réduite, nous

avons $\langle \mathbf{b}_1, \mathbf{b}_3 \rangle \geqslant -\|\mathbf{b}_1\|^2/2$ et $\langle \mathbf{b}_2, \mathbf{b}_3 \rangle \geqslant -\|\mathbf{b}_2\|^2/2$. Ainsi, $2\|\mathbf{b}_3\|^2 - \|\mathbf{b}_1\|^2 - \|\mathbf{b}_2\|^2 \leqslant 0$. Par conséquent, on a l'égalité des longueurs et les inégalités sur les produits scalaires ci-dessus sont des égalités. Il reste à montrer que $\langle \mathbf{b}_1, \mathbf{b}_2 \rangle = 0$. Comme $\mathbf{b}_1$ est un plus court vecteur, nous avons : $\|\mathbf{b}_3 + \mathbf{b}_2 + \mathbf{b}_1\|^2 \geqslant \|\mathbf{b}_1\|^2$, ce qui est équivalent à $\langle \mathbf{b}_1, \mathbf{b}_2 \rangle \geqslant 0$. De plus, en développant l'inégalité $\|\mathbf{b}_1 + \mathbf{b}_2 + 2\mathbf{b}_3\|^2 \leqslant \|\mathbf{b}_1 - \mathbf{b}_2\|^2$, nous avons $\langle \mathbf{b}_1, \mathbf{b}_2 \rangle \leqslant 0$.

Dans la seconde situation, le développement de $\|\mathbf{b}_1 + 2\mathbf{b}_2 + \mathbf{b}_3\|^2 \leqslant \|\mathbf{b}_1 + \mathbf{b}_3\|^2$ donne l'égalité des longueurs. Dans le cas de l'hypothèse supplémentaire $\|\mathbf{b}_1\| = \|\mathbf{b}_2\| = \|\mathbf{b}_3\|$, la base $[\mathbf{b}_1, \mathbf{b}_3, \mathbf{b}_2]$ est aussi réduite, et on peut utiliser la première partie du lemme. Ce dernier argument permet aussi de prouver le troisième point du lemme. □

2.8.2 Les cellules de Voronoï pour des vecteurs ε-G-réduits.

Nous généralisons ici les résultats du paragraphe précédent au cas de vecteurs ε-G-réduits. L'idée générale est qu'en rendant compact l'ensemble des bases Minkowski-réduites et qu'en le grossissant légèrement, les coordonnées de Voronoï possibles restent les mêmes. Une difficulté vient du fait qu'en faisant cela, certains vecteurs que nous considérons peuvent être nuls, ce qui donne alors une infinité de coordonnées de Voronoï : par exemple, en dimension 2, si $\mathbf{b}_1 = \mathbf{0}$, alors tout couple $(x_1, 0)$ est une coordonnée de Voronoï de $[\mathbf{b}_1, \mathbf{b}_2]$. Pour contourner cette difficulté nous nous restreignons au cas où les vecteurs $\mathbf{b}_i$ ont des longueurs similaires. Plus précisément, nous utilisons ce que nous appelons le lemme Topologique : si on peut garantir que les coordonnées de Voronoï possibles de l'espace de bases agrandi sont bornées, alors pour un agrandissement suffisamment petit, les coordonnées de Voronoï possibles restent les mêmes. Avant ces considérations, nous donnons des résultats simples sur les ensembles de vecteurs ε-G-réduits et sur leurs orthogonalisations de Gram-Schmidt. Puis nous donnons le lemme Topologique (lemme 28), à partir duquel on peut prouver des versions étendues des lemmes 23, 24 et 25.

Lemme 26. *Pour tout $\varepsilon > 0$, si les vecteurs $[\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3]$ sont ε-G-réduits, alors les inégalités suivantes sont valides :*

$$\forall i < j, |\langle \mathbf{b}_i, \mathbf{b}_j \rangle| \leqslant \tfrac{1+\varepsilon}{2}\|\mathbf{b}_i\|^2,$$
$$\forall s_1, s_2 \in \{-1, 1\}, |\langle \mathbf{b}_3, s_1\mathbf{b}_1 + s_2\mathbf{b}_2 \rangle| \leqslant \tfrac{1+\varepsilon}{2}\|s_1\mathbf{b}_1 + s_2\mathbf{b}_2\|^2.$$

Démonstration. Puisque les vecteurs $[\mathbf{b}_1, \mathbf{b}_2]$ sont ε-G-réduits, nous avons $\mathbf{b}_2' \in (1+\varepsilon)\mathrm{Vor}(\mathbf{b}_1)$, où $\mathbf{b}_2'$ est la projection orthogonale de $\mathbf{b}_2$ sur l'espace engendré par le vecteur $\mathbf{b}_1$. Par conséquent, nous pouvons écrire $\mathbf{b}_2' = (1+\varepsilon)\mathbf{u}$ avec $\mathbf{u} \in \mathrm{Vor}(\mathbf{b}_1)$. En développant les inégalités $\|\mathbf{u} \pm \mathbf{b}_1\|^2 \geqslant \|\mathbf{u}\|^2$, nous obtenons que $|\langle \mathbf{u}, \mathbf{b}_1 \rangle| \leqslant \|\mathbf{b}_1\|^2/2$. Ainsi, on a $|\langle \mathbf{b}_3, \mathbf{b}_1 \rangle| \leqslant \frac{1+\varepsilon}{2}\|\mathbf{b}_1\|^2$.

Par ailleurs, les vecteurs $[\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3]$ sont ε-G-réduits, de telle sorte que $\mathbf{b}_3' \in (1+\varepsilon)\mathrm{Vor}(\mathbf{b}_1, \mathbf{b}_2)$, où $\mathbf{b}_3'$ est la projection orthogonale du vecteur $\mathbf{b}_3$ sur l'espace engendré par $[\mathbf{b}_1, \mathbf{b}_2]$. Par conséquent on peut écrire $\mathbf{b}_3' = (1+\varepsilon)\mathbf{u}$, avec $\mathbf{u} \in \mathrm{Vor}(\mathbf{b}_1, \mathbf{b}_2)$. Nous continuons comme ci-dessus en développant les inégalités $\|\mathbf{u} + s_1\mathbf{b}_1 + s_2\mathbf{b}_2\|^2 \geqslant \|\mathbf{u}\|^2$ pour tout $s_1, s_2 \in \{-1, 0, 1\}$, et cela donne le résultat. □

Le lemme précédent implique que si les vecteurs $[\mathbf{b}_1, \ldots, \mathbf{b}_d]$ sont ε-G-réduits, le seul cas pour lequel les $\mathbf{b}_i$ peuvent ne pas être linéairement indépendants arrive quand certains

d'entre eux sont nuls, ce qui ne peut malheureusement pas être évité quand on rend compact l'ensemble des bases réduites au sens de Minkowski. Le lemme suivant généralise le lemme 22. Il montre que si les vecteurs sont ε-G-réduits, et si on est en dimension inférieure ou égale à 4, alors le procédé d'orthogonalisation de Gram-Schmidt ne peut pas faire décroître arbitrairement la longueur des vecteurs initiaux.

Lemme 27. *Il existe $C > 0$ tel que pour tout $1 \leqslant d \leqslant 4$ et tout $\varepsilon > 0$ suffisamment petit, si les vecteurs $[\mathbf{b}_1, \ldots, \mathbf{b}_d]$ sont ε-G-réduits, alors nous avons $\|\mathbf{b}_d^*\| \geqslant C\|\mathbf{b}_d\|$.*

Démonstration. Puisque les vecteurs $[\mathbf{b}_1, \ldots, \mathbf{b}_d]$ sont ε-G-réduits, nous savons que si le vecteur $\mathbf{b}'_d$ est la projection orthogonale de $\mathbf{b}_d$ sur l'espace engendré par $[\mathbf{b}_1, \ldots, \mathbf{b}_{d-1}]$, alors on a $\mathbf{b}'_d \in (1+\varepsilon)\mathrm{Vor}(\mathbf{b}_1, \ldots, \mathbf{b}_{d-1})$. Ainsi, grâce au lemme 22 et puisque les vecteurs sont ordonnés,

$$\|\mathbf{b}'_d\| \leqslant (1+\varepsilon)\|\mathbf{b}_{d-1}\|\frac{\sqrt{d-1}}{2} \leqslant (1+\varepsilon)\|\mathbf{b}_d\|\frac{\sqrt{d-1}}{2}.$$

Par ailleurs, le théorème de Pythagore donne que $\|\mathbf{b}_d\|^2 = \|\mathbf{b}_d^*\|^2 + \|\mathbf{b}'_d\|^2$, ce qui finit la preuve du lemme. □

Le lemme Topologique est l'argument-clé pour étendre les résultats sur les coordonnées de Voronoï pour les bases réduites au sens de Minkowski vers des bases ε-G-réduites. Quand nous l'utiliserons, l'ensemble X_0 correspondra aux x_i, l'ensemble K_0 aux vecteurs $\mathbf{b}_i$ qui sont ε-G-réduits, l'ensemble X aux coordonnées de Voronoï possibles, l'ensemble K à un sous-ensemble compact des bases Minkowski-réduites, et f à la fonction continue à variables réelles $f : (y_i)_{i\leqslant d}, (\mathbf{b}_i)_{i\leqslant d} \longrightarrow \|y_1\mathbf{b}_1 + \ldots + y_d\mathbf{b}_d\|$.

Lemme 28 (Lemme topologique). *Soient $n, m \geqslant 1$. Soient X_0 et K_0 des ensembles compacts de $\mathbb{R}^n$ et $\mathbb{R}^m$. Soit f une fonction continue de $K_0 \times X_0$ vers $\mathbb{R}$. Pour tout $a \in K_0$ nous définissons $M_a = \{x \in X_0 \cap \mathbb{Z}^n, f(a,x) = \min_{x' \in X_0 \cap \mathbb{Z}^n} f(a,x')\}$. Soient $K \subset K_0$ un compact et $X = \cup_{a\in K} M_a \subset X_0 \cap \mathbb{Z}^n$. Avec ces notations, il existe $\varepsilon > 0$ tel que si $b \in K_0$ satisfait $\mathrm{Dist}(b, K) \leqslant \varepsilon$, nous avons $M_b \subset X$.*

Démonstration. Tout d'abord, toutes les notations ont un sens : $X_0 \cap \mathbb{Z}^n$ est fini, et donc le minimum de $f(a,\cdot)$ sur cet ensemble existe, et M_a est fini. Comme $X \subset X_0 \cap \mathbb{Z}^n$, l'ensemble X est fini. Pour finir, puisque K est compact, la notation $\mathrm{Dist}(\cdot, K)$ est bien définie aussi.

Pour chaque $x \in X_0$, nous définissons $K_x = \{a \in K_0, x \in M_a\}$. L'ensemble K_x est compact. En effet, il est clairement borné, et, si (a_k) est une suite d'éléments de K_x qui converge vers un $a \in K_0$, nous montrons que $a \in K_x$. Pour tout $x' \in X_0 \cap \mathbb{Z}^n$ et pour tout k, nous avons $f(a_k, x) \leqslant f(a_k, x')$. Par continuité, ceci est aussi valide pour la limite a, ce qui montre que $x \in M_a$.

Maintenant, nous choisissons un $x \in X_0 \cap \mathbb{Z}^n \setminus X$. Puisque K_x et K sont tous les deux compacts et $x \notin X$ (ce qui implique que $K \cap K_x = \emptyset$), nous avons $\mathrm{Dist}(K_x, K) > 0$. Puisque l'ensemble $(X_0 \cap \mathbb{Z}^n) \setminus X$ est fini, pour finir la preuve il suffit de choisir $\varepsilon = \frac{1}{2}\min[\mathrm{Dist}(K_x, K), x \in (X_0 \cap \mathbb{Z}^n) \setminus X]$. □

Pour utiliser le lemme Topologique, nous avons besoin de considérer les bases ε-G-réduites comme un ensemble compact. Pour tout $\varepsilon \geqslant 0$ et tout $\alpha \in [0,1]$, nous définissons :

$$\begin{aligned} K_2(\varepsilon,\alpha) &= \{(\mathbf{b}_1,\mathbf{b}_2), [\mathbf{b}_1,\mathbf{b}_2] \ \varepsilon\text{-G-réduits}, \ \alpha \leqslant \|\mathbf{b}_1\| \leqslant \|\mathbf{b}_2\| = 1\} \\ K_3(\varepsilon,\alpha) &= \{(\mathbf{b}_1,\mathbf{b}_2,\mathbf{b}_3), [\mathbf{b}_1,\mathbf{b}_2,\mathbf{b}_3] \ \varepsilon\text{-G-réduits}, \ \alpha \leqslant \|\mathbf{b}_1\| \leqslant \|\mathbf{b}_2\| \leqslant \|\mathbf{b}_3\| = 1\}. \end{aligned}$$

Lemme 29. *Si $\varepsilon \geqslant 0$ et $\alpha \in [0,1]$, les ensembles $K_2(\varepsilon,\alpha)$ et $K_3(\varepsilon,\alpha)$ sont des compacts.*

Le lemme suivant est une version étendue du lemme 23. Il peut aussi être vu comme une réciproque au lemme 26.

Lemme 30. *Il existe $\varepsilon > 0$ tel que pour tout $\alpha \in]0,1]$, les coordonnées de Voronoï possibles de $[\mathbf{b}_1,\mathbf{b}_2] \in K_2(\varepsilon,\alpha)$ sont les mêmes que pour les bases Minkowski-réduites, c'est-à-dire $(1,0)$ et $(1,1)$, à tout changement de signes et toute permutation de coordonnées près.*

Démonstration. Rappelons qu'il y a un ensemble de coordonnées de Voronoï possibles pour chaque élément non nul de $(\mathbb{Z}/2\mathbb{Z})^2$: nous nous intéressons aux minima des éléments de $L/2L$. Comme il y a un nombre fini de tels éléments (trois en dimension 2), nous les traitons séparément. Soit $(a_1,a_2) \in \{0,1\}^2$. Nous cherchons les couples $(k_1,k_2) \in \mathbb{Z}^2$ qui minimisent $\|(a_1+2k_1)\mathbf{b}_1 + (a_2+2k_2)\mathbf{b}_2\|$, où les vecteurs $[\mathbf{b}_1,\mathbf{b}_2]$ sont ε-G-réduits. Nous montrons d'abord que le minimum pris par rapport aux k_i peut en fait être pris sur un ensemble fini. Remarquons que :

$$2 \geqslant \|\mathbf{b}_1\| + \|\mathbf{b}_2\| \geqslant \|a_1\mathbf{b}_1 + a_2\mathbf{b}_2\| \geqslant \|(a_1+2k_1)\mathbf{b}_1 + (a_2+2k_2)\mathbf{b}_2\|.$$

De plus, le lemme 27 donne que :

$$\|(a_1+2k_1)\mathbf{b}_1 + (a_2+2k_2)\mathbf{b}_2\| \geqslant |a_2+2k_2|\|\mathbf{b}_2^*\| \geqslant |a_2+2k_2|C,$$

ce qui donne le résultat pour k_2. En appliquant l'inégalité triangulaire, nous obtenons le résultat pour k_1 :

$$|a_1+2k_1|\alpha \leqslant \|(a_1+2k_1)\mathbf{b}_1\| \leqslant 2 + \|(a_2+2k_2)\mathbf{b}_2\| \leqslant 2 + |a_2+2k_2|.$$

À partir de cette équation nous déduisons que le couple $(k_1,k_2) \in \mathbb{Z}^2$ peut être borné indépendamment de $(\mathbf{b}_1,\mathbf{b}_2)$. Nous savons que $K_2(\varepsilon,\alpha)$ est compact, donc le lemme Topologique peut être utilisé. Ceci donne le résultat escompté. □

Nous étendons maintenant le lemme 24 de la même manière. Pour faire cela, nous imitons la preuve ci-dessus.

Lemme 31. *Il existe $\varepsilon > 0$ tel que pour tout $\alpha \in]0,1]$, les coordonnées de Voronoï possibles de $[\mathbf{b}_1,\mathbf{b}_2,\mathbf{b}_3] \in K_3(\varepsilon,\alpha)$ sont les mêmes que pour les bases réduites au sens de Minkowski.*

Démonstration. Nous considérons chaque élément de $L/2L$ séparément (il y en a 7 en dimension 3). Soit $(a_1, a_2, a_3) \in \{0,1\}^3$. Nous cherchons les triplets $(k_1, k_2, k_3) \in \mathbb{Z}^3$ qui minimisent $\|(a_1 + 2k_1)\mathbf{b}_1 + (a_2 + 2k_2)\mathbf{b}_2 + (a_3 + 2k_3)\mathbf{b}_3\|$, où les vecteurs $[\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3]$ sont ε-G-réduits. Pour appliquer le lemme Topologique, à partir duquel le résultat découle facilement, il suffit de montrer que le minimum peut être pris relativement à un ensemble fini de k_i. Remarquons que :

$$3 \geqslant \|a_1\mathbf{b}_1 + a_2\mathbf{b}_2 + a_3\mathbf{b}_3\| \geqslant \|(a_1 + 2k_1)\mathbf{b}_1 + (a_2 + 2k_2)\mathbf{b}_2 + (a_3 + 2k_3)\mathbf{b}_3\|.$$

De plus, le lemme 27 donne que :

$$\|(a_1 + 2k_1)\mathbf{b}_1 + (a_2 + 2k_2)\mathbf{b}_2 + (a_3 + 2k_3)\mathbf{b}_3\| \geqslant |a_3 + 2k_3|\|\mathbf{b}_3^*\| \geqslant |a_3 + 2k_3|C,$$

pour tout $\varepsilon > 0$ suffisamment petit. Ceci donne le résultat pour k_3.

Avec l'inégalité triangulaire, le lemme 27, et le fait que $\|\mathbf{b}_2\| \geqslant \alpha$, nous avons :

$$3 + |a_3 + 2k_3| \geqslant \|(a_1 + 2k_1)\mathbf{b}_1 + (a_2 + 2k_2)\mathbf{b}_2\| \geqslant |a_2 + 2k_2|\|\mathbf{b}_2^*\| \geqslant |a_2 + 2k_2|C\alpha.$$

Le fait que k_3 est pris sur un domaine fini implique que k_2 est borné.

Pour obtenir le résultat sur k_1, il suffit d'utiliser l'inégalité triangulaire une fois de plus :

$$3 + |a_3 + 2k_3| + |a_2 + 2k_2| \geqslant \|(a_1 + 2k_1)\mathbf{b}_1\| \geqslant |a_1 + 2k_1|\alpha.$$

□

Le lemme suivant généralise le lemme 25 pour la coordonnée possible $(1, 1, 2)$. Contrairement aux deux résultats précédents, il n'y a pas besoin d'utiliser le lemme Topologique puisqu'un nombre fini (en fait un seul) de triplets (x_1, x_2, x_3) est considéré.

Lemme 32. *Il existe $c > 0$ tel que pour tout $\varepsilon > 0$ suffisamment petit, si les vecteurs $[\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3]$ sont ε-G-réduits et $\|\mathbf{b}_3\| = 1$, alors :*

1. *Si $(s_1, s_2, 2)$ est une coordonnée de Voronoï avec $s_i = \pm 1$ pour $i \in \{1, 2\}$, alors : $\|\mathbf{b}_1\| \geqslant 1 - c\varepsilon$, $|\langle \mathbf{b}_1, \mathbf{b}_2\rangle| \leqslant c\varepsilon$ et $\left|\langle \mathbf{b}_i, \mathbf{b}_3\rangle + \frac{s_i}{2}\|\mathbf{b}_1\|^2\right| \leqslant c\varepsilon$ pour $i \in \{1, 2\}$.*
2. *Si $(s_1, 2, s_3)$ est une coordonnée de Voronoï avec $s_i = \pm 1$ pour $i \in \{1, 3\}$, alors : $(1 - c\varepsilon)\|\mathbf{b}_2\| \leqslant \|\mathbf{b}_1\| \leqslant \|\mathbf{b}_2\|$. De plus, si $\|\mathbf{b}_1\| \geqslant 1 - \varepsilon$, alors $|\langle \mathbf{b}_1, \mathbf{b}_3\rangle| \leqslant c\varepsilon$ et $\left|\langle \mathbf{b}_i, \mathbf{b}_2\rangle + \frac{s_i}{2}\|\mathbf{b}_1\|^2\right| \leqslant c\varepsilon$ pour $i \in \{1, 3\}$.*
3. *Si $(2, s_2, s_3)$ est une coordonnée de Voronoï avec $s_i = \pm 1$ pour $i \in \{2, 3\}$, et si $\|\mathbf{b}_1\| \geqslant 1 - \varepsilon$, alors : $|\langle \mathbf{b}_2, \mathbf{b}_3\rangle| \leqslant c\varepsilon$ et $\left|\langle \mathbf{b}_i, \mathbf{b}_1\rangle + \frac{s_i}{2}\|\mathbf{b}_1\|^2\right| \leqslant c\varepsilon$ pour $i \in \{2, 3\}$.*

La preuve de ce résultat est une généralisation directe de la preuve du lemme 25.

Démonstration. Sans perte de généralité, nous supposons que pour tout i, nous avons $s_i = 1$. Les preuves des différents cas étant similaires, nous ne nous intéressons qu'à la première situation.

Nous considérons l'inégalité $\|\mathbf{b}_1 + \mathbf{b}_2 + 2\mathbf{b}_3\|^2 \leqslant \|\mathbf{b}_1 + \mathbf{b}_2\|^2$: elle est équivalente à $\|\mathbf{b}_3\|^2 + \langle \mathbf{b}_1, \mathbf{b}_3\rangle + \langle \mathbf{b}_2, \mathbf{b}_3\rangle \leqslant 0$. Puisque les vecteurs $[\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3]$ sont ε-G-réduits, en utilisant le lemme 26, nous obtenons : $\langle \mathbf{b}_1, \mathbf{b}_3\rangle \geqslant -\frac{1+\varepsilon}{2}\|\mathbf{b}_1\|^2$ et $\langle \mathbf{b}_2, \mathbf{b}_3\rangle \geqslant -\frac{1+\varepsilon}{2}\|\mathbf{b}_2\|^2$.

Ainsi, $2\|\mathbf{b}_3\|^2 - (1+\varepsilon)\|\mathbf{b}_1\|^2 - (1+\varepsilon)\|\mathbf{b}_2\|^2 \leqslant 0$. Par conséquent, la « quasi-égalité » des longueurs est valide, et les inégalités ci-dessus sur les produits scalaires sont des « quasi-égalités », comme annoncé dans le théorème.

Il reste à montrer que $|\langle \mathbf{b}_1, \mathbf{b}_2\rangle|$ est très petit. En développant la relation $\|\mathbf{b}_1 + \mathbf{b}_2 + 2\mathbf{b}_3\|^2 \leqslant \|\mathbf{b}_1 - \mathbf{b}_2\|^2$, on obtient que $\langle \mathbf{b}_1, \mathbf{b}_2\rangle \leqslant c_1\varepsilon$ pour une certaine constante $c_1 > 0$. De manière similaire, la relation $|\langle \mathbf{b}_3, \mathbf{b}_1 + \mathbf{b}_2\rangle| \leqslant \frac{1+\varepsilon}{2}\|\mathbf{b}_1 + \mathbf{b}_2\|^2$ (donnée par le lemme 26) donne $\langle \mathbf{b}_1, \mathbf{b}_2\rangle \geqslant -c_2\varepsilon$ pour une certaine constante $c_2 > 0$. □

2.8.3 Le lemme du vide.

La finalité de ce paragraphe est de montrer qu'avec des bases ε-G-réduites, si l'on additionne un vecteur du réseau, avec des coordonnées qui ne sont pas trop petites, à un vecteur de la cellule de Voronoï, alors ce dernier vecteur devient plus long de manière significative. Ce résultat a été utilisé à la section 2.3 dans l'autre sens : si les x_i trouvés à l'étape 5 de l'algorithme glouton de réduction récursif ne sont pas trop petits, alors le vecteur $\mathbf{b}_d$ est significativement plus court que le vecteur $\mathbf{a}_d$. Nous commençons par généraliser les ensembles compacts K_2 et K_3. Pour tout $\varepsilon \geqslant 0$ et tout $\alpha \in [0,1]$, nous définissons :

$$\begin{aligned} K_2'(\varepsilon,\alpha) &= \{(\mathbf{b}_1,\mathbf{b}_2,\mathbf{u}), (\mathbf{b}_1,\mathbf{b}_2) \in K_2(\varepsilon,\alpha) \text{ et } \mathbf{u} \in \mathrm{Vor}(\mathbf{b}_1,\mathbf{b}_2)\} \\ K_3'(\varepsilon,\alpha) &= \{(\mathbf{b}_1,\mathbf{b}_2,\mathbf{b}_3,\mathbf{u}), (\mathbf{b}_1,\mathbf{b}_2,\mathbf{b}_3) \in K_3(\varepsilon,\alpha) \text{ et } \mathbf{u} \in \mathrm{Vor}(\mathbf{b}_1,\mathbf{b}_2,\mathbf{b}_3)\}. \end{aligned}$$

Lemme 33. *Si $\varepsilon \geqslant 0$ et $\alpha \in [0,1]$, les ensembles $K_2'(\varepsilon,\alpha)$ et $K_3'(\varepsilon,\alpha)$ sont compacts.*

Démonstration. Comme les preuves en dimensions 2 et 3 sont les mêmes, nous montrons le résultat uniquement pour $K_2'(\varepsilon,\alpha)$. Il suffit de montrer que l'ensemble $K_2'(\varepsilon,\alpha)$ est fermé et borné. Le fait qu'il est borné est évident puisque $\|\mathbf{b}_1\| \leqslant \|\mathbf{b}_2\| = 1$ et $\|\mathbf{u}\| \leqslant \|\mathbf{b}_1\| + \|\mathbf{b}_2\| \leqslant 2$. Nous supposons désormais que $K_2'(\varepsilon,\alpha)$ ne soit pas fermé, et nous cherchons à faire apparaître une contradiction. Soit $(\mathbf{b}_1^{(n)}, \mathbf{b}_2^{(n)}, \mathbf{u}^{(n)})$ une suite d'éléments de $K_2'(\varepsilon,\alpha)$ qui converge vers $(\mathbf{b}_1,\mathbf{b}_2,\mathbf{u}) \notin K_2'(\varepsilon,\alpha)$. Par définition de $K_2'(\varepsilon,\alpha)$, il existe un couple $(x_1,x_2) \neq (0,0)$ tel que $\|x_1\mathbf{b}_1 + x_2\mathbf{b}_2 + \mathbf{u}\| > \|\mathbf{u}\|$. Il existe donc un entier $n \geqslant 0$ tel que $\|x_1\mathbf{b}_1^{(n)} + x_2\mathbf{b}_2^{(n)} + \mathbf{u}^{(n)}\| > \|\mathbf{u}^{(n)}\|$. Dans ce cas, $\mathbf{u}^{(n)} \notin \mathrm{Vor}(\mathbf{b}_1^{(n)}, \mathbf{b}_2^{(n)})$, ce qui est impossible. □

Le résultat suivant est le lemme du vide en dimension 2.

Lemme 34. *Il existe deux constantes $\varepsilon, C > 0$ telles que pour tous vecteurs ε-G-réduits $[\mathbf{b}_1,\mathbf{b}_2]$ et tout vecteur $\mathbf{u} \in \mathrm{Vor}(\mathbf{b}_1,\mathbf{b}_2)$, si l'une des deux conditions suivantes est vérifiée, alors : $\|\mathbf{u} + x_1\mathbf{b}_1 + x_2\mathbf{b}_2\|^2 \geqslant \|\mathbf{u}\|^2 + C\|\mathbf{b}_2\|^2$.*

1. *$|x_2| \geq 2$,*
2. *$|x_1| \geq 2$ et $\|\mathbf{b}_1\| \geqslant (1-\varepsilon)\|\mathbf{b}_2\|$.*

Démonstration. La preuve fait intervenir trois faits. Le premier aide à rendre compacte l'ensemble des variables $[\mathbf{b}_1,\mathbf{b}_2]$. Le deuxième montre que l'on peut supposer que $\mathbf{b}_1$

et $\mathbf{b}_2$ ont des longueurs similaires, et le dernier est l'étape-clé pour pouvoir appliquer le lemme 30, qui permet de finir la preuve.

Fait 1 : Sans perte de généralité, on peut supposer que $\|\mathbf{b}_2\| = 1$.

Soit $(*)$ la proposition suivante :
« Il existe deux constantes $\varepsilon, C > 0$ telles que pour tous vecteurs ε-G-réduits $[\mathbf{b}_1, \mathbf{b}_2]$ avec $\|\mathbf{b}_2\| = 1$, et pour tout vecteur $\mathbf{u} \in \mathrm{Vor}(\mathbf{b}_1, \mathbf{b}_2)$, si l'une des deux conditions suivantes est vérifiée, alors : $\|\mathbf{u} + x_1\mathbf{b}_1 + x_2\mathbf{b}_2\|^2 \geq \|\mathbf{u}\|^2 + C$.
– A- $|x_2| \geq 2$,
– B- $|x_1| \geq 2$ et $\|\mathbf{b}_1\| \geqslant 1 - \varepsilon$. »
Nous gardons les mêmes constantes $\varepsilon, C > 0$. Soient $[\mathbf{b}_1, \mathbf{b}_2]$ des vecteurs ε-G-réduits. Si $\|\mathbf{b}_2\| = 0$, le résultat est évident. Sinon, soit $\mathbf{b}'_i = \frac{1}{\|\mathbf{b}_2\|}\mathbf{b}_i$ pour $i \in \{1, 2\}$. Nous appliquons la proposition $(*)$ aux vecteurs ε-G-réduits $[\mathbf{b}'_1, \mathbf{b}'_2]$. Soient $\mathbf{u} \in \mathrm{Vor}(\mathbf{b}_1, \mathbf{b}_2)$ et $\mathbf{u}' = \frac{1}{\|\mathbf{b}_2\|}\mathbf{u}$. Alors $\mathbf{u}' \in \mathrm{Vor}(\mathbf{b}'_1, \mathbf{b}'_2)$. Si la condition (1) ou la condition (2) est satisfaite, alors $\|\mathbf{u}' + x_1\mathbf{b}'_1 + x_2\mathbf{b}'_2\|^2 \geq \|\mathbf{u}'\|^2 + C$. En multipliant les deux côtés par $\|\mathbf{b}_2\|^2$, on obtient le résultat. ■

Nous distinguons désormais deux cas : soit $\mathbf{b}_1$ est à peu près de la longueur de $\mathbf{b}_2$, soit il est nettement plus court (ce qui ne peut pas arriver si la condition (2) est satisfaite). L'idée du prochain fait est que lorsque $\mathbf{b}_1$ tend vers $\mathbf{0}$, $(\mathbf{b}_1, \mathbf{b}_2, \mathbf{u})$ tend vers $(\mathbf{0}, \mathbf{b}_2, \mathbf{u}')$ où le vecteur $\mathbf{u}'$ est proche de $\mathrm{Vor}(\mathbf{b}_2)$.

Fait 2 : Il existe $C, \alpha, \varepsilon > 0$ tels que pour tout $(\mathbf{b}_1, \mathbf{b}_2, \mathbf{u}) \in K'_2(\varepsilon, 0)$ avec $\|\mathbf{b}_1\| \leqslant \alpha$, dès que $|x_2| \geqslant 2$ on a :

$$\|\mathbf{u} + x_1\mathbf{b}_1 + x_2\mathbf{b}_2\|^2 - \|\mathbf{u}\|^2 \geqslant C.$$

Comme $\mathbf{u} \in \mathrm{Vor}(\mathbf{b}_1, \mathbf{b}_2)$, nous avons $|\langle \mathbf{u}, \mathbf{b}_i \rangle| \leqslant \|\mathbf{b}_i\|^2/2$ pour $i \in \{1, 2\}$. Par conséquent :

$$\begin{aligned}
\|\mathbf{u} + x_1\mathbf{b}_1 + x_2\mathbf{b}_2\|^2 - \|\mathbf{u}\|^2 &= \|x_1\mathbf{b}_1 + x_2\mathbf{b}_2\|^2 + 2\langle \mathbf{u}, x_1\mathbf{b}_1 + x_2\mathbf{b}_2 \rangle \\
&\geqslant \|x_1\mathbf{b}_1 + x_2\mathbf{b}_2\|^2 - |x_1|\|\mathbf{b}_1\|^2 - |x_2| \\
&\geqslant (x_1^2 - |x_1|)\|\mathbf{b}_1\|^2 + (x_2^2 - |x_2|) - 2|x_1x_2||\langle \mathbf{b}_1, \mathbf{b}_2 \rangle|.
\end{aligned}$$

Puisque les vecteurs $[\mathbf{b}_1, \mathbf{b}_2]$ sont ε-G-réduits, en utilisant le lemme 26, nous avons l'inégalité $|\langle \mathbf{b}_1, \mathbf{b}_2 \rangle| \leqslant \frac{1+\varepsilon}{2}\|\mathbf{b}_1\|^2$, d'où l'on obtient :

$$\|\mathbf{u} + x_1\mathbf{b}_1 + x_2\mathbf{b}_2\|^2 - \|\mathbf{u}\|^2 \geqslant |x_1|[|x_1| - (1+\varepsilon)|x_2| - 1]\|\mathbf{b}_1\|^2 + (x_2^2 - |x_2|).$$

Nous cherchons maintenant à minimiser cette dernière expression vis-à-vis de la variable $|x_1|$ (c'est un polynôme de degré 2), et avec le choix optimal « $|x_1| = \frac{(1+\varepsilon)|x_2|+1}{2}$ », on obtient :

$$\begin{aligned}
\|\mathbf{u} + x_1\mathbf{b}_1 + x_2\mathbf{b}_2\|^2 - \|\mathbf{u}\|^2 &\geqslant -\frac{[(1+\varepsilon)|x_2| + 1]^2}{4}\|\mathbf{b}_1\|^2 + (x_2^2 - |x_2|) \\
&\geqslant \left(1 - \frac{\alpha^2(1+\varepsilon)^2}{4}\right)|x_2|^2 - \left(1 + \frac{\alpha^2(1+\varepsilon)}{2}\right)|x_2| - \frac{\alpha^2}{4}
\end{aligned}$$

Pour une constante α suffisamment petite, le minimum de ce polynôme de degré 2 en la variable $|x_2|$ est atteint pour « $|x_2| \leqslant 2$ », et le polynôme est décroissant par rapport à $|x_2| \geqslant 2$. Par hypothèse $|x_2| \geqslant 2$, et donc on a :

$$\|\mathbf{u} + x_1\mathbf{b}_1 + x_2\mathbf{b}_2\|^2 - \|\mathbf{u}\|^2 \geqslant 2 - \alpha^2(9/4 + 3\varepsilon + \varepsilon^2),$$

ce qui donne le résultat. ■

La constante $\alpha > 0$ est choisie conformément au fait 2, et nous choisissons $\varepsilon > 0$ conformément au fait 2 et aux lemmes 27 et 30. Nous considérons maintenant une suite $(\mathbf{b}_1^{(k)}, \mathbf{b}_2^{(k)}, \mathbf{u}^{(k)}, x_1^{(k)}, x_2^{(k)})_k$ telle que :

1. $(\mathbf{b}_1^{(k)}, \mathbf{b}_2^{(k)}, \mathbf{u}^{(k)}) \in K_2'(\varepsilon, \alpha)$,
2. $x_1^{(k)}, x_2^{(k)}$ sont des entiers tels que $|x_2^{(k)}| \geqslant 2$ (respectivement $|x_1^{(k)}| \geqslant 2$),
3. $\|\mathbf{u}^{(k)} + x_1^{(k)}\mathbf{b}_1^{(k)} + x_2^{(k)}\mathbf{b}_2^{(k)}\|^2 - \|\mathbf{u}^{(k)}\|^2 \to 0$ quand $k \to \infty$.

S'il n'existe pas de telle suite alors (1) (respectivement (2)) est prouvée. Pour finir la preuve du lemme 34, supposons qu'une telle suite existe et faisons apparaître une contradiction vis-à-vis des coordonnées de Voronoï.

Fait 3 : Pour tout $\varepsilon > 0$ suffisamment petit et pour tout $\alpha > 0$, les suites $x_2^{(k)}$ et $x_1^{(k)}$ restent bornées.

Supposons que ce ne soit pas le cas. Nous allons montrer que cela implique que $\|\mathbf{u}^{(k)} + x_1^{(k)}\mathbf{b}_1^{(k)} + x_2^{(k)}\mathbf{b}_2^{(k)}\|^2 - \|\mathbf{u}^{(k)}\|^2$ n'est pas borné, ce qui est impossible. L'inégalité de Cauchy-Schwarz donne :

$$\|\mathbf{u}^{(k)} + x_1^{(k)}\mathbf{b}_1^{(k)} + x_2^{(k)}\mathbf{b}_2^{(k)}\|^2 - \|\mathbf{u}^{(k)}\|^2 \geqslant \|x_1^{(k)}\mathbf{b}_1^{(k)} + x_2^{(k)}\mathbf{b}_2^{(k)}\|[\|x_1^{(k)}\mathbf{b}_1^{(k)} + x_2^{(k)}\mathbf{b}_2^{(k)}\| - 2\|\mathbf{u}^{(k)}\|].$$

Puisque la quantité $\|\mathbf{u}^{(k)}\|$ est bornée, il suffit de montrer que $\|x_1^{(k)}\mathbf{b}_1^{(k)} + x_2^{(k)}\mathbf{b}_2^{(k)}\|$ n'est pas borné. Le lemme 27 nous donne que :

$$\|x_1^{(k)}\mathbf{b}_1^{(k)} + x_2^{(k)}\mathbf{b}_2^{(k)}\| \geqslant |x_2^{(k)}|\|\mathbf{b}_2^{(k)*}\| \geqslant C|x_2^{(k)}|.$$

Ainsi la suite $x_2^{(k)}$ est bornée. L'inégalité triangulaire et le fait que $\|\mathbf{b}_1^{(k)}\| \geqslant \alpha$ garantissent que la suite $x_1^{(k)}$ reste bornée elle aussi. ■

Le fait précédent et le lemme 33 impliquent que $(\mathbf{b}_1^{(k)}, \mathbf{b}_2^{(k)}, \mathbf{u}^{(k)}, x_1^{(k)}, x_2^{(k)})$ reste dans un sous-ensemble compact de $\mathbb{R}^{3n} \times \mathbb{Z}^2$. Nous pouvons donc en extraire une sous-suite qui converge vers une limite $(\mathbf{b}_1, \mathbf{b}_2, \mathbf{u}, x_1, x_2)$ qui est dans le même compact et qui satisfait : $\|\mathbf{u} + x_1\mathbf{b}_1 + x_2\mathbf{b}_2\| = \|\mathbf{u}\|$. Cela signifie que $x_1\mathbf{b}_1 + x_2\mathbf{b}_2$ est un vecteur de Voronoï du réseau engendré par $[\mathbf{b}_1, \mathbf{b}_2] \in K_2'(\varepsilon, \alpha)$, ce qui contredit le lemme 30. □

Nous donnons désormais le lemme du vide en dimension 3, sur lequel repose l'analyse de l'algorithme glouton de réduction en dimension 4.

Lemme 35. *Il existe deux constantes $\varepsilon, C > 0$ telles que pour tous vecteurs ε-G-réduits $[\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3]$ et tout vecteur $\mathbf{u} \in \mathrm{Vor}(\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3)$, si l'une des conditions suivantes est vérifiée, alors :*

$$\|\mathbf{u} + x_1\mathbf{b}_1 + x_2\mathbf{b}_2 + x_3\mathbf{b}_3\|^2 \geqslant \|\mathbf{u}\|^2 + C\|\mathbf{b}_3\|^2.$$

1. $|x_3| \geqslant 3$, *ou* $|x_3| = 2$ *avec* $(|x_1|, |x_2|) \neq (1,1)$.
2. $\|\mathbf{b}_2\| \geqslant (1-\varepsilon)\|\mathbf{b}_3\|$ *et :* $|x_2| \geqslant 3$, *ou* $|x_2| = 2$ *avec* $(|x_1|, |x_3|) \neq (1,1)$.
3. $\|\mathbf{b}_1\| \geqslant (1-\varepsilon)\|\mathbf{b}_3\|$ *et :* $|x_1| \geqslant 3$, *ou* $|x_1| = 2$ *avec* $(|x_2|, |x_3|) \neq (1,1)$.

Démonstration. Il est facile de voir que comme pour le lemme 34 on peut supposer que $\|\mathbf{b}_3\| = 1$. Nous considérons trois cas : les vecteurs $\mathbf{b}_1$ et $\mathbf{b}_2$ sont significativement plus courts que $\mathbf{b}_3$; le vecteur $\mathbf{b}_1$ est très court, mais les vecteurs $\mathbf{b}_2$ et $\mathbf{b}_3$ ont des longueurs similaires ; et, finalement, tous les vecteurs ont des longueurs similaires.

Fait 1 : Il existe trois constantes $C, \alpha, \varepsilon > 0$ telles que pour tout $(\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{u}) \in K_3'(\varepsilon, 0)$ avec $\|\mathbf{b}_2\| \leqslant \alpha$, dès que $|x_3| \geqslant 2$ on a : $\|\mathbf{u} + x_1\mathbf{b}_1 + x_2\mathbf{b}_2 + x_3\mathbf{b}_3\|^2 - \|\mathbf{u}\|^2 \geqslant C$.

En utilisant le lemme 26 et le fait que $\mathbf{u} \in \mathrm{Vor}(\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3)$, nous avons les inégalités suivantes :

$$\begin{aligned}\|\mathbf{u} + x_1\mathbf{b}_1 + x_2\mathbf{b}_2 + x_3\mathbf{b}_3\|^2 - \|\mathbf{u}\|^2 &= \|x_1\mathbf{b}_1 + x_2\mathbf{b}_2 + x_3\mathbf{b}_3\|^2 + 2\langle \mathbf{u}, x_1\mathbf{b}_1 + x_2\mathbf{b}_2 + x_3\mathbf{b}_3\rangle \\ &\geqslant [x_1^2 - (1+\varepsilon)|x_1|(|x_2| + |x_3|) - |x_1|]\|\mathbf{b}_1\|^2 \\ &\quad + [x_2^2 - (1+\varepsilon)|x_2||x_3| - |x_2|]\|\mathbf{b}_2\|^2 + x_3^2 - |x_3|.\end{aligned}$$

Nous minimisons ce polynôme de degré 2 en la variable $|x_1|$, et pour le choix optimal « $|x_1| = \frac{(1+\varepsilon)(|x_2|+|x_3|)+1}{2}$ », nous obtenons :

$$\begin{aligned}&\|\mathbf{u} + x_1\mathbf{b}_1 + x_2\mathbf{b}_2 + x_3\mathbf{b}_3\|^2 - \|\mathbf{u}\|^2 \\ &\quad\geqslant -\tfrac{[(1+\varepsilon)(|x_2|+|x_3|)+1]^2}{4}\|\mathbf{b}_1\|^2 + [x_2^2 - (1+\varepsilon)|x_2||x_3| - |x_2|]\|\mathbf{b}_2\|^2 + x_3^2 - |x_3| \\ &\quad\geqslant \left[\tfrac{3-2\varepsilon-\varepsilon^2}{4}x_2^2 - \left(\tfrac{3+4\varepsilon+\varepsilon^2}{2}|x_3| + \tfrac{3+\varepsilon}{2}\right)|x_2| - \tfrac{1}{4}((1+\varepsilon)|x_3|+1)^2\right]\|\mathbf{b}_2\|^2 + x_3^2 - |x_3|,\end{aligned}$$

parce que $\|\mathbf{b}_1\| \leqslant \|\mathbf{b}_2\|$. Le terme entre crochets est un polynôme de degré 2 en la variable $|x_2|$, avec un coefficient dominant strictement positif (pour un $\varepsilon > 0$ suffisamment petit). Ce polynôme est donc borné inférieurement sur $\mathbb{Z}$ et le minimum est atteint en $|x_2| = |x_3| + 1$ quand $\varepsilon = 0$. On obtient donc :

$$\|\mathbf{u} + x_1\mathbf{b}_1 + x_2\mathbf{b}_2 + x_3\mathbf{b}_3\|^2 - \|\mathbf{u}\|^2 \geqslant [a_2(\varepsilon)x_3^2 + a_1(\varepsilon)|x_3| + a_0(\varepsilon)]\|\mathbf{b}_2\|^2 + x_3^2 - |x_3|,$$

où $a_2(\varepsilon) \to -1$, $a_2(\varepsilon) \to -2$ et $a_0(\varepsilon) \to -1$ quand $\varepsilon \to 0$. Quand $\varepsilon > 0$ est suffisamment petit, le facteur devant $\|\mathbf{b}_2\|^2$ est négatif, et comme $\|\mathbf{b}_2\| \leqslant \alpha$, on obtient :

$$\|\mathbf{u} + x_1\mathbf{b}_1 + x_2\mathbf{b}_2 + x_3\mathbf{b}_3\|^2 - \|\mathbf{u}\|^2 \geqslant (1 + \alpha^2 a_2(\varepsilon))x_3^2 + (-1 + \alpha^2 a_1(\varepsilon))|x_3| + \alpha^2 a_0(\varepsilon).$$

Pour des constantes $\alpha, \varepsilon > 0$ suffisamment petites, ce polynôme de degré 2 en $|x_3|$ croît strictement sur $[2, \infty[$, de telle sorte que si l'on remplace $|x_3|$ par 2, on obtient :

$$\|\mathbf{u} + x_1\mathbf{b}_1 + x_2\mathbf{b}_2 + x_3\mathbf{b}_3\|^2 - \|\mathbf{u}\|^2 \geqslant 4[1 + \alpha^2 a_2(\varepsilon)] - 2[1 - \alpha^2 a_1(\varepsilon)] + \alpha^2 a_0(\varepsilon).$$

Quand $\alpha > 0$ est assez petit, cette quantité est plus grande qu'une constante $C > 0$.

Fait 2 : Il existe $C, c'', \varepsilon, \alpha > 0$ tels que pour tout $(\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{u}) \in K_3'(\varepsilon, 0)$ avec $\|\mathbf{b}_1\| \leqslant \alpha$ et $\|\mathbf{b}_2\| \geqslant c''\alpha$, dès que $|x_3| \geqslant 2$, on a : $\|\mathbf{u} + x_1\mathbf{b}_1 + x_2\mathbf{b}_2 + x_3\mathbf{b}_3\|^2 - \|\mathbf{u}\|^2 \geqslant C$.

La preuve ressemble à celle du fait 2 du lemme 34, mais est un peu plus technique. Nous avons les inégalités suivantes :

$$\begin{aligned}\|\mathbf{u} + x_1\mathbf{b}_1 + x_2\mathbf{b}_2 + x_3\mathbf{b}_3\|^2 - \|\mathbf{u}\|^2 &= \|x_1\mathbf{b}_1 + x_2\mathbf{b}_2 + x_3\mathbf{b}_3\|^2 + 2\langle \mathbf{u}, x_1\mathbf{b}_1 + x_2\mathbf{b}_2 + x_3\mathbf{b}_3\rangle \\ &\geqslant [x_1^2 - (1+\varepsilon)|x_1|(|x_2|+|x_3|) - |x_1|]\|\mathbf{b}_1\|^2 + \|x_2\mathbf{b}_2 + x_3\mathbf{b}_3\|^2 + 2\langle \mathbf{u}, x_2\mathbf{b}_2 + x_3\mathbf{b}_3\rangle \\ &\geqslant -\tfrac{\alpha^2}{4}[(1+\varepsilon)(|x_2|+|x_3|)+1]^2 + \|x_2\mathbf{b}_2 + x_3\mathbf{b}_3\|^2 + 2\langle \mathbf{u}, x_2\mathbf{b}_2 + x_3\mathbf{b}_3\rangle.\end{aligned}$$

Nous écrivons $\mathbf{u} = \mathbf{u}' + \mathbf{u}''$, où $\mathbf{u}'$ est dans l'espace engendré par $[\mathbf{b}_2, \mathbf{b}_3]$, et $\mathbf{u}''$ lui est orthogonal. Il est clair que $\mathbf{u}'$ est dans $\mathrm{Vor}(\mathbf{b}_2, \mathbf{b}_3)$. Grâce au lemme 22, on sait que $\|\mathbf{u}'\| \leqslant 1/\sqrt{2}$. Par ailleurs, dès que $|x_3| \geqslant 2$:

$$\begin{aligned}\|x_2\mathbf{b}_2 + x_3\mathbf{b}_3\|^2 &\geqslant [x_2^2 - (1+\varepsilon)|x_2\|x_3|]\|\mathbf{b}_2\|^2 + x_3^2 \\ &\geqslant -\frac{[(1+\varepsilon)|x_3|]^2}{4}\|\mathbf{b}_2\|^2 + x_3^2 \\ &\geqslant x_3^2\left[1 - \frac{(1+\varepsilon)^2}{4}\right] \\ &\geqslant 3 - 2\varepsilon - \varepsilon^2.\end{aligned}$$

Par conséquent, $\sqrt{2} \leqslant \sqrt{\frac{2}{3-2\varepsilon-\varepsilon^2}}\|x_2\mathbf{b}_2 + x_3\mathbf{b}_3\|$, ce qui donne, en utilisant l'inégalité de Cauchy-Schwarz :

$$\langle \mathbf{u}, x_2\mathbf{b}_2 + x_3\mathbf{b}_3\rangle \geqslant -\frac{\sqrt{2}}{2}\|x_2\mathbf{b}_2 + x_3\mathbf{b}_3\| \geqslant -\frac{1}{2}\sqrt{\frac{2}{3-2\varepsilon-\varepsilon^2}}\|x_2\mathbf{b}_2 + x_3\mathbf{b}_3\|^2.$$

À partir de cela on obtient :

$$\begin{aligned}&\|\mathbf{u} + x_1\mathbf{b}_1 + x_2\mathbf{b}_2 + x_3\mathbf{b}_3\|^2 - \|\mathbf{u}\|^2 \\ &\quad\geqslant -\tfrac{\alpha^2}{4}[(1+\varepsilon)(|x_2|+|x_3|)+1]^2 + \left(1 - \sqrt{\tfrac{2}{3-2\varepsilon-\varepsilon^2}}\right)\|x_2\mathbf{b}_2 + x_3\mathbf{b}_3\|^2 \\ &\quad\geqslant -\tfrac{\alpha^2}{4}[(1+\varepsilon)(|x_2|+|x_3|)+1]^2 + \left(1 - \sqrt{\tfrac{2}{3-2\varepsilon-\varepsilon^2}}\right)[(x_2^2 - (1+2\varepsilon)|x_2\|x_3|)\|\mathbf{b}_2\|^2 + x_3^2].\end{aligned}$$

Puisque $|x_2x_3| \leq (x_2^2+x_3^2)/2$, on obtient, pour un $\varepsilon > 0$ suffisamment petit, une borne inférieure de la forme :

$$\begin{aligned}&\|\mathbf{u} + x_1\mathbf{b}_1 + x_2\mathbf{b}_2 + x_3\mathbf{b}_3\|^2 - \|\mathbf{u}\|^2 \\ &\qquad\geqslant (\alpha^2 f_1(\varepsilon) + g_1(\varepsilon)\|\mathbf{b}_2\|^2)x_2^2 + (\alpha^2 f_2(\varepsilon) + g_2(\varepsilon)\|\mathbf{b}_2\|^2 + g_3(\varepsilon))x_3^2 + \alpha^2 f_3(\varepsilon),\end{aligned}$$

où, lorsque ε tend vers 0 : $f_i(\varepsilon) = O(1)$, $\lim_\varepsilon g_1(\varepsilon) > 0$, $0 \leq |\lim_\varepsilon g_2(\varepsilon)| < \lim_\varepsilon g_3(\varepsilon)$. On en déduit que pour un bon $c'' > 0$ (choisi pour rendre positif le coefficient de x_2^2), il existe $C' > 0$ tel que pour un $\alpha > 0$ suffisamment petit et pour un $\varepsilon > 0$ suffisamment petit, et pour tout $(\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{u}) \in K_3'(\varepsilon, 0)$ avec $\|\mathbf{b}_1\| \leqslant \alpha$ et $\|\mathbf{b}_2\| \geqslant c''\alpha$, dans la borne inférieure ci-dessus, le coefficient de x_2^2 est positif et le coefficient de x_3^2 est plus grand que C'. Cela achève la preuve du fait 2. ■

Le reste de la preuve est identique à la fin de la preuve du lemme 34 : on choisit un $\alpha > 0$ qui satisfait les conditions des deux faits précédents et on considère une suite $(\mathbf{b}_1^{(k)}, \mathbf{b}_2^{(k)}, \mathbf{b}_3^{(k)}, \mathbf{u}^{(k)}, x_1^{(k)}, x_2^{(k)}, x_3^{(k)})_k$ telle que :

1. $(\mathbf{b}_1^{(k)}, \mathbf{b}_2^{(k)}, \mathbf{b}_3^{(k)}, \mathbf{u}^{(k)}) \in K_3'(\varepsilon, \alpha)$,
2. $x_1^{(k)}, x_2^{(k)}, x_3^{(k)}$ sont des entiers tels que $|x_3^{(k)}| \geqslant 3$ ou $|x_3^{(k)}| = 2$ et $(|x_1^{(k)}|, |x_2^{(k)}|) \neq (1,1)$ (respectivement $|x_i^{(k)}| \geqslant 3$ ou etc. pour $i \in \{1,2\}$),
3. $\|\mathbf{u}^{(k)} + x_1^{(k)}\mathbf{b}_1^{(k)} + x_2^{(k)}\mathbf{b}_2^{(k)} + x_3^{(k)}\mathbf{b}_3^{(k)}\|^2 - \|\mathbf{u}^{(k)}\|^2 \to 0$ quand $k \to \infty$.

Si une telle suite n'existe pas alors le lemme est prouvé. Nous supposons donc qu'il y en a une et nous cherchons une contradiction vis-à-vis des coordonnées de Voronoï.

Fait 3 : Pour tous $\varepsilon, \alpha > 0$ suffisamment petits, les suites $x_3^{(k)}$, $x_2^{(k)}$ et $x_1^{(k)}$ sont bornées.

Supposons que ce ne soit pas le cas. Nous montrons que cela implique que $\|\mathbf{u}^{(k)} + x_1^{(k)}\mathbf{b}_1^{(k)} + x_2^{(k)}\mathbf{b}_2^{(k)} + x_3^{(k)}\mathbf{b}_3^{(k)}\|^2 - \|\mathbf{u}^{(k)}\|^2$ n'est pas borné, ce qui est impossible. L'inégalité de Cauchy-Schwarz nous donne que :

$$\|\mathbf{u}^{(k)} + x_1^{(k)}\mathbf{b}_1^{(k)} + x_2^{(k)}\mathbf{b}_2^{(k)} + x_3^{(k)}\mathbf{b}_3^{(k)}\|^2 - \|\mathbf{u}^{(k)}\|^2 \geqslant \|x_1^{(k)}\mathbf{b}_1^{(k)} + x_2^{(k)}\mathbf{b}_2^{(k)} + x_3^{(k)}\mathbf{b}_3^{(k)}\|[\|x_1^{(k)}\mathbf{b}_1^{(k)} + x_2^{(k)}\mathbf{b}_2^{(k)} + x_3^{(k)}\mathbf{b}_3^{(k)}\| - 2\|\mathbf{u}^{(k)}\|].$$

Puisque la quantité $\|\mathbf{u}^{(k)}\|$ est bornée, il suffit de montrer que $\|x_1^{(k)}\mathbf{b}_1^{(k)} + x_2^{(k)}\mathbf{b}_2^{(k)} + x_3^{(k)}\mathbf{b}_3^{(k)}\|$ n'est pas borné. Nous prenons un $\varepsilon > 0$ suffisamment petit pour pouvoir utiliser le lemme 27. Soit $C > 0$ la constante correspondante. Nous avons :

$$\|x_1^{(k)}\mathbf{b}_1^{(k)} + x_2^{(k)}\mathbf{b}_2^{(k)} + x_3^{(k)}\mathbf{b}_3^{(k)}\| \geqslant |x_3^{(k)}|\|\mathbf{b}_3^{(k)*}\| \geqslant C|x_3^{(k)}|.$$

Ainsi, la suite $x_3^{(k)}$ est bornée. Si la suite $x_2^{(k)}$ est bornée, en utilisant l'inégalité triangulaire on voit qu'il en est forcément de même de la suite $x_1^{(k)}$. Nous montrons maintenant que la suite $x_2^{(k)}$ est bornée. Nous avons l'inégalité suivante :

$$\|x_1^{(k)}\mathbf{b}_1^{(k)} + x_2^{(k)}\mathbf{b}_2^{(k)} + x_3^{(k)}\mathbf{b}_3^{(k)}\| \geqslant |x_2^{(k)}|\|\mathbf{b}_2^{(k)*}\| - |x_3^{(k)}|\|\mathbf{b}_3^{(k)}\|.$$

Puisque les vecteurs $[\mathbf{b}_1^{(k)}, \mathbf{b}_2^{(k)}]$ sont ε-G-réduits et $1 \geqslant \|\mathbf{b}_2^{(k)}\| \geqslant \|\mathbf{b}_1^{(k)}\| \geqslant \alpha$, nous sommes dans une situation similaire à celle du troisième fait du lemme 34, et ceci nous donne une constante strictement positive qui borne inférieurement $\|\mathbf{b}_2^{(k)*}\|$, et qui dépend éventuellement de ε et de α. Ceci achève la preuve du fait. ■

Ce fait et le lemme 33 impliquent que $(\mathbf{b}_1^{(k)}, \mathbf{b}_2^{(k)}, \mathbf{b}_3^{(k)}, \mathbf{u}^{(k)}, x_1^{(k)}, x_2^{(k)}, x_3^{(k)})$ reste dans un ensemble compact de $\mathbb{R}^{4n} \times \mathbb{Z}^3$. Par conséquent nous pouvons en extraire une sous-suite qui converge vers la limite $(\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{u}, x_1, x_2, x_3)$, qui est dans le même compact et satisfait : $\|\mathbf{u} + x_1\mathbf{b}_1 + x_2\mathbf{b}_2 + x_3\mathbf{b}_3\| = \|\mathbf{u}\|$. Cela signifie que $x_1\mathbf{b}_1 + x_2\mathbf{b}_2 + x_3\mathbf{b}_3$ est un vecteur de Voronoï du réseau engendré par $[\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3] \in K_3'(\varepsilon, \alpha)$. D'après le lemme 31, c'est impossible. □

Comme dans les paragraphes précédents, nous considérons maintenant le cas des coordonnées de Voronoï possibles $(\pm 1, \pm 1, \pm 2)$, auxquelles on peut appliquer une quelconque permutation des coordonnées.

Lemme 36. *Il existe deux constantes $\varepsilon, C > 0$ telles que pour tous vecteurs ε-G-réduits $[\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3]$ et tout vecteur $\mathbf{u} \in \mathrm{Vor}(\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3)$, si au moins l'une des conditions suivantes est satisfaite, alors on a :*

$$\|\mathbf{u} + x_1\mathbf{b}_1 + x_2\mathbf{b}_2 + x_3\mathbf{b}_3\|^2 \geqslant \|\mathbf{u}\|^2 + C\|\mathbf{b}_3\|^2.$$

1. $(x_1, x_2, x_3) = (s_1, s_2, 2)$, $|s_i| = 1$ *pour* $i \in \{1, 2\}$ *et au moins l'une des conditions suivantes est satisfaite :*
   $\|\mathbf{b}_1\| \leqslant (1-\varepsilon)\|\mathbf{b}_3\|$, *ou* $\|\mathbf{b}_2\| \leqslant (1-\varepsilon)\|\mathbf{b}_3\|$, *ou* $|\langle \mathbf{b}_1, \mathbf{b}_2 \rangle| \geqslant \varepsilon \|\mathbf{b}_3\|^2$, *ou* $|\langle \mathbf{b}_1, \mathbf{b}_3 \rangle + \frac{s_1}{2}\|\mathbf{b}_1\|^2| \geqslant \varepsilon \|\mathbf{b}_3\|^2$, *ou* $|\langle \mathbf{b}_2, \mathbf{b}_3 \rangle + \frac{s_2}{2}\|\mathbf{b}_1\|^2| \geqslant \varepsilon \|\mathbf{b}_3\|^2$.
2. $(x_1, x_2, x_3) = (s_1, 2, s_3)$, $|s_i| = 1$ *pour* $i \in \{1, 3\}$ *et* $\|\mathbf{b}_1\| \leqslant (1-\varepsilon)\|\mathbf{b}_2\|$.
3. $(x_1, x_2, x_3) = (s_1, 2, s_3)$, $|s_i| = 1$ *pour* $i \in \{1, 3\}$, $\|\mathbf{b}_1\| \geqslant (1-\varepsilon)\|\mathbf{b}_3\|$ *et au moins l'une des conditions suivantes est satisfaite :* $|\langle \mathbf{b}_1, \mathbf{b}_3 \rangle| \geqslant \varepsilon \|\mathbf{b}_3\|^2$, *ou* $|\langle \mathbf{b}_1, \mathbf{b}_2 \rangle + \frac{s_1}{2}\|\mathbf{b}_1\|^2| \geqslant \varepsilon \|\mathbf{b}_3\|^2$, *ou* $|\langle \mathbf{b}_3, \mathbf{b}_2 \rangle + \frac{s_3}{2}\|\mathbf{b}_1\|^2| \geqslant \varepsilon \|\mathbf{b}_3\|^2$.
4. $(x_1, x_2, x_3) = (2, s_2, s_3)$, $|s_i| = 1$ *pour* $i \in \{2, 3\}$, $\|\mathbf{b}_1\| \geqslant (1-\varepsilon)\|\mathbf{b}_3\|$ *et au moins l'une des conditions suivantes est satisfaite :* $|\langle \mathbf{b}_2, \mathbf{b}_3 \rangle| \geqslant \varepsilon \|\mathbf{b}_3\|^2$, *ou* $|\langle \mathbf{b}_2, \mathbf{b}_1 \rangle + \frac{s_2}{2}\|\mathbf{b}_1\|^2| \geqslant \varepsilon \|\mathbf{b}_3\|^2$, *ou* $|\langle \mathbf{b}_3, \mathbf{b}_1 \rangle + \frac{s_3}{2}\|\mathbf{b}_1\|^2| \geqslant \varepsilon \|\mathbf{b}_3\|^2$.

Démonstration. Sans perte de généralité, nous pouvons supposer que $\|\mathbf{b}_3\| = 1$. Nous considérons chacun des quatre cas et chaque triplet (x_1, x_2, x_3) séparément (il n'y a qu'un nombre fini de cas). La constante $\varepsilon > 0$ est choisie de telle sorte que si l'une des conditions du cas considéré n'est pas satisfaite, alors le lemme 32 devient faux. Dans ce cas, le triplet (x_1, x_2, x_3) ne peut pas être une coordonnée de Voronoï pour les vecteurs ε-G-réduits $[\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3]$. Ceci implique que $\mathrm{Dist}[V + x_1\mathbf{b}_1 + x_2\mathbf{b}_2 + x_3\mathbf{b}_3, V] > 0$, où $V = \mathrm{Vor}(\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3)$. Les faits que la fonction $\mathrm{Dist}[V + x_1\mathbf{b}_1 + x_2\mathbf{b}_2 + x_3\mathbf{b}_3, V]$ est continue par rapport aux variables $(\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3)$ et que ces variables $(\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3)$ appartiennent à un ensemble compact permettent d'obtenir le résultat. □

2.9 Qu'en est-il en plus grande dimension ?

Nous avons vu que l'algorithme glouton de réduction peut renvoyer des bases arbitrairement mauvaises à partir de la dimension 5. En utilisant les conditions de Minkowski, il est facile de voir qu'en dimensions 5 et 6 la généralisation suivante de l'algorithme glouton de réduction renvoie des bases réduites au sens de Minkowski. Aux étapes 2 et 3 de la version itérative de l'algorithme glouton de réduction, au lieu de raccourcir $\mathbf{b}_k$ en utilisant une combinaison linéaire entière des vecteurs $\mathbf{b}_1, \ldots, \mathbf{b}_{k-1}$, on s'autorise aussi l'utilisation des vecteurs $\mathbf{b}_{k+1}, \ldots, \mathbf{b}_d$. Cet algorithme est décrit à la figure 2.11. On sait très peu de choses sur cet algorithme quand la dimension est plus grande que 6. Renvoie-t-il une base Minkowski-réduite ? Existe-t-il une bonne borne sur son nombre d'itérations de boucle ? Finit-il en temps polynomial en la taille de l'entrée ?

Par ailleurs, même si l'algorithme glouton de réduction en dimension 5 ne renvoie pas forcément une base intéressante, on peut tout de même se demander s'il a une complexité quadratique. Pour faire fonctionner la technique de la section 2.6, il suffirait de montrer que l'algorithme glouton itératif de réduction en dimension 5 a un nombre d'itérations de boucle linéaire en la taille de l'entrée, c'est-à-dire de généraliser le théorème 14, page 65. En dimension inférieure ou égale à 4, pour faire cela, nous pouvions utiliser l'approche locale et l'approche globale.

Avec l'approche globale, il semble raisonnable que l'on puisse montrer que le nombre de boucles de la version récursive de l'algorithme glouton de réduction en dimension 5

Entrée : Une base $[\mathbf{b}_1, \ldots, \mathbf{b}_d]$ avec sa matrice de Gram.
Sortie : Une base Minkowski-réduite de $L(\mathbf{b}_1, \ldots, \mathbf{b}_d)$, avec sa matrice de Gram.

1. k:=2. Tant que $k \leqslant d$, répéter
2. Calculer un vecteur $\mathbf{c} \in L(\mathbf{b}_1, \ldots, \mathbf{b}_{k-1}, \mathbf{b}_{k+1}, \ldots, \mathbf{b}_d)$ le plus proche de $\mathbf{b}_k$,
3. $\mathbf{b}_k$:=$\mathbf{b}_k - \mathbf{c}$, mettre à jour la matrice de Gram,
4. Si $\|\mathbf{b}_k\| \geqslant \|\mathbf{b}_{k-1}\|$, k:=$k+1$
5. Sinon insérer $\mathbf{b}_k$ à son rang k' (les vecteurs sont rangés par longueurs croissantes), mettre à jour la matrice de Gram, k:=$k'+1$.

FIG. 2.11 – Une généralisation de l'algorithme glouton de réduction.

est linéaire en la taille de l'entrée, mais cela ne serait de toute manière pas suffisant car on voudrait borner le nombre d'itérations de la version itérative. Bien qu'insuffisant, ce résultat semble déjà difficile à obtenir. En effet, le lemme 13 (qui indique qu'une base stable par l'algorithme η-**Glouton** est plutôt orthogonale) n'est plus valide. On peut cependant isoler les mauvais cas : il s'agit essentiellement du réseau du lemme 12, page 63. Il s'agirait de montrer que si l'on n'est pas « proche » de ce mauvais cas, alors le lemme 13 reste vrai, et on pourrait ensuite utiliser le lemme 14, page refle :remain, qui demeure correct en dimension 5. Si on est dans le mauvais cas, la géométrie de la base est connue assez précisément, et il s'agirait de montrer que toutes les $O(1)$ itérations de boucle (de la variante récursive) après la fin de la η-phase, on a une décroissance significative du produit des longueurs.

L'analyse locale en dimensions 2, 3 et 4 s'appuie essentiellement sur le fait que si l'un des x_i calculés à l'étape 5 de la version récursive a une valeur absolue plus grande que 2, alors le vecteur $\|\mathbf{b}_d^{(i)}\|$ est considérablement plus court que le vecteur $\|\mathbf{a}_d^{(i)}\|$. Ce fait provient du lemme du vide (théorème 16, page 75). En dimension 4, il n'est que partiellement vérifié, mais le cas gênant (le cas « 211 ») n'arrive que très peu souvent et peut être analysé en considérant la forme très particulière des réseaux qui peuvent le faire apparaître. La situation empire en dimension 5. En effet, $(1,1,1,2)$ et $(1,1,2,2)$ — avec n'importe quel changement de signes et n'importe quelle permutation des coordonnées — sont des coordonnées de Voronoï possibles. Nous donnons ici un exemple de réseau pour lequel $(1,1,2,2)$ est une coordonnée de Voronoï :

$$\begin{bmatrix} 1 & -1 & 0 & 0 \\ 1 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

La base donnée par les vecteurs colonnes est réduite au sens de Minkowski (puisqu'elle est G-réduite), mais :

$$\|\mathbf{b}_1 + \mathbf{b}_2 + 2\mathbf{b}_3 + 2\mathbf{b}_4\| = 2 = \|\mathbf{b}_1 + \mathbf{b}_2\| \leqslant \|(2k_1+1)\mathbf{b}_1 + (2k_2+1)\mathbf{b}_2 + 2k_3\mathbf{b}_3 + 2k_4\mathbf{b}_4\|,$$

pour tous $k_1, k_2, k_3, k_4 \in \mathbb{Z}$. Remarquons que $(1,1,2,2)$ ne peut pas être une coordonnée de Voronoï stricte : si $\mathbf{b}_1 + \mathbf{b}_2 + 2\mathbf{b}_3 + 2\mathbf{b}_4$ atteint le minimum des longueurs de son orbite de $L/2L$, alors il a la même norme que $\mathbf{b}_1 + \mathbf{b}_2$. Ainsi, il est peut-être possible de contourner la difficulté pour $(1,1,2,2)$ comme on l'a fait pour le cas « 211 » en dimension 4. Cependant, le cas $(1,1,1,2)$ resterait encore à traiter, et cette coordonnée de Voronoï peut être stricte.

Chapitre II-3

L'algorithme LLL en arithmétique flottante

Le travail de recherche correspondant à ce chapitre a fait l'objet d'une publication dans les actes de la conférence Eurocrypt'05 [139]. Le présent chapitre est une version étendue de [139].

Sommaire

3.1 Rappels sur l'algorithme LLL. 102
3.1.1 Orthogonalisation et LLL-réduction. 102
3.1.2 L'algorithme LLL. 103
3.2 L'algorithme L^2. 108
3.2.1 Orthogonalisation en arithmétique flottante. 108
3.2.2 Un algorithme paresseux de proprification. 110
3.2.3 L'algorithme L^2. 111
3.3 Correction de l'algorithme L^2. 113
3.3.1 Acuité lors des calculs d'orthogonalisation. 113
3.3.2 Acuité des calculs de proprification. 117
3.3.3 Correction de l'algorithme L^2. 121
3.4 Analyse de complexité de l'algorithme L^2. 122
3.4.1 Une analyse très naïve de l'algorithme d'Euclide. 123
3.4.2 Une analyse amortie en dimension arbitraire. 124
3.5 Conclusion et problèmes ouverts. 126

L'algorithme de Lenstra, Lenstra et Lovász de réduction des réseaux (LLL ou L^3) est un outil fréquemment utilisé dans divers domaines de l'algorithmique, en particulier en théorie algorithmique des nombres [40] (pour trouver des polynômes minimaux ou travailler dans les corps de nombres, ...) et en cryptographie [130, 141] (pour casser les cryptosystèmes de type « sac-à-dos », RSA et DSA dans certaines conditions, ...). Nous décrirons dans la deuxième partie de cette thèse de nouvelles applications de cet algorithme, dans le domaine de l'arithmétique des ordinateurs.

Étant donnés d vecteurs de normes inférieures à B dans un espace de dimension n, l'algorithme LLL renvoie une base dite LLL-réduite du réseau engendré par les d vecteurs, en temps $O(d^5(d+n)\log^3 B)$, en faisant intervenir des opérations arithmétiques sur des entiers de longueur $O(d \log B)$. Bien que polynomiale en la taille de l'entrée, cette borne de complexité est trop élevée pour de nombreux réseaux intervenant en pratique, où d et/ou $\log B$ sont souvent assez grands. À cause de cela, l'algorithme LLL original n'est pratiquement jamais utilisé, sauf pour de petits réseaux (en petite dimension, avec des vecteurs assez courts). À la place, on utilise des variantes flottantes de l'algorithme, où les nombres rationnels sont approchés par des nombres flottants de tailles moindres. Malheureusement, on sait que ces variantes pratiques sont numériquement peuvent nécessiter une très grande précision pour se comporter comme escompté : avec une précision raisonnable, la transposition directe de l'algorithme en arithmétique flottante peut ne pas terminer, et la base renvoyée peut ne pas être LLL-réduite. Dans ce chapitre, nous décrivons l'algorithme L^2, qui est une nouvelle variante flottante de l'algorithme LLL, et qui renvoie toujours une base LLL-réduite, en temps $O(d^4(d+n)(d+\log B)\log B)$. Il s'agit de la première version de l'algorithme LLL dont le temps d'exécution croît quadratiquement en la taille de la donnée en dimension fixée (c'est-à-dire quadratiquement en $\log B$), sans faire appel à de l'arithmétique rapide [158]. Il s'agit donc d'une généralisation très naturelle des algorithmes d'Euclide, de Lagrange (attribué à tort à Gauss habituellement) et de l'algorithme glouton en dimensions 3 et 4 présenté au chapitre II-2.

La date charnière en ce qui concerne l'algorithmique de la réduction des réseaux est celle de 1981, avec les travaux de Lenstra sur la programmation entière [113, 114], qui reposaient sur une nouvelle technique pour réduire des réseaux (voir [113], qui est une version préliminaire de [114]). L'algorithme de Lenstra avait une complexité polynomiale en dimension fixée, ce qui suffisait dans [114]. Lovász s'est servi de cela pour développer un algorithme de réduction de complexité polynomiale (y compris en la dimension), qui a atteint une forme définitive dans le célèbre article de Lenstra, Lenstra et Lovász [112], où l'algorithme était utilisé pour factoriser les polynômes à coefficients rationnels en temps polynomial. Le nom de l'algorithme, LLL, vient de cet article.

De nombreuses améliorations de l'algorithme LLL ont été proposées par la suite. Du point de vue de la qualité de la base renvoyée, Schnorr [150] a décrit une hiérarchie d'algorithmes allant de la réduction LLL à la réduction de Hermite-Korkine-Zolotarev (voir le chapitre I-1), il s'agit de l'algorithme BKZ (Korkine-Zolotarev par blocs). Cet algorithme doit être utilisé conjointement avec un algorithme qui résout exactement le problème SVP, par exemple celui de Kannan et Helfrich [75, 88] ou l'algorithme probabiliste de Ajtai, Kumar et Sivakumar [11]. Du point de vue de l'efficacité de l'algorithme LLL, on peut distinguer deux types d'améliorations : certaines cherchent à optimiser « l'aspect algèbre linéaire » de l'algorithme, alors que d'autres cherchent à optimiser « l'aspect arithmétique ». Parmi les premières, on peut citer [94, 156, 174]. Les secondes sont souvent plus efficaces car le gain de complexité est plus élevé asymptotiquement, le coût de la mise en œuvre plus faible, et pour de nombreuses applications, le terme $\log B$ est plus grand et donc plus significatif que le terme d. Parmi celles-ci on peut citer [87, 95, 151, 153, 154]. Notre amélioration se positionne dans cette deuxième famille.

Comme nous l'avons affirmé plus haut, la complexité $O(d^5(d+n)\log^3 B)$ de l'algo-

rithme LLL original s'avère trop élevée pour de nombreuses applications. Par exemple, dans le cas de la cryptanalyse de RSA quand la moitié des bits d'un facteur du module est connue [45], il est possible d'avoir à réduire un réseau de dimension 64 avec des coefficients plus longs que le module, par exemple de 1024 bits. On a alors « $d^5(d+n)\log^3 B \approx 2^{66}$ ». En général, on utilise en pratique des variantes pour lesquelles l'arithmétique rationnelle utilisée dans le processus d'orthogonalisation de Gram-Schmidt (central dans l'algorithme LLL), est remplacée par une arithmétique flottante sur des nombres de tailles nettement inférieures. L'utilisation de l'arithmétique flottante dans l'algorithme LLL date du début des années 1980, quand cet algorithme était utilisé pour résoudre des problèmes de sac-à-dos de faibles densités issus de cryptosystèmes [34, 101], en particulier celui de Merkle et Hellman [126]. Malheureusement, cette utilisation de l'arithmétique flottante dans l'algorithme LLL n'est pas très bien comprise, et il n'existe pas de variante prouvée ayant un coût pratique raisonnable. En pratique, la variante la plus employée est celle de Schnorr et Euchner [154], qui n'a aucune garantie ni de terminaison ni de correction, et dont l'implantation la plus utilisée est celle de Shoup dans NTL [162]. Quand des phénomènes douteux surviennent, on augmente la précision jusqu'à ce que cela fonctionne, et c'est pourquoi l'algorithme LLL flottant de NTL a quatre variantes : LLL_FP (double précision), LLL_XD (double précision avec exposant stocké séparément sur un entier), LLL_QP (quadruple précision) et LLL_RR (précision arbitraire). Par exemple, des problèmes dûs à l'utilisation de l'arithmétique flottante sont apparus quand Nguyễn et Stern [140] ont montré que le cryptosystème Ajtai-Dwork [10] pouvait être cassé pour toute taille de clef réaliste, puis lorsque Nguyễn [136] a cassé les « challenges » du cryptosystème GGH [69]. Ces difficultés ont permis à l'époque d'améliorer le code de NTL.

Il existe cependant une variante flottante prouvée de l'algorithme LLL, découverte en 1988 par Schnorr [151], qui améliore significativement la complexité dans le cas le pire. La variante de Schnorr renvoie une base quasiment LLL-réduite[11], en temps $O(d^3(d+n)\log B(d+\log B)^2)$, en utilisant une précision de $O(d+\log B)$ bits pour les calculs flottants. Néanmoins, cet algorithme est essentiellement d'intérêt théorique et n'est implanté dans aucune des principales bibliothèques utilisées en théorie des nombres [1, 20, 119, 162]. Trois raisons expliquent cela : l'algorithme est difficile à décrire, les constantes cachées dans la complexité sont grandes et les calculs d'erreurs numériques sont faits de façon approchée (ce qui, en toute rigueur, ne permet de montrer que l'existence d'un algorithme et non d'expliciter l'algorithme). Par ailleurs, la précision requise pour les calculs flottants semble être supérieure à $(12d + 7\mathrm{lg}B)$ bits. Avec le même exemple que plus haut ($d = 64$ et $\mathrm{lg}B = 1024$), on obtient une précision de ≈ 7800 bits.

Dans d'autres cas, par exemple dans [95, 153], le processus d'orthogonalisation de Gram-Schmidt est remplacé par des algorithmes d'orthogonalisation plus stables numériquement en toute généralité, comme ceux de Givens (implanté dans NTL) et Householder [71, 104]. Ils sont plus coûteux mais semblent générer moins d'anomalies en pratique dans le contexte de l'algorithme LLL. D'un point de vue théorique, les processus d'orthogonalisation de Givens et Householder sont stables dans le sens inverse (la solution

[11] Plus précisément, Schnorr affaiblit la condition de propreté en autorisant les coefficients d'orthogonalisation $\mu_{i,j}$ à être légèrement plus grand que 1/2 en valeur absolue. Nous faisons de même que lui au paragraphe 3.1.1. Cela ne change pas sensiblement la qualité des bases renvoyées.

calculée est la solution exacte d'une instance proche de l'instance de départ), et pas dans le sens direct (la solution calculée serait proche de la vraie solution), alors que le processus d'orthogonalisation de Gram-Schmidt est instable dans les deux sens. Cette distinction entre sens direct et inverse dans le cas des processus de Givens et Householder est à l'origine d'erreurs[12] dans [95] et une version préliminaire de [153]. Dans l'algorithme que l'on décrit ci-dessous, nous utilisons le processus d'orthogonalisation de Cholesky (qui est exactement celui de Gram-Schmidt, mais où les produits scalaires sont déjà donnés et n'ont pas à être calculés), et nous nous servons du fait que nous ne sommes pas du tout dans un cas général : le processus d'orthogonalisation n'est appliqué qu'à des bases à peu près orthogonales. Cependant, il pourrait être intéressant de voir si notre algorithme peut être adapté voire amélioré avec les processus d'orthogonalisation de Givens et Householder, ce qui n'est pas clair pour le moment.

Dans ce chapitre, nous présentons l'algorithme L^2, une nouvelle variante flottante de l'algorithme LLL, qui est simple à décrire, utilise un modèle d'arithmétique flottante standard et renvoie des bases quasiment LLL-réduites, en temps polynomial en la taille de la donnée. Plus précisément, son temps d'exécution est $O(d^4(d+n)(d+\log B)\log B)$, et il fait intervenir des calculs sur des nombres flottants dont les mantisses ont $d\lg 3 + o(d)$ bits, ce qui est indépendant de $\log B$. Il s'agit de la première variante de l'algorithme LLL dont le temps d'exécution ne croît que quadratiquement en $\log B$, toutes les autres variantes prouvées étant cubiques (sans utiliser d'arithmétique rapide). Cette amélioration est très significative pour des réseaux où $\log B$ est plus grand que d, par exemple ceux qui apparaissent dans les calculs de polynômes minimaux [40] ou dans la méthode de Coppersmith [45] pour trouver de petites racines de polynômes (nous utilisons cette méthode due à Coppersmith dans la deuxième partie de la thèse). L'efficacité pratique de l'algorithme L^2 est illustrée au chapitre II-4. Par ailleurs l'algorithme LLL peut être considéré comme une généralisation de l'algorithme d'Euclide, de celui de Lagrange et de celui décrit au chapitre II-2, qui sont tous de complexité quadratique, et non cubique comme l'est l'algorithme LLL original. Cela fait de l'algorithme L^2 la variante « naturelle » de l'algorithme LLL du point de vue de la complexité. Nos résultats sont illustrés à la figure 3.1, dans laquelle nous avons choisi $d = \Theta(n)$, pour simplifier.

	L^3 [112]	Schnorr [151]	L^2
Précision des calculs	$O(d \log B)$	$> 12d + 7\lg B$	$d\lg 3 \approx 1.58d$
Complexité	$O(d^6 \log^3 B)$	$O(d^4(d+\log B)^2 \log B)$	$O(d^5(d+\log B)\log B)$

FIG. 3.1 – Comparaison de différentes variantes de l'algorithme LLL.

L'algorithme L^2 repose sur plusieurs améliorations, vis-à-vis de l'algorithme LLL lui-même mais aussi vis-à-vis de son analyse. D'un point de vue algorithmique, nous améliorons la qualité des calculs effectués lors du processus d'orthogonalisation de Gram-Schmidt en utilisant systématiquement la matrice de Gram, et en adaptant le processus

[12] Dans [95], l'erreur se situe à la page 17 des actes de la conférence CALC'01, au niveau des équations (1) et (2).

de proprification — qui est exactement l'algorithme de l'hyperplan le plus proche de Babai [18] — pour le rendre correct malgré l'utilisation d'une arithmétique flottante. Nous donnons des bornes qui semblent fines pour le processus d'orthogonalisation de Cholesky lié à l'algorithme LLL. Cette analyse nous a amené à découvrir des bases particulièrement mauvaises : par exemple, nous avons trouvé[13] un réseau de dimension 55 avec des entrées d'une centaine de bits pour lequel le LLL_FP de NTL (une version améliorée de l'algorithme de Schnorr et Euchner [154]) boucle indéfiniment, ce qui contredit [95] où il est affirmé que la double précision est suffisante dans [154] pour LLL-réduire les réseaux jusqu'en dimension 250 avec le processus d'orthogonalisation de Gram-Schmidt classique. Cependant, pour des bases « aléatoires », les anomalies provenant de l'utilisation d'une arithmétique flottante semblent apparaître en dimension nettement plus élevée que 55. Nous rediscuterons de ce dernier point au chapitre II-4. Par ailleurs, notre processus de proprification pourrait être utile dans d'autres contextes car il répond aux mêmes exigences que l'algorithme d'hyperplan le plus proche de Babai, qui sert à résoudre de manière approchée le problème CVP. Par exemple, cela pourrait être utile dans la variante de Micciancio [128] du cryptosystème GGH [69], dans laquelle l'algorithme de Babai est utilisé pour déchiffrer.

Du point de vue de l'analyse de l'algorithme, pour prouver la borne de complexité quadratique, nous généralisons à notre situation la cascade qui apparaît dans l'analyse de l'algorithme d'Euclide et dans l'analyse de l'algorithme du chapitre II-2. Il s'agit d'une analyse amortie des coûts des différents processus de proprification successivement effectués. Cette analyse amortie est rendue possible par l'efficacité de notre variante du processus de proprification, et ne peut pas être adaptée à l'algorithme LLL classique : la mise à jour des coefficients de Gram-Schmidt en arithmétique rationnelle s'avère déjà trop coûteuse, en raison de la taille de leurs numérateurs et dénominateurs ($O(d \log B)$ dans le cas le pire).

Nous présentons dans ce chapitre un algorithme qui permet de trouver une base réduite d'un réseau donné par des vecteurs générateurs qui ne sont pas forcément linéairement indépendants. Nous avons fait ce choix pour deux raisons : d'une part pour rendre l'algorithme plus général sans le compliquer vraiment, et d'autre part parce que nous ne connaissons qu'une seule preuve correcte d'une telle variante. En effet, l'idée d'adapter l'algorithme LLL pour des vecteurs linéairement dépendants vient de [146], où elle est donnée sans preuve, et la preuve de [40] est incorrecte. Remarquons qu'une preuve correcte d'une variante différente (de celle décrite dans [146]) de l'algorithme LLL pour des vecteurs linéairement dépendants est donnée dans [154]. Par ailleurs, de même que l'algorithme LLL, l'algorithme L^2 fonctionne sur les formes quadratiques définies positives. Il peut être adapté à toutes les formes quadratiques, comme il est possible de le faire avec l'algorithme LLL [165].

Dans la section 3.1, nous rappelons quelques résultats sur les réseaux et sur l'algorithme LLL. Nous décrivons l'algorithme L^2 à la section 3.2, et nous prouvons sa correction à la section 3.3 et bornons sa complexité à la section 3.4. La section 3.5 dresse une

[13] voir l'URL `http://www.loria.fr/~stehle/these.html`

petite liste de questions ouvertes liées à l'algorithme L^2.

3.1 Rappels sur l'algorithme LLL.

Dans cette section, nous rappelons quelques résultats de base sur les réseaux euclidiens (nous renvoyons au chapitre I-1 pour les définitions les plus élémentaires), et décrivons l'algorithme LLL classique.

3.1.1 Orthogonalisation et LLL-réduction.

Soient $\mathbf{b}_1, \ldots, \mathbf{b}_d$ des vecteurs dans $\mathbb{R}^n$, qui ne sont pas forcément linéairement indépendants. Leur orthogonalisation de Gram-Schmidt $\mathbf{b}_1^*, \ldots, \mathbf{b}_d^*$ est la famille de vecteurs orthogonaux (certains peuvent être nuls) définie récursivement de la façon suivante : le vecteur $\mathbf{b}_i^*$ est la composante du vecteur $\mathbf{b}_i$ qui est orthogonale à l'espace linéaire engendré par $\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}$. Nous avons $\mathbf{b}_i^* = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j}\mathbf{b}_j^*$ où :

$$\mu_{i,j} = \begin{cases} 0 & \text{si} \quad \mathbf{b}_j^* = \mathbf{0}, \\ \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\|\mathbf{b}_j^*\|^2} & \text{sinon.} \end{cases}$$

Pour $i \leqslant d$, nous définissons $\mu_{i,i} = 1$ si $\mathbf{b}_i^* \neq \mathbf{0}$ et $\mu_{i,i} = 0$ sinon. Le volume du réseau L engendré par les $\mathbf{b}_i$ satisfait $\mathrm{vol}\, L = \prod_{i=1, \mathbf{b}_i^* \neq \mathbf{0}}^{d} \|\mathbf{b}_i^*\|$. L'orthogonalisation de Gram-Schmidt dépend de l'ordre des vecteurs. Si les $\mathbf{b}_i$ sont à coordonnées entières, alors les coordonnées des vecteurs $\mathbf{b}_i^*$ et les coefficients $\mu_{i,j}$ sont des nombres rationnels. Les coefficients $\mu_{i,j}$ de l'orthogonalisation de Gram-Schmidt correspondent au « R » de la factorisation $Q \cdot R$ de la matrice représentant les vecteurs $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$, où Q est une matrice orthogonale et R est une matrice triangulaire. Si $R = (r_{i,j})$, on a $r_{i,j} = \mu_{i,j}\|\mathbf{b}_j^*\|$ pour tous $i \geqslant j$. En fait, dans la suite du chapitre, la notation $r_{i,j}$ désignera $\mu_{i,j}\|\mathbf{b}_j^*\|^2$. Quand nous parlerons d'orthogonalisation de Gram-Schmidt, nous ferons référence aux $\mu_{i,j}$ et aux $r_{i,j} = \mu_{i,j}\|\mathbf{b}_j^*\|^2$. Il y a de la redondance d'information en arithmétique rationnelle, mais dans le contexte des calculs flottants, il est important d'avoir toutes ces variables pour minimiser le nombre d'opérations arithmétiques effectuées par l'algorithme L^2, et donc pour minimiser la perte de précision.

Des vecteurs $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ sont dits *propres* avec un facteur $\eta \geqslant 1/2$ si leur orthogonalisation de Gram-Schmidt satisfait $|\mu_{i,j}| \leqslant \eta$ pour tous $1 \leqslant j < i \leqslant d$. Le i-ème vecteur $\mathbf{b}_i$ est dit *propre* si $|\mu_{i,j}| \leqslant \eta$ pour tout $j < i$. La proprification est en général considérée avec le facteur $\eta = 1/2$, mais il est fondamental dans notre algorithme de prendre un facteur strictement plus grand que $1/2$.

Des vecteurs $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ sont dits *LLL-réduits* avec des facteurs (δ, η) où $1/4 < \delta \leqslant 1$ et $1/2 \leqslant \eta < \sqrt{\delta}$ si les premiers sont nuls jusqu'à un certain rang ζ, puis les autres sont linéairement indépendants, propres et satisfont les $d - \zeta - 1$ conditions de Lovász :

$$\forall \kappa \in [|\zeta + 2, d|], (\delta - \mu_{\kappa,\kappa-1}^2) r_{\kappa-1,\kappa-1} \leqslant r_{\kappa,\kappa}.$$

De façon équivalente, on peut écrire $\delta\|\mathbf{b}_{\kappa-1}^*\|^2 \leqslant \|\mathbf{b}_\kappa^* + \mu_{\kappa,\kappa-1}\mathbf{b}_{\kappa-1}^*\|^2$. Ces conditions impliquent que les longueurs des vecteurs $\mathbf{b}_i^*$ ne décroissent pas trop vite (cela signifie intuitivement que les vecteurs $\mathbf{b}_{\zeta+1}, \ldots, \mathbf{b}_d$ ne sont pas très loin d'être orthogonaux). Le premier vecteur non nul d'une telle famille est raisonnablement court :

$$\|\mathbf{b}_{\zeta+1}\| \leqslant (\delta - \eta^2)^{-(d-1)/4}(\text{vol } L)^{1/d}.$$

Pour simplifier la compréhension du chapitre, le lecteur pourra supposer que les vecteurs donnés en entrée dans les divers algorithmes sont toujours linéairement indépendants. Cela revient à fixer $\zeta = 0$ pour l'ensemble du chapitre. La situation où les vecteurs peuvent être linéairement dépendants n'induit que peu de modifications, c'est pour cela que nous la considérons en même temps.

La notion de LLL-réduction est usuellement employée pour la paire de facteurs $(\delta, \eta) = (3/4, 1/2)$, qui est celle qui a été choisie initialement dans [112]. Cependant, plus δ et η sont proches de 1 et 1/2 respectivement, plus la borne sur la longueur du vecteur $\mathbf{b}_{\zeta+1}$ est intéressante. En pratique, on choisit souvent $\delta \approx 1$ et $\eta \approx 1/2$, de telle sorte que l'on a approximativement $\|\mathbf{b}_{\zeta+1}\| \leqslant (4/3)^{(d-1)/4}(\text{vol } L)^{1/d}$. L'algorithme LLL classique calcule en temps polynomial une telle famille LLL-réduite pour toute paire de facteurs $(\delta, 1/2)$, où $\delta < 1$ peut être choisi arbitrairement proche de 1. L'algorithme L^2 que nous décrivons dans ce chapitre calculera une famille LLL-réduite pour les facteurs (δ, η), où $\delta < 1$ pourra être choisi arbitrairement proche de 1 et $\eta > 1/2$ pourra être choisi arbitrairement proche de 1/2. On ne sait pas si $\delta = 1$ peut être atteint en temps polynomial (quand la dimension augmente, sinon c'est le cas : cela a été prouvé indépendamment par Akhavi dans [14] et par Lenstra dans [115]), et on ne sait pas non plus si $\eta = 1/2$ peut être atteint en temps quadratique en dimension fixée. Le cas $(\delta, \eta) = (1, 1/2)$ est très proche d'une notion de réduction décrite par Hermite [76] dans le langage des formes quadratiques (voir le chapitre I-1).

3.1.2 L'algorithme LLL.

L'algorithme LLL [112] est décrit à la figure 3.2. Il calcule une famille LLL-réduite de manière itérative : l'indice κ est tel qu'à tout moment de l'exécution de l'algorithme, la famille tronquée $(\mathbf{b}_1, \ldots, \mathbf{b}_{\kappa-1})$ est LLL-réduite. À chaque itération de boucle, l'indice κ est soit incrémenté soit décrémenté : on sort de la boucle quand $\kappa = d + 1$, auquel cas la famille $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ est LLL-réduite. L'indice ζ permet de savoir combien de vecteurs nuls on a au début de notre famille : à tout moment, les vecteurs $\mathbf{b}_1, \ldots, \mathbf{b}_\zeta$ sont nuls, et les vecteurs $\mathbf{b}_{\zeta+1}, \ldots, \mathbf{b}_d$ sont non nuls.

L'algorithme LLL effectue deux sortes d'opérations : des échanges de vecteurs consécutifs et des proprifications, dont chacune effectue au plus d translations de la forme $\mathbf{b}_\kappa := \mathbf{b}_\kappa - m\mathbf{b}_i$, où m est un entier et $i < \kappa$. Les échanges servent à garantir les conditions de Lovász, alors que les proprifications permettent de satisfaire la condition de propreté. Nous expliquons maintenant les étapes 5 à 9 : si la condition de Lovász est satisfaite à l'étape 5, les étapes 6 à 8 sont sans effet, et l'indice κ est incrémenté comme dans les descriptions plus classiques de l'algorithme LLL. Sinon, l'étape 5 sert à déterminer le bon

Entrée : Des vecteurs $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ non nuls et un facteur $\delta \in]1/4, 1[$.
Sortie : Une base LLL-réduite de $L(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ pour des facteurs $(\delta, 1/2)$.

1. κ:=2, ζ:=0, $r_{1,1}$:=$\|\mathbf{b}_1\|^2$.
2. Tant que $\kappa \leqslant d$, faire
3. Calculer les $\mu_{\kappa,j}$ et $r_{\kappa,j}$ pour $j \leqslant \kappa$.
4. Proprifier $\mathbf{b}_\kappa$ vis-à-vis de $(\mathbf{b}_1, \ldots, \mathbf{b}_{\kappa-1})$ en utilisant l'algorithme de la figure 3.3.
5. κ':=κ. Tant que $\kappa \geqslant \zeta + 2$ et $\delta r_{\kappa-1,\kappa-1} > \sum_{i=\kappa-1}^{\kappa'} \mu_{\kappa',i}^2 r_{i,i}$, faire : κ:=$\kappa - 1$.
6. $r_{\kappa,\kappa}$:=$\sum_{i=\kappa}^{\kappa'} \mu_{\kappa',i}^2 r_{i,i}$. Pour i de 1 à $\kappa - 1$, faire : $\mu_{\kappa,i}$:=$\mu_{\kappa',i}$, $r_{\kappa,i}$:=$r_{\kappa',i}$.
7. Insérer $\mathbf{b}_{\kappa'}$ entre $\mathbf{b}_{\kappa-1}$ et $\mathbf{b}_\kappa$.
8. Si $\mathbf{b}_\kappa = \mathbf{0}$, ζ:=$\zeta + 1$.
9. κ:=max$(\zeta + 2, \kappa + 1)$.
10. Renvoyer $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$.

FIG. 3.2 – L'algorithme LLL.

indice d'insertion du vecteur $\mathbf{b}_{\kappa'}$, en regroupant ainsi plusieurs échecs successifs du test de la condition de Lovász.

Entrée : Des vecteurs $(\mathbf{b}_1, \ldots, \mathbf{b}_\kappa)$ et leur orthogonalisation de Gram-Schmidt.
Sortie : Un vecteur $\mathbf{b}'_\kappa = \mathbf{b}_\kappa + \sum_{i<\kappa} x_i \mathbf{b}_i$ tel que $\mathbf{b}_{\kappa'}$ est propre relativement à $(\mathbf{b}_1, \ldots, \mathbf{b}_{\kappa-1})$, et l'orthogonalisation de Gram-Schmidt de $(\mathbf{b}_1, \ldots, \mathbf{b}'_\kappa)$.

1. Pour i de $\kappa - 1$ à 1, faire
2. $\mathbf{b}_\kappa$:=$\mathbf{b}_\kappa - \lfloor \mu_{\kappa,i} \rceil \mathbf{b}_i$.
3. Pour j de 1 à i, faire
4. $\mu_{\kappa,j}$:=$\mu_{\kappa,j} - \lfloor \mu_{\kappa,i} \rceil \mu_{i,j}$, $r_{\kappa,j}$:=$r_{\kappa,j} - \lfloor \mu_{\kappa,i} \rceil r_{i,j}$.
5. Renvoyer les $\mu_{i,j}$ pour $j \leqslant i \leqslant \kappa$.

FIG. 3.3 – L'algorithme de proprification.

Si l'algorithme LLL termine, il est clair que la sortie sera une famille LLL-réduite. Ce qui est moins évident a priori, c'est pourquoi l'algorithme LLL a une complexité polynomiale.

Théorème 17. *L'algorithme de la figure 3.2 est correct. De plus, si l'on suppose que* $d = O(B)$, *alors son exécution requiert* $O[d^3 \log B(d^3 \log^2 B + n(d + \log B)^2)] = O(d^5(d + n) \log^3 B)$ *opérations élémentaires, où* d *est le nombre de vecteurs donnés en entrée,* n *est la dimension de l'espace dans lequel on travaille, et* $B = \max_i \|\mathbf{b}_i\|$.

Démonstration. Nous commençons par montrer qu'il y a $O(d^2 \log B)$ itérations de boucle. Soient d_k le produit des k premiers $\|\mathbf{b}_i^*\|^2$ non nuls, et $D = (\Pi_{k \leqslant \dim L} d_k) \cdot \left(\Pi_{k, \mathbf{b}_k^* = \mathbf{0}} 4^k\right)$. La quantité D reflète l'évolution de l'algorithme : elle diminue de façon significative à chaque fois qu'un échange est effectué. Nous décomposons chaque insertion de vecteur d'une

étape 7 en une série d'échanges. Supposons que les vecteurs $\mathbf{b}_\kappa$ et $\mathbf{b}_{\kappa-1}$ ne satisfassent pas la condition de Lovász et qu'ils sont échangés. On distingue trois cas :

1. Les deux vecteurs $\mathbf{b}^*_{\kappa-1}$ et $\mathbf{b}^*_\kappa$ sont tous les deux non-nuls. Alors la partie droite de D est constante, et la partie gauche diminue d'un facteur $\geqslant 1/\delta$: en effet, les premiers et derniers d_k sont constants (les premiers d_k ne changent pas car aucun des termes des produits en question ne change, et les derniers ne changent pas non plus car $\|\mathbf{b}^*_{k-1}\| \cdot \|\mathbf{b}^*_k\|$ est constant), et le d_k restant n'a qu'un terme qui change, le terme $\|\mathbf{b}^*_{\kappa-1}\|^2$, qui décroît d'un facteur $\geqslant 1/\delta$.
2. Si $\mathbf{b}^*_{\kappa-1} \neq \mathbf{0}$ et $\mathbf{b}^*_\kappa = \mathbf{0}$ et si le nouveau $\mathbf{b}^*_{\kappa-1}$ est nul, alors après l'échange, on aura $\mathbf{b}^*_{\kappa-1} = \mathbf{0}$ et $\mathbf{b}^*_\kappa \neq \mathbf{0}$. Plus précisément, le nouveau $\mathbf{b}^*_\kappa$ sera exactement l'ancien $\mathbf{b}^*_{\kappa-1}$. Par conséquent, la partie gauche de D est invariante, et la partie droite diminue d'un facteur 4.
3. Si $\mathbf{b}^*_{\kappa-1} \neq \mathbf{0}$ et $\mathbf{b}^*_\kappa = \mathbf{0}$ et si le nouveau $\mathbf{b}^*_{\kappa-1}$ n'est pas nul, alors le nouveau $\mathbf{b}^*_\kappa$ est nul car la dimension du réseau $L(\mathbf{b}_1, \ldots, \mathbf{b}_\kappa)$ est constante. Ainsi, la partie droite de D est constante. En ce qui concerne la partie gauche, tous les d_k à partir d'un certain rang décroissent d'un facteur $\geqslant 1/\delta$ car aucun des $\|\mathbf{b}^*_i\|^2$ ne change sauf $\|\mathbf{b}^*_{\kappa-1}\|^2$ qui décroît d'un facteur $\geqslant 1/\delta$.

La situation $\mathbf{b}^*_{\kappa-1} = \mathbf{0}$ ne peut pas arriver, car à l'itération de boucle où le vecteur $\mathbf{b}_{\kappa-1}$ aurait été créé et placé à une position $\kappa - 1 \geqslant \zeta + 1$, la condition de Lovász avec le vecteur précédent n'aurait pas été vérifiée, et ce vecteur n'aurait pas été mis à cette position. Initialement, on a $D \leqslant (4B)^{d(d+1)}$, à chaque échange cette quantité diminue d'un facteur $\geqslant \min(1/\delta, 4) = 1/\delta$. Le nombre de tests de la condition de Lovász est inférieur à d plus le double du nombre d'échanges. Puisqu'il y a au moins un tel test par itération de boucle, le nombre d'itérations de boucle est borné par :

$$d + 2d(d+1)\log_{1/\delta}(4B) = O(d^2 \log B).$$

Dans chacune des itérations de boucle, les étapes 3 et 4 nécessitent $O(d^2)$ opérations arithmétiques sur les termes de l'orthogonalisation de Gram-Schmidt, qui sont des nombres rationnels, et $O(dn)$ opérations arithmétiques sur les coefficients des vecteurs, qui sont des entiers. L'étape 5 requiert $O(d)$ opérations arithmétiques, car la somme en question peut être mise à jour facilement d'un test de condition de Lovász au suivant. Le coût des étapes 6 à 9 est négligeable.

Bornons maintenant la taille des coefficients des vecteurs au cours de l'algorithme. Tout d'abord, les quantités $\max_{i \leqslant k} \|\mathbf{b}^*_i\|$ pour $k \leqslant d$ diminuent au cours de l'algorithme : au cours d'un échange, le nouveau $\mathbf{b}^*_{\kappa-1}$ est plus court que celui qu'il remplace, et le nouveau $\mathbf{b}^*_\kappa$ est plus court que l'ancien $\mathbf{b}^*_{\kappa-1}$ (c'est l'orthogonalisé du même vecteur, mais par rapport à plus de vecteurs). Nous prouvons que tout au long de l'algorithme, les vecteurs $\mathbf{b}_k$ sont de longueur inférieure à $\sqrt{d} \cdot B$, sauf éventuellement au cours d'une étape 4 de la figure 3.2. Au début de l'exécution de l'algorithme, c'est vrai. Supposons que l'on soit à une itération de boucle avec comme indice courant κ. Les vecteurs autres que le vecteur $\mathbf{b}_\kappa$ ne sont pas modifiés au cours de cette itération de boucle, donc comme ils étaient de longueur au plus $\sqrt{d} \cdot B$ à l'itération de boucle précédente, c'est toujours le cas.

L'argument tient aussi pour le vecteur $\mathbf{b}_\kappa$ avant l'étape 4. Après l'étape 4, le vecteur $\mathbf{b}_\kappa$ est proprifié par rapport aux $\kappa - 1$ précédents. On a donc :

$$\|\mathbf{b}_\kappa\|^2 \leqslant \|\mathbf{b}_\kappa^*\|^2 + \sum_{i<\kappa} \mu_{\kappa,i}^2 \|\mathbf{b}_i^*\|^2 \leqslant d \max_{i \leqslant \kappa} \|\mathbf{b}_i^*\|^2 \leqslant dB^2.$$

Au cours de l'étape 4, la longueur de $\mathbf{b}_\kappa$ peut augmenter, mais pas arbitrairement. Pour montrer cela, nous étudions l'évolution des coefficients $\mu_{\kappa,j}$ au cours de la proprification. Nous montrons par récurrence décroissante sur $j \in [|1, \kappa|]$ qu'au cours de cette étape, on a toujours :

$$|\mu_{\kappa,j}| \leqslant \left(\frac{3}{2}\right)^{\kappa-j} \left(\frac{\kappa}{4} + \max_{i \in [|j,\kappa|]} \left|\mu_{\kappa,i}^{(init)}\right|\right) \qquad (*)$$

où les $\mu_{\kappa,i}^{(init)}$ sont les $\mu_{\kappa,i}$ au début de l'étape 4. Pour $j = \kappa$, le résultat est immédiat car $\mu_{\kappa,\kappa}$ n'est pas modifié. Soit $j \in [|1, \kappa - 1|]$. Comme les $\kappa - 1$ premiers vecteurs sont LLL-réduits, on a $|\mu_{i,j}| \leqslant 1/2$ pour tout $i \in [|j+1, \kappa-1|]$. En utilisant l'hypothèse de récurrence et la majoration $|\lfloor \mu_{\kappa,i} \rceil| \leqslant \frac{1}{2} + |\mu_{\kappa,i}|$, on obtient qu'à tout moment de l'étape 4 :

$$\begin{aligned}
|\mu_{\kappa,j}| &\leqslant \left|\mu_{\kappa,j}^{(init)}\right| + \sum_{i=j+1}^{\kappa-1} |\mu_{i,j}| \left[\frac{1}{2} + \left(\frac{3}{2}\right)^{\kappa-i} \left(\frac{\kappa}{4} + \max_{i' \in [|i,\kappa|]} \left|\mu_{\kappa,i'}^{(init)}\right|\right)\right] \\
&\leqslant \left|\mu_{\kappa,j}^{(init)}\right| + \frac{1}{2} \sum_{i=j+1}^{\kappa-1} \left[\frac{1}{2} + \left(\frac{3}{2}\right)^{\kappa-i} \left(\frac{\kappa}{4} + \max_{i' \in [|j,\kappa|]} \left|\mu_{\kappa,i'}^{(init)}\right|\right)\right] \\
&\leqslant \frac{\kappa}{4} + \left|\mu_{\kappa,j}^{(init)}\right| + \left(\left(\frac{3}{2}\right)^{\kappa-j} - 1\right) \left(\frac{\kappa}{4} + \max_{i \in [|j,\kappa|]} \left|\mu_{\kappa,i}^{(init)}\right|\right).
\end{aligned}$$

Ceci achève la preuve de l'équation $(*)$. Elle nous permet de borner la longueur de $\mathbf{b}_\kappa$ au cours de l'étape 4. En effet :

$$\begin{aligned}
\|\mathbf{b}_\kappa\|^2 &\leqslant \sum_{j=1}^{\kappa} \mu_{\kappa,j}^2 \|\mathbf{b}_j^*\|^2 \\
&\leqslant \sum_{j=1}^{\kappa} \left(\frac{9}{4}\right)^{\kappa-j} \left(\frac{\kappa}{4} + \max_{i \in [|j,\kappa|]} \left|\mu_{\kappa,i}^{(init)}\right|\right)^2 \|\mathbf{b}_j^*\|^2 \\
&\leqslant \sum_{j=1}^{\kappa} \left(\frac{9}{4} \cdot \frac{1}{\delta - \frac{1}{4}}\right)^{\kappa-j} \left(\frac{\kappa}{4} \max_{i \in [|j,\kappa|]} \|\mathbf{b}_i^*\| + \max_{i \in [|j,\kappa|]} \left(\left|\mu_{\kappa,i}^{(init)}\right| \|\mathbf{b}_i^*\|\right)\right)^2,
\end{aligned}$$

car les vecteurs $\mathbf{b}_1, \ldots, \mathbf{b}_{\kappa-1}$ sont LLL-réduits. Puisque pour tout i, on a $\|\mathbf{b}_i^*\| \leqslant B$ et $\left|\mu_{\kappa,i}^{(init)}\right| \|\mathbf{b}_i^*\| \leqslant \left\|\mathbf{b}_\kappa^{(init)}\right\|$, on a ainsi :

$$\begin{aligned}
\|\mathbf{b}_\kappa\|^2 &\leqslant \left(\frac{\kappa}{4} B + \left\|\mathbf{b}_\kappa^{(init)}\right\|\right)^2 \sum_{j=1}^{\kappa} \left(\frac{9}{4} \cdot \frac{1}{\delta - \frac{1}{4}}\right)^{\kappa-j} \\
&= 2^{O(d)} B^2.
\end{aligned}$$

Ceci nous garantit qu'à tout moment de l'exécution de l'algorithme LLL, les coefficients (entiers) des vecteurs sont de taille $O(d + \log B)$.

Il reste encore à borner la taille des numérateurs et des dénominateurs des $\mu_{i,j}$ et des $r_{j,j}$. Remarquons que pour tout k, on a $d_k = (\text{vol } L((\mathbf{b}_i)_{i \leqslant j, \mathbf{b}_i^* \neq \mathbf{0}}))^2$, où j correspond au k-ième indice tel que $\mathbf{b}_j^* \neq \mathbf{0}$. Ainsi les d_k sont des entiers bornés par $(dB^2)^d$. Fixons maintenant un $j \geqslant 0$. Si $r_{j,j} = 0$, alors les $\mu_{i,j}$ sont tous nuls. Supposons que $\mathbf{b}_j^* \neq \mathbf{0}$. Soit k tel que $d_k = \Pi_{i \leqslant j, \mathbf{b}_i^* \neq \mathbf{0}} \|\mathbf{b}_i^*\|^2$. On a tout d'abord $r_{j,j} = d_k/d_{k-1}$, ce qui implique que le numérateur et le dénominateur de $r_{j,j}$ s'écrivent sur $O(d \log B)$ bits (nous avons supposé que $\log d = O(\log B)$). Par ailleurs, on peut écrire $\mathbf{b}_j^* = \mathbf{b}_j - \sum_{i<j, \mathbf{b}_i^* \neq \mathbf{0}} \lambda_{j,i} \mathbf{b}_i$ où les $\lambda_{j,i}$ sont solutions du système

$$\left(\langle \mathbf{b}_j, \mathbf{b}_l \rangle = \sum_{i<j, \mathbf{b}_i^* \neq \mathbf{0}} \lambda_{j,i} \langle \mathbf{b}_i, \mathbf{b}_l \rangle \right)_{l<j, \mathbf{b}_l^* \neq \mathbf{0}},$$

ce qui implique que $d_{k-1}\mathbf{b}_j^*$ est un vecteur à coefficients entiers. Ainsi, puisque $d_k \mu_{i,j} = \langle \mathbf{b}_i, d_{k-1}\mathbf{b}_j^* \rangle$, on peut borner les numérateurs et dénominateurs des $\mu_{i,j}$ par des entiers de taille $O(d \log B)$.

Si l'on récapitule, les vecteurs $\mathbf{b}_i$ ont des entrées entières d'au plus $O(d + \log B)$ bits, les $\mu_{i,j}$ et $r_{i,j}$ sont des rationnels dont les numérateurs et dénominateurs ont au plus $O(d \log B)$ bits, et les $\lfloor \mu_{i,j} \rceil$ sont des entiers de $O(d + \log B)$ bits : en effet, $|\mu_{i,j}| \leqslant \|\mathbf{b}_i\|/\|\mathbf{b}_j^*\|$, et comme les vecteurs $\mathbf{b}_1, \ldots, \mathbf{b}_j$ sont LLL-réduits, on a $\|\mathbf{b}_j^*\|^2 \geqslant \frac{1}{(\delta - 1/4)^j} \|\mathbf{b}_{\zeta+1}\|^2 \geqslant \frac{1}{(\delta-1/4)^d}$. On peut donc finalement borner le nombre d'opérations élémentaires effectuées lors de l'exécution de l'algorithme LLL par :

$$O\left[d^2 \log B \cdot d \cdot \left(d(d \log B)^2 + n(d + \log B)^2\right)\right].$$

□

Le coût de l'algorithme LLL est dominé par les opérations arithmétiques permettant de calculer et mettre à jour les coefficients de Gram-Schmidt, qui sont des rationnels avec de gros numérateurs et dénominateurs. Il est donc tentant pour accélérer les calculs de remplacer l'arithmétique rationnelle par une arithmétique flottante, c'est-à-dire de ne se servir que d'approximations flottantes des coefficients de l'orthogonalisation de Gram-Schmidt. Faire cela directement crée des problèmes numériques, et l'algorithme peut éventuellement ne pas finir (la quantité D de la preuve ne diminue pas forcément à chaque échange de vecteurs), ou renvoyer des vecteurs qui ne constituent pas une base LLL-réduite (à cause d'erreurs trop grosses sur les coefficients de Gram-Schmidt). Jusqu'ici, la seule variante flottante prouvée de l'algorithme LLL était celle de Schnorr [151], qui simule le comportement de l'algorithme LLL en utilisant des approximations flottantes des coefficients de la matrice inverse des $\mu_{i,j}$. Les nombres d'itérations de boucle et d'opérations arithmétiques se bornent comme pour l'algorithme LLL : seuls les coûts de chaque opération arithmétique liée aux coefficients de Gram-Schmidt diminuent. Au lieu de manipuler des entiers de longueur $O(d \log B)$, cet algorithme manipule des nombres flottants dont les

mantisses ont $O(d + \log B)$ bits (avec des constantes cachées assez importantes, comme nous l'avons dit plus haut), ce qui permet de faire décroître la complexité dans le cas le pire à $O(d^3(d+n)\log B(d+\log B)^2)$. Cette borne est toujours cubique en $\log B$. Cet algorithme a un intérêt essentiellement théorique et n'est pas utilisé dans les principaux outils de calcul en théorie des nombres [1, 20, 119, 162]. Dans ceux-ci, on utilise plutôt des variantes heuristiques proches de l'algorithme de Schnorr et Euchner [154].

3.2 L'algorithme L^2.

Nous décrivons ici l'algorithme L^2, qui est une variante flottante naturelle de l'algorithme LLL. Le principe général est de conserver des approximations flottantes suffisamment précises des coefficients de l'orthogonalisation de Gram-Schmidt au cours de l'exécution de l'algorithme, pour simuler assez finement l'exécution de la variante exacte de l'algorithme LLL. Les questions de précision sont essentielles lors des proprifications et pour les tests des conditions de Lovász. Il faut particulièrement se méfier des pertes de précision qui apparaissent lors des proprifications : celles-ci contiennent des opérations de la forme $x{:=}x - \lfloor x \rceil$, qui sont à l'origine de grandes annulations, et donc de pertes de précision. Dans les algorithmes LLL flottants décrits dans [151] et [153], une précision élevée (linéaire en d et en $\log B$) est requise pour garantir la correction des calculs, ce qui ralentit fortement le temps d'exécution de l'algorithme. Nous diminuons ici la précision requise (pour aboutir à une précision linéaire en d uniquement), principalement en utilisant des produits scalaires exacts et non calculés en arithmétique flottante (paragraphe 3.2.1), et en se servant d'un algorithme paresseux pour la proprification (paragraphe 3.2.2).

3.2.1 Orthogonalisation en arithmétique flottante.

Pour l'algorithme L^2, il est important d'avoir de bonnes formules pour calculer les coefficients de Gram-Schmidt. Dans [154], les formules récursives suivantes sont utilisées :

$$\mu_{i,j} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j \rangle - \sum_{k=1}^{j-1} \mu_{j,k} \cdot \mu_{i,k} \cdot \|\mathbf{b}_k^*\|^2}{\|\mathbf{b}_j^*\|^2} \text{ et } \|\mathbf{b}_i^*\|^2 = \|\mathbf{b}_i\|^2 - \sum_{j=1}^{i-1} \mu_{i,j}^2 \cdot \|\mathbf{b}_j^*\|^2.$$

Quand ils utilisent ces formules, Schnorr et Euchner [154] calculent en arithmétique flottante les produits scalaires $\langle \mathbf{b}_i, \mathbf{b}_j \rangle$, ce qui crée éventuellement des pertes de précision qui peuvent être de l'ordre de $2^{-\ell}\|\mathbf{b}_i\|\|\mathbf{b}_j\|$, où ℓ est la taille des mantisses : cela arrive en particulier quand les vecteurs $\mathbf{b}_i$ et $\mathbf{b}_j$ sont assez orthogonaux, c'est-à-dire quand leur produit scalaire est petit devant le produit de leurs normes. Cela a l'inconvénient suivant : pour assurer que la base finale est LLL-réduite, on demande des erreurs absolues sur les $\mu_{i,j}$, et donc si l'erreur est de l'ordre de $2^{-\ell}\|\mathbf{b}_i\|\|\mathbf{b}_j\|$, on a besoin de prendre une taille de mantisse en $\Omega(\log B)$ dans le cas le pire (par exemple si le vecteur $\mathbf{b}_i$ est très long, alors que $\|\mathbf{b}_j\| = O(1)$). Les solutions apportées dans [151, 153] ne résolvent pas ce problème. Nous le résolvons en conservant les valeurs exactes des produits scalaires au cours de l'exécution de l'algorithme L^2 : on les calculera exactement au début, et on les mettra à jour à chaque étape de notre algorithme, pour un coût proportionnel au coût de

la mise à jour de la base des vecteurs. Cette simple modification nous permet de n'avoir besoin que d'une précision linéaire en d pour effectuer correctement les orthogonalisations de Gram-Schmidt au cours de l'algorithme L^2. Par ailleurs, pour diminuer encore les pertes de précision, nous utilisons des formules d'orthogonalisation différentes. Celles-ci font intervenir les $r_{i,j} = \mu_{i,j}\|\mathbf{b}_j^*\|^2 = \langle \mathbf{b}_i, \mathbf{b}_j^* \rangle$ pour tous $i \geqslant j$:

$$r_{i,j} = \langle \mathbf{b}_i, \mathbf{b}_j \rangle - \sum_{k=1}^{j-1} \mu_{j,k} \cdot r_{i,k} \quad \text{et} \quad \mu_{i,j} = \frac{r_{i,j}}{r_{j,j}}.$$

La précision requise pour les calculs est moins importante aussi grâce au fait que dans chaque terme de la somme n'intervient qu'une seule multiplication au lieu de deux. Pour $i = j$, la première formule devient $r_{i,i} = \|\mathbf{b}_i\|^2 - \sum_{k=1}^{i-1} \mu_{i,k} \cdot r_{i,k}$, ce qui amène à définir $s_j^{(i)} = \|\mathbf{b}_i\|^2 - \sum_{k=1}^{j-1} \mu_{i,k} \cdot r_{i,k}$ pour tout $j \in [|1, i|]$, de telle sorte que $\|\mathbf{b}_i^*\|^2 = r_{i,i} = s_i^{(i)}$. Contrairement aux $\mu_{i,j}$ et $r_{i,j}$, à tout moment de l'exécution de l'algorithme L^2, seulement un nombre linéaire (en d) de s_i sera stocké : c'est pour cela que nous avons choisi une notation différente. Les quantités $s_j^{(i)}$ s'avèrent utiles pour vérifier plusieurs conditions de Lovász consécutives. En effet, la condition de Lovász $(\delta - \mu_{\kappa,\kappa-1}^2)\|\mathbf{b}_{\kappa-1}^*\|^2 \leq \|\mathbf{b}_\kappa^*\|^2$ peut se récrire $\delta\|\mathbf{b}_{\kappa-1}^*\|^2 \leq \|\mathbf{b}_\kappa^*\|^2 + \mu_{\kappa,\kappa-1}^2\|\mathbf{b}_{\kappa-1}^*\|^2$, c'est-à-dire

$$\delta r_{\kappa-1,\kappa-1} \leqslant s_{\kappa-1}^{(\kappa)}.$$

Si cette condition n'est pas satisfaite, l'algorithme LLL échange les vecteurs $\mathbf{b}_{\kappa-1}$ et $\mathbf{b}_\kappa$, et teste ensuite la condition de Lovász suivante :

$$\delta r_{\kappa-2,\kappa-2}^{(\kappa)} \leq s_{\kappa-2}^{(\kappa)}.$$

Ainsi, le fait de stocker les $s_j^{(\kappa)}$ permet de tester plusieurs conditions de Lovász consécutives (quand il y a des échanges consécutifs) sans avoir besoin de faire de calcul supplémentaire : on calcule les $s_j^{(\kappa)}$ pour déterminer $r_{\kappa,\kappa}$. Nous récapitulons le calcul des $r_{i,j}, \mu_{i,j}$ et $s_j^{(d)}$ dans l'algorithme de factorisation de Cholesky décrit à la figure 3.4. Pour simplifier la description de l'algorithme de factorisation de Cholesky, nous avons séparé le calcul des $s_j^{(d)}$. En fait, celui-ci est déjà effectué dans la boucle pour les indices $i = j = d$.

Bien entendu, quand on utilise une arithmétique flottante, les valeurs exactes des coefficients de Gram-Schmidt ne sont pas connues. À la place, on calcule des approximations $\bar{r}_{i,j}, \bar{\mu}_{i,j}$ et $\bar{s}_j^{(d)}$. Dans ce cas, les étapes 4 à 6 de l'algorithme de la figure 3.4 sont effectuées de la manière suivante[14] :

$$\begin{aligned}
\bar{r}_{i,j} &:= \diamond(\bar{r}_{i,j} - \diamond(\bar{\mu}_{j,k} \cdot \bar{r}_{i,k})), \\
\bar{\mu}_{i,j} &:= \diamond\left(\bar{r}_{i,j}/\bar{r}_{j,j}\right), \\
\bar{s}_j^{(d)} &:= \diamond(\bar{s}_{j-1}^{(d)} - \diamond(\bar{\mu}_{d,j-1} \cdot \bar{r}_{d,j-1})).
\end{aligned}$$

Nous n'utiliserons pas directement l'algorithme de factorisation de Cholesky dans l'algorithme L^2. En fait nous utiliserons des parties de celui-ci au cours de l'exécution de

[14] Nous rappelons que la notation $\diamond(x)$ désigne un nombre réel représentable le plus proche de x.

Entrée : La matrice de Gram des vecteurs $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$.
Sortie : Les $r_{i,j}, \mu_{i,j}$ et $s_j^{(d)}$, pour $j \leqslant i \leqslant d$.

1. Pour i de 1 à d et pour j de 1 à i, faire
2. Si $j < i$ et $r_{j,j} = 0$, alors $r_{i,j}{:=}0, \mu_{i,j}{:=}0$.
3. Sinon :
4. $r_{i,j}{:=}\langle \mathbf{b}_i, \mathbf{b}_j \rangle$,
5. Pour k de 1 à $j-1$, faire : $r_{i,j}{:=}r_{i,j} - \mu_{j,k} r_{i,k}$,
6. $\mu_{i,j}{:=}r_{i,j}/r_{j,j}$.
7. $s_1^{(d)}{:=}\|\mathbf{b}_d\|^2$. Pour j de 1 à d, faire : $s_j^{(d)}{:=}s_{j-1}^{(d)} - \mu_{d,j-1} r_{d,j-1}$.

FIG. 3.4 – L'algorithme de factorisation de Cholesky.

l'algorithme : on réalisera toujours la partie de la factorisation correspondant à un vecteur donné. On conservera le début de la factorisation de Cholesky de la matrice de Gram pour ne pas refaire ce calcul à chaque itération de boucle. L'algorithme de factorisation de Cholesky s'avérera aussi très utile pour établir la correction de l'algorithme L^2, à la section 3.3.

3.2.2 Un algorithme paresseux de proprification.

Ce qui est au centre de l'algorithme L^2 est une version paresseuse flottante de l'algorithme de proprification décrit à la figure 3.3. Au lieu de proprifier le vecteur $\mathbf{b}_\kappa$ d'un seul coup comme on le fait dans l'algorithme LLL, notre algorithme de proprification flottant réalise la même chose en plusieurs petites étapes qui utilisent l'algorithme de factorisation de Cholesky. Proprifier en une seule étape fait perdre beaucoup de précision car il s'agit d'une opération qui peut éventuellement faire perdre beaucoup de précision. Pour pallier cela, il faudrait utiliser de grandes mantisses, de longueurs éventuellement proportionnelles à $\log B$. Plutôt que faire cela, on effectue la proprification petit à petit, c'est-à-dire on diminue un peu les $\mu_{i,j}$, puis on les recalcule avec l'algorithme de la figure 3.4 pour connaître les nouveaux $\mu_{i,j}$ avec plus d'acuité, puis on recommence le procédé jusqu'à avoir des $|\mu_{i,j}|$ suffisamment petits. Si l'on considère les x_i qui auraient été calculés par l'algorithme de proprification de la figure 3.3, cela signifie intuitivement qu'on calcule d'abord les bits les plus significatifs des x_i, que l'on raffine notre connaissance des coefficients de Gram-Schmidt courants, que l'on calcule les bits suivants des x_i, ... jusqu'à avoir les x_i complètement. Cette procédure paresseuse rend l'algorithme L^2 particulièrement efficace : en général, les x_i qui apparaissent lors d'une proprification sont petits et une faible précision de calcul suffit pour les trouver, mais en faisant la proprification en un seul coup on serait obligé de prendre une grande précision pour le cas très rare où celle-ci serait nécessaire.

À l'étape 4, on utilise $\bar{\eta} = \frac{\eta+1/2}{2} \in]1/2, \eta[$ à la place de η pour prendre en considération le fait que les $\mu_{\kappa,i}$ ne sont connus que de façon approchée. À l'étape 6, il suffit de mettre à jour les produits scalaires $\langle \mathbf{b}_i, \mathbf{b}_\kappa \rangle$ pour $i \leqslant d$, en utilisant par exemple les formules

Entrée : Un facteur $\eta > 1/2$, une taille de mantisse ℓ, un indice κ, des vecteurs $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ avec leur matrice de Gram, et des flottants $\bar{r}_{i,j}$ et $\bar{\mu}_{i,j}$ pour $j \leqslant i < \kappa$.
Sortie : Des vecteurs $(\mathbf{b}_1, \ldots, \mathbf{b}_{\kappa-1}, \mathbf{b}'_\kappa, \mathbf{b}_{\kappa+1}, \ldots, \mathbf{b}_d)$ avec leur matrice de Gram, des flottants $\bar{r}_{\kappa,j}, \bar{\mu}_{\kappa,j}$ et $\bar{s}_j^{(\kappa)}$ pour $j \leqslant \kappa$, où $\mathbf{b}'_\kappa = \mathbf{b}_\kappa - \sum_{i<\kappa} x_i \mathbf{b}_i$ pour des entiers x_i et : $|\langle \mathbf{b}'_\kappa, \mathbf{b}_i^* \rangle| \leqslant \eta \|\mathbf{b}_i^*\|^2$ pour tout $i < \kappa$.

1. $\bar{\eta}$:=$\frac{\eta+1/2}{2}$. Répéter
2. Calculer les $\bar{r}_{\kappa,j}, \bar{\mu}_{\kappa,j}$ et $\bar{s}_j^{(\kappa)}$ avec l'algorithme de la figure 3.4 pour « $i = \kappa$ ».
3. Pour i de $\kappa - 1$ à 1, faire
4. Si $|\bar{\mu}_{\kappa,i}| \geqslant \bar{\eta}$, alors X_i:=$\lfloor \bar{\mu}_{\kappa,i} \rceil$, sinon X_i:=0,
5. Pour j de 1 à $i-1$, faire : $\bar{\mu}_{\kappa,j}$:=$\diamond(\bar{\mu}_{\kappa,j} - \diamond(X_i \cdot \bar{\mu}_{i,j}))$.
6. $\mathbf{b}_\kappa$:=$\mathbf{b}_\kappa - \sum_{i=1}^{\kappa-1} X_i \mathbf{b}_i$, mettre à jour $G(\mathbf{b}_1, \ldots, \mathbf{b}_d)$.
7. Jusqu'à ce que tous les X_i soient nuls.

FIG. 3.5 – L'algorithme paresseux de proprification.

suivantes :

$$\begin{aligned} \|\mathbf{b}'_\kappa\|^2 &= \|\mathbf{b}_\kappa\|^2 + \sum_{j\neq\kappa} X_j^2 \|\mathbf{b}_j\|^2 - 2\sum_{j\neq\kappa} X_j \langle \mathbf{b}_j, \mathbf{b}_\kappa \rangle + 2 \sum_{j\neq\kappa, i\neq\kappa} X_i X_j \langle \mathbf{b}_i, \mathbf{b}_j \rangle, \\ \langle \mathbf{b}_i, \mathbf{b}'_\kappa \rangle &= \langle \mathbf{b}_i, \mathbf{b}_\kappa \rangle - \sum_{j\neq\kappa} X_j \langle \mathbf{b}_i, \mathbf{b}_j \rangle \quad \text{pour} \quad i \neq \kappa. \end{aligned}$$

3.2.3 L'algorithme L^2.

L'algorithme L^2 est donné à la figure 3.6. Lors de son exécution, il n'y a pas besoin de conserver des approximations de tous les coefficients de Gram-Schmidt : comme l'algorithme est itératif, il suffit d'avoir des approximations de ces coefficients jusqu'à l'indice κ. Remarquons que le coût de la première étape est borné par $O(d^2 n \log^2 B)$, ce qui est négligeable par rapport au coût des étapes suivantes. À l'étape 4, on utilise $\bar{\delta} = \frac{\delta+1}{2} \in]\delta, 1[$ à la place de δ pour prendre en considération le fait que $\bar{r}_{\kappa-1,\kappa-1}$ et $\bar{s}_{\kappa-1}^{(\kappa)}$ ne sont connus que de manière approximative. Le résultat principal de ce chapitre est le suivant :

Théorème 18. *Soit (δ, η) tel que $1/4 < \delta < 1$ et $1/2 < \eta < \sqrt{\delta}$. Soit $c > \lg \frac{(1+\eta)^2}{\delta - \eta^2}$. Étant donnés en entrée d vecteurs non nuls $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ dans $\mathbb{Z}^n$, l'algorithme L^2 de la figure 3.6 avec une taille de mantisse $\ell = cd + o(d)$ renvoie une base LLL-réduite de $L(\mathbf{b}_1, \ldots, \mathbf{b}_d)$, pour la paire de facteurs (δ, η). Ce calcul nécessite $O(d^4(d+n)\log B(d+\log B))$ opérations élémentaires, avec $B = \max_{i\leqslant d} \|\mathbf{b}_i\|$. Plus précisément, si $\tau = O(d^2 \log B)$ est le nombre d'itérations de boucle effectuées, alors le temps d'exécution est borné par :*

$$O\left(d^2(d+n)(\tau + d\log dB)(d + \log B)\right).$$

Ce résultat doit être complété par un certain nombre de remarques. Tout d'abord, la borne de complexité de l'algorithme L^2 est meilleure que celle de [151] d'un facteur $\frac{d+\log B}{d}$.

Entrée : Une paire valide (δ, η) de facteurs (comme au théorème 18), une taille de mantisse ℓ et des vecteurs non nuls $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$.
Sortie : Une base LLL-réduite de $L(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ pour la paire de facteurs (δ, η).
Variables : Une matrice $d \times d$ entière G, deux matrices $d \times d$ flottantes $(\bar{r}_{i,j})$ et $(\bar{\mu}_{i,j})$, et un vecteur flottant $\bar{s}$.

1. Calculer exactement $G = G(\mathbf{b}_1, \ldots, \mathbf{b}_d)$.
2. $\bar{\delta}$:=$\frac{\delta+1}{2}$, κ:=2, ζ:=0, $\bar{r}_{1,1}$:=$\diamond(\langle \mathbf{b}_1, \mathbf{b}_1 \rangle)$.
3. Tant que $\kappa \leqslant d$, faire
4. Proprifier $\mathbf{b}_\kappa$ en utilisant l'algorithme de la figure 3.5. Cela met à jour l'orthogonalisation de Gram-Schmidt.
5. κ':=κ. Tant que $\kappa \geqslant \zeta + 2$ et $\diamond(\bar{\delta} \cdot \bar{r}_{\kappa-1,\kappa-1}) \geqslant \bar{s}^{(\kappa)}_{\kappa-1}$, faire : κ:=$\kappa - 1$.
6. Pour i de 1 à $\kappa - 1$, faire : $\bar{\mu}_{\kappa,i}$:=$\bar{\mu}_{\kappa',i}$, $\bar{r}_{\kappa,i}$:=$\bar{r}_{\kappa',i}$, $\bar{r}_{\kappa,\kappa}$:=$\bar{s}^{(\kappa)}_{\kappa}$.
7. Insérer $\mathbf{b}_{\kappa'}$ entre les vecteurs $\mathbf{b}_{\kappa-1}$ et $\mathbf{b}_\kappa$, et mettre à jour G exactement.
8. Si $\mathbf{b}_{\kappa'} = \mathbf{0}$, ζ:=$\zeta + 1$.
9. Sinon, κ:=max$(\zeta + 2, \kappa + 1)$.
10. Renvoyer $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$.

FIG. 3.6 – L'algorithme L^2.

Par ailleurs, on peut faire tendre la paire de facteurs (δ, η) vers la paire optimale $(1, 1/2)$. En effet, la complexité est plus précisément :

$$O\left(\frac{c^2 d^3 (d+n) \log B (d + \log B)\left(d - \log\left(\eta - \frac{1}{2}\right)\right)}{(1-\delta) \cdot \left(c - \lg \frac{(1+\eta)^2}{\delta - \eta^2}\right)} \cdot\right).$$

On peut donc choisir $\eta = 1/2 + 2^{-d}$ sans augmenter la complexité, et faire tendre δ vers 1, par exemple $\delta = 1 - \frac{1}{\log d}$, en gardant quasiment la même complexité asymptotique. En choisissant la paire de facteurs (δ, η) proche de la paire optimale, on voit que l'on peut prendre c arbitrairement proche de $\lg 3 < 1.585$. La dernière partie du théorème liée au nombre d'itérations de boucle est utile pour certaines bases données en entrée qui apparaissent assez fréquemment en pratique et pour lesquelles on sait borner le nombre d'itérations de boucle plus finement que dans le cas général. Par exemple pour les réseaux liés à la recherche de relations linéaires entre des nombres entiers ou réels, on a la borne $\tau = O(d \log B) = o(d^2 \log B)$. Finalement, le terme $o(d)$ qui apparaît dans la condition $\ell = cd + o(d)$ peut être rendu effectif (voir les hypothèses du théorème 22) et peut aussi être utilisé dans le sens inverse : si l'on effectue les calculs dans une précision fixée (par exemple en double précision), alors on pourra garantir la correction de l'algorithme jusqu'à une certaine dimension.

Pour prouver le théorème 18, nous procéderons de la manière suivante. Nous établirons la propriété de correction à la section 3.3, en étudiant successivement l'algorithme de factorisation de Cholesky pour des bases apparaissant au cours d'une LLL-réduction (théorème 19, page 113), l'algorithme de proprification paresseux (théorème 21, page 120),

pour enfin aboutir au résultat désiré au théorème 22, à la page 121. Les résultats de complexité annoncés dans le théorème 18 seront démontrés à la section 3.4. On y généralisera l'analyse amortie réalisée pour l'algorithme **Glouton** à la section 2.6 du chapitre II-2.

3.3 Correction de l'algorithme L^2.

Pour établir la correction de l'algorithme L^2, nous avons besoin d'estimer l'acuité des approximations flottantes à différents moments de l'exécution de l'algorithme.

3.3.1 Acuité lors des calculs d'orthogonalisation.

L'algorithme de Gram-Schmidt classique est connu comme étant particulièrement instable numériquement dans le cas général [21, 71, 104, 186]. Cependant, dans le cas de l'algorithme LLL nous ne sommes pas du tout dans le cas général car les vecteurs sont LLL-réduits progressivement, ce qui implique qu'il convient ici d'étudier les procédés d'orthogonalisation pour des vecteurs dont tous sauf le dernier sont LLL-réduits. Nous sommes ainsi en train d'orthogonaliser une famille de vecteurs assez orthogonale. Dans ce contexte particulier, le résultat suivant montre qu'une précision de $\approx d\lg 3$ bits suffit pour que l'algorithme de factorisation de Cholesky se comporte raisonnablement, quand la paire de facteurs (δ, η) est suffisamment proche de la paire optimale $(1, 1/2)$.

Théorème 19. *Soit (δ, η) une paire de facteurs valide, comme au théorème 18. Soit $\rho = \frac{(1+\eta)^2+\varepsilon}{\delta-\eta^2}$, avec $\varepsilon \in]0, 1]$. Soient $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ des vecteurs de $\mathbb{Z}^n$ dont la matrice de Gram est donnée en entrée à l'algorithme de factorisation de Cholesky de la figure 3.4. Supposons que les vecteurs $(\mathbf{b}_1, \ldots, \mathbf{b}_{d-1})$ soient LLL-réduits pour les facteurs (δ, η). Dans le cas où l'on effectue les calculs en arithmétique flottante avec une taille de mantisse ℓ qui satisfait $\ell \geqslant 2 + \lg d - \lg\varepsilon + d\lg\rho$, les nombres flottants renvoyés satisfont les relations suivantes, pour tous $j \leqslant i < d$:*

$$\frac{|\bar{r}_{i,j} - r_{i,j}|}{r_{j,j}} \leqslant (d2^4\rho^{-1})\rho^j \cdot 2^{-\ell} \quad et \quad |\bar{\mu}_{i,j} - \mu_{i,j}| \leqslant (d2^6\rho^{-1})\rho^j \cdot 2^{-\ell}.$$

De plus, si $M = \max_{j<d} |\mu_{d,j}|$, alors on a pour tout $j < d$:

$$\frac{|\bar{r}_{d,j} - r_{d,j}|}{r_{j,j}} \leqslant (d2^4\rho^{-1})\rho^j \cdot M \cdot 2^{-\ell} \quad et \quad |\bar{\mu}_{d,j} - \mu_{d,j}| \leqslant (d2^6\rho^{-1})\rho^j \cdot M \cdot 2^{-\ell}.$$

Finalement, si le vecteur $\mathbf{b}_d$ est propre par rapport aux précédents pour le facteur η, alors pour tout $j \leqslant d$:

$$\left|\bar{s}_j^{(d)} - s_j^{(d)}\right| \leqslant (d2^7\rho^{-1})\rho^j \cdot 2^{-\ell} \cdot r_{d,d} + d2^{-\ell} \cdot s_j^{(d)}.$$

Le deuxième ensemble d'inégalités s'avérera utile pour l'analyse de l'algorithme de proprification, alors que le dernier ensemble d'inégalités fournit des garanties pour les tests des conditions de Lovász.

Il est crucial dans le théorème que les premiers vecteurs soient LLL-réduits. Plus précisément, si la condition de Lovász ou la condition de propreté est violée, le résultat ne reste absolument plus valide. Heuristiquement, la condition de Lovász permet de garantir que lorsque l'on calcule un $r_{i,j}$ à l'aide des $r_{i,k}$ pour $k > j$, l'erreur sur $r_{i,k}$ relativement à $r_{k,k}$ ne peut être arbitrairement grande par rapport à $r_{j,j}$. La condition de propreté permet de son côté de borner l'erreur des effectuée sur $r_{i,j}$ par rapport à $r_{j,j}$, et donc indépendamment du vecteur $\mathbf{b}_i$ (celui-ci peut-être arbitrairement long par rapport au vecteur $\mathbf{b}_j$) : cela permet d'obtenir une erreur absolue sur les $\mu_{i,j}$.

Avant de donner la preuve du théorème 19, nous donnons une idée de celle-ci pour la situation où l'on a $\eta \approx 1/2$ et $\delta \approx 1$. La perte principale de précision vient de l'étape 5 (calcul de $r_{i,j}$), qui amplifie l'erreur géométriquement. Nous définissons $err_j = \max_{j \leqslant i < d} \frac{|\bar{r}_{i,j} - r_{i,j}|}{r_{j,j}}$, c'est-à-dire l'erreur sur $r_{i,j}$ relativement à $r_{j,j}$, et nous estimons la croissance de celle-ci quand l'indice j augmente. On a $err_1 \leqslant \max_{j \leqslant i < d} \frac{|\diamond(\langle \mathbf{b}_i, \mathbf{b}_1 \rangle) - \langle \mathbf{b}_i, \mathbf{b}_1 \rangle|}{\|\mathbf{b}_1\|^2} \leqslant 2^{-\ell} \max_{j \leqslant i < d} \frac{|\langle \mathbf{b}_i, \mathbf{b}_1 \rangle|}{\|\mathbf{b}_1\|^2} \leqslant 2^{-\ell-1}\eta \leqslant 2^{-\ell}$, grâce à la propriété de propreté. Nous fixons désormais $j \in [|2, d-1|]$. Le résultat pour $j = d$ s'obtient en modifiant légèrement la preuve du cas $j \leqslant d-1$, intuitivement en remplaçant « $\mathbf{b}_d$ » par « $\frac{1}{M}\mathbf{b}_d$ » dans celle-ci. Grâce à l'étape 6 (calcul de $\mu_{i,j}$), on a pour tous $i < d$ et $k < j$:

$$|\bar{\mu}_{i,k} - \mu_{i,k}| \leqslant \left|\frac{r_{k,k}}{\bar{r}_{k,k}}\right| err_k + |r_{i,k}| \left|\frac{1}{\bar{r}_{k,k}} - \frac{1}{r_{k,k}}\right| \lesssim \left(\frac{3}{2}\right) err_k,$$

où l'on a négligé les termes d'erreur secondaires et utilisé le fait que $|r_{i,k}| \lesssim \|\mathbf{b}_k\|^2/2$, qui provient de la propriété de propreté. Ceci implique que :

$$\begin{aligned} |\diamond(\bar{\mu}_{j,k} \cdot \bar{r}_{i,k}) - \mu_{j,k} r_{i,k}| &\leqslant |\bar{\mu}_{j,k} - \mu_{j,k}| \cdot |\bar{r}_{i,k}| + |\mu_{j,k}| \cdot |\bar{r}_{i,k} - r_{i,k}| \\ &\lesssim \left(\frac{5}{4}\right) err_k \cdot \|\mathbf{b}_k^*\|^2, \end{aligned}$$

où l'on a aussi négligé des termes d'erreur secondaires et utilisé la propriété de propreté deux fois. Ainsi,

$$err_j \lesssim \left(\frac{5}{4}\right) \sum_{k<j} \frac{\|\mathbf{b}_k^*\|^2}{\|\mathbf{b}_j^*\|^2} err_k \lesssim \left(\frac{5}{4}\right) \sum_{k<j} \left(\frac{4}{3}\right)^{j-k} err_k,$$

en utilisant la propriété de Lovász. Cette dernière inégalité donne finalement :

$$err_j \lesssim 3^j \cdot err_1 \lesssim 3^j 2^{-\ell}.$$

Démonstration. Nous pouvons supposer sans perte de généralité que les $d-1$ premiers vecteurs sont linéairement indépendants : comme ils sont LLL-réduits, on a forcément des vecteurs nuls puis des vecteurs linéairement indépendants, il suffit donc de décaler les indices pour pouvoir se restreindre au cas où il n'y a pas de vecteur nul. Nous commençons par la première série d'équations. Le but ici est de borner l'erreur commise lors du calcul des $r_{i,j}$. À l'étape 5 (calcul de $r_{i,j}$), nous calculons la somme $r_{i,j} - \sum_{k=1}^{j} \mu_{j,k} r_{i,k}$

séquentiellement. L'analyse d'erreur pour une telle sommation est donnée dans [78] (voir le chapitre I-2, page 21). Nous avons donc, en utilisant l'inégalité triangulaire :

$$\begin{aligned}
|\bar{r}_{i,j} - r_{i,j}| &\leqslant \frac{d2^{-\ell-1}}{1-d2^{-\ell-1}}\left[|\langle \mathbf{b}_i, \mathbf{b}_j\rangle| + \sum_{k=1}^{j-1} |\diamond(\bar{\mu}_{j,k}\cdot\bar{r}_{i,k})|\right] + \sum_{k=1}^{j-1} |\diamond(\bar{\mu}_{j,k}\cdot\bar{r}_{i,k}) - \mu_{j,k}r_{i,k}| \\
&\leqslant \frac{1}{1-d2^{-\ell-1}}\left(\sum_{k=1}^{j-1} |\diamond(\bar{\mu}_{j,k}\cdot\bar{r}_{i,k}) - \mu_{j,k}r_{i,k}| + d2^{-\ell-1}\left[|\langle \mathbf{b}_i, \mathbf{b}_j\rangle| + \sum_{k=1}^{j-1} |\mu_{j,k}r_{i,k}|\right]\right) \\
&\leqslant \frac{1}{1-d2^{-\ell-1}}\left(\sum_{k=1}^{j-1} |\diamond(\bar{\mu}_{j,k}\cdot\bar{r}_{i,k}) - \mu_{j,k}r_{i,k}| + d2^{-\ell-1}\left[|\langle \mathbf{b}_i, \mathbf{b}_j\rangle| + \sum_{k=1}^{j-1} r_{k,k}\right]\right)
\end{aligned}$$

car $\eta < 1$, ce qui implique $|\mu_{j,k}r_{i,k}| \leqslant \eta\cdot\eta r_{k,k} \leqslant r_{k,k}$. Nous montrons par récurrence sur $j < d$ la borne d'erreur suivante :

$$\forall i \in [|j, d-1|],\ \frac{|\bar{r}_{i,j} - r_{i,j}|}{r_{j,j}} \leqslant d\rho^{j-1}2^{-\ell+4}.$$

Nous définissons $err_j = \max_{i\geqslant j} \frac{|\bar{r}_{i,j}-r_{i,j}|}{r_{j,j}}$ et $E = d\rho^d 2^{-\ell+1} \leqslant \varepsilon/2$. Nous considérons d'abord le cas où $j = 1$. Nous avons :

$$|\diamond(\langle \mathbf{b}_i, \mathbf{b}_1\rangle) - \langle \mathbf{b}_i, \mathbf{b}_1\rangle| \leqslant 2^{-\ell-1}|\mu_{i,1}|r_{1,1} \leqslant 2^{-\ell-1}\eta r_{1,1} \leqslant 2^{-\ell-1}r_{1,1}.$$

Supposons maintenant que $j \in [|2, d-1|]$ et que le résultat soit déjà prouvé pour tout $k < j$. Si $k < j \leqslant i$, nous avons :

$$\begin{aligned}
|\bar{\mu}_{i,k} - \mu_{i,k}| &\leqslant (1+2^{-\ell-1})\left|\frac{\bar{r}_{i,k}}{\bar{r}_{k,k}} - \frac{r_{i,k}}{r_{k,k}}\right| + 2^{-\ell-1}\eta \\
&\leqslant err_k(1+\eta)\frac{1+2^{-\ell-1}}{1-E} + 2^{-\ell-1},
\end{aligned}$$

à cause de l'hypothèse de récurrence et du fait que $\eta < 1$. Ainsi, si $k < j \leqslant i$, nous avons :

$$\begin{aligned}
\frac{|\diamond(\bar{\mu}_{j,k}\cdot\bar{r}_{i,k}) - \mu_{j,k}r_{i,k}|}{r_{k,k}} &\leqslant (1+2^{-\ell-1})\frac{|\bar{\mu}_{j,k}\bar{r}_{i,k} - \mu_{j,k}r_{i,k}|}{r_{k,k}} + 2^{-\ell-1} \\
&\leqslant (1+2^{-\ell-1})\left[err_k(\eta + err_k) + \eta|\bar{\mu}_{j,k} - \mu_{j,k}|\right] + 2^{-\ell-1} \\
&\leqslant (1+2^{-\ell-1})err_k\left[\eta(2+\eta) + \varepsilon + \frac{2^{-\ell}}{1-E}\right] + 129\cdot 2^{-\ell-7} \\
&\leqslant err_k(\eta(2+\eta)+\varepsilon) + 257\cdot 2^{-\ell-7},
\end{aligned}$$

en utilisant l'hypothèse de récurrence et les inégalités $E \leqslant \varepsilon \leqslant 1/2, \ell \geqslant 5$ et $\eta < 1$. Par conséquent, si $j \leqslant i$, on a :

$$\frac{\sum_{k=1}^{j-1}|\diamond(\bar{\mu}_{j,k}\cdot\bar{r}_{i,k}) - \mu_{j,k}r_{i,k}|}{r_{j,j}} \leqslant (\eta(2+\eta)+\varepsilon)\sum_{k=1}^{j-1} err_k(\delta-\eta^2)^{k-j} + 257\cdot 2^{-\ell-2}(\delta-\eta^2)^{1-j},$$

en utilisant le fait que $1 - \delta + \eta^2 \in [1/4, 1[$. Avec la même inégalité nous obtenons :

$$\frac{|\langle \mathbf{b}_i, \mathbf{b}_j \rangle|}{r_{j,j}} \leqslant \frac{|r_{i,j}|}{r_{j,j}} + \sum_{k=1}^{j-1} |\mu_{j,k}| \frac{|\langle \mathbf{b}_i, \mathbf{b}_k^* \rangle|}{r_{j,j}} \leqslant \sum_{k=1}^{j} (\delta - \eta^2)^{k-j} \leqslant 4(\delta - \eta^2)^{1-j}.$$

Finalement, en utilisant les inégalités $d2^{-\ell-1} \leqslant 2^{-7}$ et $d \geqslant 2$, on obtient :

$$\begin{aligned}
err_j &\leqslant \frac{\eta(2+\eta)+\varepsilon}{1-d2^{-\ell-1}} \sum_{k=1}^{j-1} err_k (\delta - \eta^2)^{k-j} + d2^{-\ell+3}(\delta - \eta^2)^{1-j} \\
&\leqslant d2^{-\ell+3}(\delta - \eta^2)^{1-j} \left[1 + \frac{\eta(2+\eta)+\varepsilon}{1-d2^{-\ell-1}} \right]^{j-1} \\
&\leqslant d2^{-\ell+3} \left(\frac{(1+\eta)^2+\varepsilon}{\delta - \eta^2} \right)^{j-1} (1 + d2^{-\ell-1})^{j-1} \\
&\leqslant d2^{-\ell+4} \left(\frac{(1+\eta)^2}{\delta - \eta^2} \right)^{j-1},
\end{aligned}$$

où l'on a utilisé le fait que $(1+d2^{-\ell-1})^d \leqslant 2$, ce qui est impliqué par l'inégalité $d^2 2^{-\ell-1} \leqslant 1$. Le résultat sur les μ est évident si l'on utilise les faits que $E \leqslant 1/4$ et $\ell \geqslant 5$. De plus, il est facile de modifier légèrement la preuve pour obtenir les résultats annoncés pour les $r_{d,j}$ et les $\mu_{d,j}$ pour $j < d$.

Nous nous intéressons désormais au troisième ensemble d'équations. Nous supposons que le vecteur $\mathbf{b}_d$ est propre pour le facteur η vis-à-vis des vecteurs $(\mathbf{b}_1, \ldots, \mathbf{b}_{d-1})$. L'analyse est similaire à celle qui précède. Pour tout $j \leqslant d$, nous avons :

$$\begin{aligned}
|\bar{s}_j^{(d)} - s_j^{(d)}| &\leqslant \frac{d2^{-\ell-1}}{1-d2^{-\ell-1}} \left[\|\mathbf{b}_d\|^2 + \sum_{k=1}^{j-1} |\diamond(\bar{\mu}_{d,k} \cdot \bar{r}_{d,k})| \right] + \sum_{k=1}^{j-1} |\diamond(\bar{\mu}_{d,k} \cdot \bar{r}_{d,k}) - \mu_{d,k} r_{d,k}| \\
&\leqslant \frac{d2^{-\ell-1}}{1-d2^{-\ell-1}} \left(s_j^{(d)} + 2 \sum_{k=1}^{j-1} r_{k,k} \right) + d2^{-\ell} \sum_{k=1}^{j-1} r_{k,k} + d2^{-\ell+8} \sum_{k=1}^{j-1} r_{k,k} \rho^{k-1} \\
&\leqslant d2^{-\ell} s_j^{(d)} + r_{j,j} d2^{-\ell+4} (\delta - \eta^2)^{2-j} + r_{j,j} d2^{-\ell+8} \sum_{k=1}^{j-1} (\delta - \eta^2)^{k-j} \rho^{k-1} \\
&\leqslant d2^{-\ell} s_j^{(d)} + r_{j,j} d2^{-\ell+9} \rho^{j-1},
\end{aligned}$$

où nous avons utilisé les inégalités $\frac{1}{\delta - \eta^2} \leqslant \rho$ et $\ell \geqslant 5$. □

Les bornes du théorème 19 semblent être fines en pratique : quand on utilise une précision $\leq d\lg 3$ bits, l'algorithme d'orthogonalisation de Gram-Schmidt classique et celui de factorisation de Cholesky font apparaître des anomalies lors de l'exécution de l'algorithme LLL, quand on lui donne en entrée certaines bases qui sont déjà LLL-réduites. On peut par exemple considérer la base LLL-réduite formée des vecteurs lignes de la matrice $d \times d$ suivante :

$$\begin{aligned}
L_{j,j} &= (\sqrt{4/3})^{d-j} \\
L_{i,j} &= (-1)^{i-j+1} L_{j,j} \cdot \text{random}[0.49, 0.5] \quad \text{si} \quad j < i \\
L_{i,j} &= 0 \quad \text{si} \quad j > i.
\end{aligned}$$

Pour obtenir un réseau entier, on peut multiplier cette matrice par une grande constante et arrondir les entrées. Cette base est déjà LLL-réduite, et correspond au cas extrême de la première partie de la preuve du théorème 19. Avec des calculs en double précision (53 bits de mantisse), les erreurs sur les $\mu_{i,j}$ deviennent significatives (de l'ordre de 0.1) autour de la dimension 30. En ajoutant à la base un vecteur « aléatoire », nous avons réussi à faire boucler le LLL_FP de NTL en dimension 55. Nous avons aussi un exemple en dimension 35 pour lequel LLL_FP « s'aperçoit » de problèmes numériques et recommence les calculs en précision supérieure. Ces deux réseaux sont disponibles à l'URL `http://www.loria.fr/~stehle/these.html`. Ces expériences contredisent [95] où il est affirmé que l'algorithme de Schnorr-Euchner [154] suffit pour des réseaux jusqu'en dimension 250 :

> *Up to dimension* 250 *Gram-Schmidt orthogonalization of an LLL-reduced basis can be done in fpa* [floating-point arithmetic] *with some correction steps* [...].

Il semble cependant que nos réseaux soient « exceptionnellement » mauvais, et que l'algorithme LLL flottant se comporte mieux en pratique que ne le laisse supposer la situation extrême décrite ci-dessus. Nous donnons une explication heuristique à cela au paragraphe 4.4.2 du chapitre II-4, dans laquelle nous estimons la précision requise à $\approx 0.18d$ bits en « pratique » en dimension d. Cette valeur est nettement inférieure à la borne dans le cas le pire, c'est-à-dire $\approx 1.58d$.

3.3.2 Acuité des calculs de proprification.

Avant d'estimer l'acuité de la variante paresseuse de l'algorithme de proprification de la figure 3.5 que l'on utilise dans l'algorithme L^2, nous étudions la version flottante de l'algorithme classique de proprification, décrite à la figure 3.7. Nous utilisons le théorème 19 pour montrer les propriétés de correction suivantes de l'algorithme de la figure 3.7.

Entrée : Un facteur $\eta > 1/2$, une taille de mantisse ℓ, des vecteurs $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ avec leur matrice de Gram, et des nombres flottants $\bar{r}_{i,j}$ et $\bar{\mu}_{i,j}$ pour $j \leqslant i < d$.
Sortie : Des entiers $x_1, \ldots, x_{d-1} \in \mathbb{Z}$, $\mathbf{b}'_d = \mathbf{b}_d - \sum_{i<d} x_i \mathbf{b}_i$ et $G(\mathbf{b}_1, \ldots, \mathbf{b}_{d-1}, \mathbf{b}'_d)$.

1. Calculer les $\bar{\mu}_{d,j}$ pour $j < d$ avec l'algorithme de la figure 3.4 pour « $i = d$ ».
2. $\bar{\eta} := \frac{\eta + 1/2}{2}$. Pour i de $d-1$ à 1, faire
3. Si $|\bar{\mu}_{d,i}| \geqslant \bar{\eta}$, alors $x_i := \lfloor \bar{\mu}_{d,i} \rceil$, sinon $x_i := 0$,
4. Pour j de 1 à $i-1$, faire : $\bar{\mu}_{d,j} := \diamond(\bar{\mu}_{d,j} - \diamond(x_i \cdot \bar{\mu}_{i,j}))$.
5. Mettre à jour $\mathbf{b}_d$.
6. Calculer $G(\mathbf{b}_1, \ldots, \mathbf{b}_{d-1}, \mathbf{b}'_d)$ à partir de $G(\mathbf{b}_1, \ldots, \mathbf{b}_{d-1}, \mathbf{b}_d)$.

FIG. 3.7 – L'algorithme de proprification flottant classique.

Théorème 20. *Soient (δ, η) une paire de facteurs valide (comme au théorème 18) et $\rho = \frac{(1+\eta)^2+\varepsilon}{\delta-\eta^2}$ avec $\varepsilon \in]0, 1/2]$. Soient $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ des vecteurs de $\mathbb{Z}^n$ donnés en entrée à l'algorithme de la figure 3.7. Soit $B = \max_i \|\mathbf{b}_i\|$. Supposons que les vecteurs $(\mathbf{b}_1, \ldots, \mathbf{b}_{d-1})$*

soient LLL-réduits pour les facteurs (δ, η) et que les flottants $\bar{r}_{i,j}$ et $\bar{\mu}_{i,j}$ donnés en entrée sont exactement ceux qui auraient été calculés par l'algorithme de factorisation de Cholesky de la figure 3.4 avec une taille de mantisse ℓ. Soit $M = \max_j |\mu_{d,j}|$. Si ℓ satisfait $\ell \geqslant 4 + 2\lg d - \lg\varepsilon + d\lg\rho$, alors l'algorithme de la figure 3.7 calcule des entiers $x_1, \ldots, x_{d-1}$ tels que pour tout $i < d$:

$$|x_i| \leqslant 2(1+\eta)^{d-1-i}(1+M) \quad et \quad \frac{|\langle \mathbf{b}'_d, \mathbf{b}^*_i \rangle|}{\|\mathbf{b}^*_i\|^2} \leqslant \bar{\eta} + d\rho^d(1+M)2^{-\ell+6}.$$

De plus, si on a[15] *$\ell = O(d + \log B)$ et $\ell = \Omega(\sqrt{\log(d + \log M)})$, alors l'exécution de l'algorithme s'achève en temps $O(d(d+n)\ell(d+\log B))$.*

Démonstration. On définit $\mu^{(i)}_{d,j} = \mu_{d,j} - \sum_{k=i}^{d-1} x_k \mu_{k,j}$ pour $i > j$. En utilisant le théorème 19, on obtient :

$$\begin{aligned} |\diamond(x_i \cdot \bar{\mu}_{i,j}) - x_i\mu_{i,j}| &\leqslant (1+2^{-\ell-1})|x_i||\bar{\mu}_{i,j} - \mu_{i,j}| + 2^{-\ell-1}|x_i||\mu_{i,j}| \\ &\leqslant |x_i|2^{-\ell-1}\left[1 + 2^7(1+2^{-\ell-1})d\rho^{j-1}\right] \\ &\leqslant 9|x_i|d\rho^{j-1}2^{-\ell+3}, \end{aligned}$$

où l'on a utilisé les inégalités $\eta < 1$ et $\ell \geqslant 5$. À l'étape 4, nous mettons à jour $\mu_{d,j}$. Il s'agit d'une somme calculée séquentiellement. Avec la borne d'erreur pour une somme donnée dans [78] (voir aussi le chapitre II-3, page 21), nous avons :

$$\begin{aligned} |\bar{\mu}^{(i)}_{d,j} - \mu^{(i)}_{d,j}| &\leqslant \frac{d2^{-\ell-1}}{1-d2^{-\ell-1}}\left[|\mu_{d,j}| + \sum_{k=i}^{d-1} |\diamond(x_k \cdot \bar{\mu}_{k,j})|\right] + \sum_{k=i}^{d-1} |\diamond(x_k \cdot \bar{\mu}_{k,j}) - x_k\mu_{k,j}| \\ &\leqslant \frac{d2^{-\ell-1}}{1-d2^{-\ell-1}}\left[M + \sum_{k=i}^{d-1} |x_k\mu_{k,j}|\right] + \frac{1}{1-d2^{-\ell-1}} \sum_{k=i}^{d-1} |\diamond(x_k \cdot \bar{\mu}_{k,j}) - x_k\mu_{k,j}| \\ &\leqslant \frac{d2^{-\ell-1}}{1-d2^{-\ell-1}}\left[M + \sum_{k=i}^{d-1} |x_k|\right] + \frac{10d2^{-\ell+3}\rho^{j-1}}{1-d2^{-\ell-1}} \sum_{k=i}^{d-1} |x_k| \\ &\leqslant d2^{-\ell}M + d2^{-\ell+7}\rho^{j-1} \sum_{k=i}^{d-1} |x_k|. \end{aligned}$$

En choisissant $i = j+1$ on a pour $j < d$:

$$\begin{aligned} |x_j| &\leqslant \bar{\eta} + \left|\bar{\mu}^{(j+1)}_{d,j}\right| \\ &\leqslant \bar{\eta} + \left|\bar{\mu}^{(j+1)}_{d,j} - \mu^{(j+1)}_{d,j}\right| + \left|\mu^{(j+1)}_{d,j}\right| \\ &\leqslant \bar{\eta} + (1+d2^{-\ell})M + (\eta + d2^{-\ell+7}\rho^{j-1}) \sum_{k=j+1}^{d-1} |x_k| \\ &\leqslant (1+\eta+d2^{-\ell+7}\rho^{d-2})^{d-1-j}\left(\bar{\eta} + (1+d2^{-\ell})M\right) \\ &\leqslant 2(1+\eta)^{d-1-j}(1+M), \end{aligned}$$

[15] On choisira par la suite $\ell = \Theta(d)$.

où nous avons utilisé les inégalités $\bar{\eta} > 1$ et $\ell \geqslant 5$. On a ainsi prouvé la première affirmation du théorème. Par ailleurs, pour $i > j$, nous avons :

$$\begin{aligned} |\bar{\mu}_{d,j}^{(i)} - \mu_{d,j}^{(i)}| &\leqslant d2^{-\ell} M + d2^{-\ell+7}\rho^{j-1}(1+M)\sum_{k=i}^{d-1}(1+\eta)^{d-1-k} \\ &\leqslant d2^{-\ell+9}\rho^{i-2}(1+M)(1+\eta)^{d-i}, \end{aligned}$$

où nous avons utilisé l'inégalité $\eta > 1/2$. Ceci nous permet d'établir la deuxième affirmation du théorème et une borne sur les nombres que l'on manipule :

$$\begin{aligned} \frac{|\langle \mathbf{b}'_d, \mathbf{b}_i^* \rangle|}{\|\mathbf{b}_i^*\|^2} &\leqslant \bar{\eta} + d2^{-\ell+6}\rho^i(1+M)(1+\eta)^{d-i}, \\ |\bar{\mu}_{d,j}^{(i)}| &\leqslant d2^{-\ell+9}\rho^{i-2}(1+M)(1+\eta)^{d-i} + M + \sum_{k=i}^{d-1}|x_k| = 2^{O(d)}(1+M). \end{aligned}$$

Nous analysons désormais les coûts des opérations arithmétiques. En ce qui concerne les exposants des nombres flottants, les opérations les plus coûteuses sont les additions. Ainsi, les coûts concernant les exposants sont bornés par $O\left(\log\log(2^{O(d)}(1+M))\right)$, ce qui est négligeable par rapport aux coûts arithmétiques sur les mantisses. Désormais, on ne s'intéresse plus qu'aux mantisses. Le coût de l'étape 1 est borné par $O(d^2\ell^2)$. Comparé au coût de l'étape 4, le coût de l'étape 3 est négligeable. Il y a au plus $O(d^2)$ opérations arithmétiques effectuées pendant les étapes 2 à 4, chacune ayant son coût dominé par celui de la multiplication $\diamond(x_i \cdot \bar{\mu}_{i,j})$. Puisque $x_i = \left\lfloor \bar{\mu}_{d,i}^{(i+1)} \right\rceil$, chaque x_i est un entier que l'on peut représenter sur $O(\ell)$ bits (on a cela grâce à la première partie du théorème) : si $\ell < d$, alors x_i est de la forme $2^{t_i}x'_i$ pour un certain entier impair x'_i, et un certain entier $t_i = O(\log(d + \log(1+M)))$. Par conséquent, le coût des étapes 2 à 4 est borné par $O(d^2\ell^2)$. Aux étapes 5 et 6, nous avons $O(d(d+n))$ opérations arithmétiques sur des entiers. En utilisant la relation $|\langle \mathbf{b}_i, \mathbf{b}_d \rangle| \leqslant \|\mathbf{b}_i\| \cdot \|\mathbf{b}_d\|$ et notre borne sur les x_i, on voit que tous les produits scalaires impliqués sont plus petits que $2^{O(d)}B^2$. Les opérations les plus coûteuses sont du type « $x_i \cdot \langle \mathbf{b}_d, \mathbf{b}_i \rangle$ », et comme les x_i sont de taille $O(\ell)$, on peut borner le coût des étapes 5 et 6 par $O(d(d+n)\ell(d+\log B))$. □

Remarquons qu'en utilisant la relation $\log M = O(d + \log B)$ (qui vient du fait que les $d-1$ premiers vecteurs sont LLL-réduits), le théorème 20 implique que si l'on prend $\ell = O(d + \log B)$, cela est suffisant pour rendre les $|\mu_{d,i}|$ plus petits que η. Malheureusement, ceci requiert d'avoir aussi pré-calculé les $r_{i,j}$ et $\mu_{i,j}$ avec cette précision $O(d+\log B)$. Cette méthode est trop brutale pour résoudre le problème puisque $O(d + \log M)$ bits suffisent, et en général M est beaucoup plus petit que B. En effet, si l'on compare notre situation à celle de l'algorithme d'Euclide, l'analogie serait de calculer les quotients d'Euclide en regardant tous les bits des restes alors que les bits les plus significatifs suffisent.

L'algorithme paresseux de proprification de la figure 3.5 est un moyen de contourner le fait que l'on ne peut pas borner finement M à l'avance. En utilisant une taille de mantisse

de $O(d)$ bits, on peut trouver progressivement les x_i de la proprification en effectuant des étapes de « petite proprification », qui font chacune décroître $\log M$ de $\Omega(d)$, jusqu'à ce que l'on obtienne $M \leqslant \eta$. Une stratégie similaire était déjà utilisée de façon empirique dans le code de NTL [162], qui répète la proprification jusqu'à ce que plus rien ne se passe.

L'algorithme paresseux de proprification de la figure 3.5 utilise une taille de mantisse de $\ell = \left(\lg \frac{(1+\eta)^2}{\delta-\eta^2} + C\right) d + o(d)$ bits, où $C > 0$ est une constante arbitraire. Les nombres flottants donnés en entrée sont ceux qu'aurait calculé l'algorithme de factorisation de Cholesky de la figure 3.4 avec une taille de mantisse de ℓ bits. Le théorème 19 nous indique ainsi que les $\approx Cd$ bits les plus significatifs des nombres flottants donnés en entrée sont corrects (c'est-à-dire sont cohérents avec les bits significatifs des nombres approchés). Ainsi, on ne connaît pas les $r_{i,j}$ et les $\mu_{i,j}$ suffisamment bien pour pouvoir effectuer une proprification directement avec l'algorithme de la figure 3.7, mais le théorème 20 nous assure que leurs approximations suffisent pour faire décroître la longueur de $M = \max_{i<\kappa} \frac{|\langle \mathbf{b}_\kappa, \mathbf{b}_i^* \rangle|}{\|\mathbf{b}_i^*\|^2}$ d'à peu près $\approx Cd$ bits. En effectuant $O\left(1 + \frac{\log M}{d}\right)$ « petites proprifications », une proprification complète peut être atteinte. En pratique, le choix le plus judicieux de C peut dépendre du réseau considéré : plus C est grand, moins il y a de « petites proprifications », mais plus le coût arithmétique est élevé. Si l'on pense que les x_i vont être petits, il est tentant de choisir un petit C.

Théorème 21. *Soient (δ, η) une paire de facteurs valide (comme au théorème 18) et $\rho = \frac{(1+\eta)^2+\varepsilon}{\delta-\eta^2}$ avec $\varepsilon \in]0, 1/2]$. Soit $C > 0$ une constante. Soient $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ des vecteurs dans $\mathbb{Z}^n$ donnés en entrée à l'algorithme de la figure 3.7, et $B = \max_i \|\mathbf{b}_i\|$. Supposons que les vecteurs $(\mathbf{b}_1, \ldots, \mathbf{b}_{\kappa-1})$ soient LLL-réduits pour la paire de facteurs (δ, η) et que les flottants $\bar{r}_{i,j}$ et $\bar{\mu}_{i,j}$ donnés en entrée sont exactement ceux qui auraient été calculés par l'algorithme de factorisation de Cholesky de la figure 3.4 avec une taille de mantisse de ℓ bits. Soit $M = \max_{j<\kappa} |\mu_{\kappa,j}|$. Si ℓ satisfait $\ell \geqslant 7 + 2\lg d - \lg \min\left(\varepsilon, \eta - \frac{1}{2}\right) + d(C + \lg\rho)$, alors l'algorithme de la figure 3.5 renvoie une sortie correcte et les flottants $\bar{r}_{\kappa,j}$, $\bar{\mu}_{\kappa,j}$ et $\bar{s}_j^{(d)}$ renvoyés sont ceux qui auraient été renvoyés par l'algorithme de factorisation de Cholesky de la figure 3.4 avec une taille de mantisse de ℓ bits. De plus, si $\ell = O(d)$, alors le temps d'exécution est borné par $O(d(d+n)(d+\log B)(d+\log M))$.*

Démonstration. Nous commençons par montrer la correction de l'algorithme. À la dernière itération de boucle, les X_j calculés sont tous nuls, ce qui signifie que rien ne se passe au cours des étapes 3 à 6 en question. Cela implique que pour tout $j < \kappa$, on a $|\bar{\mu}_{\kappa,j}| \leqslant \bar{\eta}$. En utilisant la deuxième série d'inégalités du théorème 19, on obtient que :

$$\left(1 - d\rho^{d-2} 2^{-\ell+6}\right) \max_{j<d} |\mu_{\kappa,j}| \leqslant \bar{\eta}.$$

Avec l'hypothèse sur la taille de mantisse ℓ, on en déduit que $\frac{|\langle \mathbf{b}'_\kappa, \mathbf{b}_j^* \rangle|}{\|\mathbf{b}_j^*\|^2} \leqslant \eta$, pour tout $j < \kappa$. Cela donne aussi que les flottants $\bar{r}_{\kappa,j}, \bar{\mu}_{\kappa,j}$ et $\bar{s}_j^{(\kappa)}$ qui sont renvoyés sont ceux qu'aurait renvoyé l'algorithme de factorisation de Cholesky de la figure 3.4.

Nous considérons maintenant l'effet d'une itération de boucle sur la quantité M. Soit M_1 le « nouveau M » après l'itération de boucle. Le théorème 20 et l'hypothèse

sur ℓ donnent l'inégalité :

$$M_1 \leqslant \bar{\eta} + d\rho^d(1+M)2^{-\ell+6} \leqslant \frac{1/2+3\eta}{4} + 2^{-Cd}M.$$

Par conséquent, il y a au plus $1 + \frac{1}{Cd}\left(-\lg\frac{\eta-1/2}{4} + \lg M\right)$ itérations de boucle.

Le théorème 20 permet de borner le coût de chaque itération de boucle par $O(d^2(d+n)(d+\log B))$, parce que lors de l'exécution de l'algorithme, les entrées de la matrice de Gram sont des entiers dont les longueurs ne dépassent pas $O(d+\log B)$ bits. Le coût de l'arithmétique sur les exposants est négligeable. Le fait que nous ayons d'autres vecteurs dans la famille (plus précisément les vecteurs $\mathbf{b}_{\kappa+1}, \ldots, \mathbf{b}_d$) est aussi pris en compte dans la borne de complexité. Finalement, le coût de l'algorithme est borné par :

$$O\left(d^2(d+n)(d+\log B)\left(1+\frac{\log M}{d}\right)\right) = O(d(d+n)(d+\log B)(d+\log M)).$$

□

3.3.3 Correction de l'algorithme L^2.

Nous prouvons ici la correction de l'algorithme L^2, annoncée au théorème 18, page 111. Elle se déduit aisément du résultat suivant :

Théorème 22. *Soient (δ, η) une paire de facteurs valide (comme au théorème 18), $\rho = \frac{(1+\eta)^2+\varepsilon}{\delta-\eta^2}$ avec $\varepsilon \in]0, 1/2]$, et $C > 0$. Soient $(\mathbf{b}_1^{(1)}, \ldots, \mathbf{b}_d^{(1)})$ des vecteurs de $\mathbb{Z}^n$ donnés en entrée à l'algorithme L^2. Pour n'importe quelle itération de boucle t, soient $(\mathbf{b}_1^{(t)}, \ldots, \mathbf{b}_d^{(t)})$ les vecteurs courants au début de l'itération de boucle. Supposons que la taille de mantisse ℓ satisfasse $d^2\rho^d 2^{-\ell+7+Cd} \leqslant \min\left(\varepsilon, \eta - \frac{1}{2}\right)$. Alors nous avons les propriétés suivantes :*

1. *Les vecteurs $(\mathbf{b}_1^{(t)}, \ldots, \mathbf{b}_{\kappa(t)-1}^{(t)})$ sont LLL-réduits pour la paire de facteurs (δ, η).*
2. *Pour tout $i \leqslant d$, on a $\max_{j\leqslant i} \|\mathbf{b}_j^{(t)*}\| \leqslant \max_{j\leqslant i} \|\mathbf{b}_j^{(1)*}\|$ et $\|\mathbf{b}_i^{(t)}\| \leqslant \sqrt{d}\cdot\max_{j\leqslant i} \|\mathbf{b}_i^{(1)}\|$.*

Démonstration. Nous montrons ces résultats par récurrence sur t. Évidemment, toutes ces propriétés sont vérifiées pour $t = 1$ car alors on a $\kappa = 2$. Supposons maintenant que nous commençons la t-ième itération de boucle, pour un certain $t \geqslant 1$, et supposons les propriétés vérifiées au début de la t-ième itération de boucle. Nous allons montrer qu'elles le sont encore au début de la $(t+1)$-ième.

Nous montrons maintenant que les tests des conditions de Lovász fonctionnent comme escompté[16]. Rappelons que le vecteur $\mathbf{b}_\kappa$ est échangé avec le vecteur $\mathbf{b}_i$ pour $i < \kappa$ si et

[16]Dans l'algorithme LLLclassique, il y a un échange si et seulement si la condition de Lovász est violée pour le facteur δ (pour les indices κ et $\kappa+1$). Nous ne pouvons pas ici arriver à un comportement aussi net car nous ne connaissons les coefficients de Gram-Schmidt qu'approximativement, mais ce n'est pas un problème : s'il n'y a pas d'échange, la condition de Lovász n'était pas violée pour le facteur δ ; s'il y a un échange, elle l'était, pour un facteur légèrement plus grand que δ (dans cette preuve on a choisi $\frac{3-\delta}{2}$). Il y a donc une zone pour laquelle il peut y avoir ou non un échange, mais cela ne pose aucun problème.

seulement si pour tout $j \in [|i, \kappa-1|]$, nous avons $\bar{s}_j^{(\kappa)} \leqslant \diamond(\bar{\delta} \cdot \bar{r}_{j,j})$. Supposons d'abord que le vecteur $\mathbf{b}_\kappa$ ne soit pas échangé avec le vecteur $\mathbf{b}_j$. Le théorème 19 nous donne :

$$s_j^{(\kappa)}(1+d2^{-\ell}) \quad \geqslant \quad \bar{r}_{j,j}\left(\bar{\delta} - d\rho^{j-1}2^{-\ell+8}\right).$$

Ainsi, puisque $d^2\rho^d 2^{-\ell+7} \leqslant 1-\delta$, si un échange est réalisé lors de l'exécution de l'algorithme L^2, alors la condition de Lovász correspondante n'était pas satisfaite pour le facteur $\frac{3-\delta}{2} \in]\delta, 1[$. À l'opposé, supposons que les vecteurs $\mathbf{b}_\kappa$ et $\mathbf{b}_j$ soient échangés. Le théorème 19 nous donne :

$$s_j^{(\kappa)}(1-d2^{-\ell}) \quad \leqslant \quad r_{j,j}(\bar{\delta} + d\rho^{j-1}2^{-\ell+8}).$$

Ainsi, puisque $d^2\rho^d 2^{-\ell+7} \leqslant 1-\delta$, s'il n'y a pas d'échange, alors la condition de Lovász correspondante était satisfaite pour le facteur δ. Tout cela prouve la première affirmation du théorème pour l'itération de boucle considérée. Nous avons montré que l'exécution de l'algorithme L^2 pour le facteur $\bar{\delta}$ simule une exécution de l'algorithme LLL avec un facteur non déterminé dans l'intervalle $\left[\delta, \frac{3-\delta}{2}\right]$: si $s_{\kappa-1}^{(\kappa)} < \delta r_{\kappa-1,\kappa-1}$, il doit y avoir un échange, si $s_{\kappa-1}^{(\kappa)} > \frac{3-\delta}{2} r_{\kappa-1,\kappa-1}$, il ne doit pas y avoir d'échange, et sinon il peut y avoir un échange ou non.

En ce qui concerne la deuxième affirmation, observons que lors d'un échange entre deux vecteurs $\mathbf{b}_k$ et $\mathbf{b}_{k-1}$, on a, comme dans le cas de l'algorithme LLL classique :

- $\|\mathbf{b}_{k-1}^{*(nouveau)}\| \leqslant \|\mathbf{b}_{k-1}^{*(ancien)}\|$ à cause de la propriété de Lovász,
- $\|\mathbf{b}_k^{*(nouveau)}\| \leqslant \|\mathbf{b}_{k-1}^{*(ancien)}\|$ parce que le vecteur $\mathbf{b}_k^{*(nouveau)}$ est une projection orthogonale du vecteur $\mathbf{b}_{k-1}^{*(ancien)}$,

ce qui nous permet d'établir la première partie de la seconde affirmation du théorème. Finalement, si un vecteur $\mathbf{b}_i^{(t)}$ apparaît lors de l'exécution de l'algorithme L^2 et qu'il est propre par rapport aux vecteurs qui le précèdent, alors :

$$\|\mathbf{b}_i^{(t)}\|^2 \leqslant d \cdot \max_{j \leqslant i} \|\mathbf{b}_j^{(t)*}\|^2 \leqslant d \cdot \max_{j \leqslant i} \|\mathbf{b}_j^{(0)*}\|^2 \leqslant d \cdot \max_{j \leqslant i} \|\mathbf{b}_j^{(0)}\|^2.$$

Ceci montre la deuxième partie de l'affirmation pour $i < \kappa(t)$. Si $i \geqslant \kappa(t)$, nous considérons le plus grand $t' < t$ tel que $\kappa(t'+1) - 1 = i$. Le vecteur $\mathbf{b}_i^{(t)}$ a été créé lors de l'itération de boucle t'. Si t' n'existe pas, le vecteur en question est un vecteur initial et le résultat est immédiat. Sinon, le vecteur $\mathbf{b}_i^{(t)} = \mathbf{b}_{\kappa(t'+1)-1}^{(t'+1)}$ est propre à la $(t'+1)$-ième itération de boucle. Nous avons ainsi $\|\mathbf{b}_i^{(t)}\|^2 \leqslant d \cdot \max_{j \leqslant \kappa(t'+1)-1} \|\mathbf{b}_j^{(0)}\|^2$. Cela conclut la preuve du théorème. □

3.4 Analyse de complexité de l'algorithme L^2.

Dans la section 3.3.3, nous avons montré que l'on a une précision suffisante pour tester les conditions de Lovász de manière cohérente. Pour chaque test d'une condition de Lovász, l'indice κ est soit incrémenté soit décrémenté de 1, et quand il est décrémenté, la quantité $D = (\Pi_{k \leqslant \dim L} d_k) \cdot \left(\Pi_{k, \mathbf{b}_k^* = \mathbf{0}} 4^k\right)$ décroît au moins d'un facteur $\frac{3-\delta}{2} > 1$. Comme dans le

cas de l'algorithme LLL classique, cela permet de borner le nombre total d'itérations de boucle par $O(d^2 \log B)$.

Dans la suite de cette section, nous montrons la borne de complexité annoncée $O(d^4(d+n)\log B(d+\log B))$. Nous faisons cela en généralisant les analyses amorties de l'algorithme d'Euclide, de l'algorithme de Lagrange et de la généralisation de celui-ci en dimensions 3 et 4 (voir le chapitre II-2).

3.4.1 Une analyse très naïve de l'algorithme d'Euclide.

Comme nous l'avons mentionné dans l'introduction de ce chapitre, l'algorithme LLL est une généralisation multi-dimensionnelle de l'algorithme d'Euclide. Cependant, jusqu'alors cette analogie était incomplète : l'algorithme d'Euclide admet une borne de complexité quadratique, alors que l'algorithme LLL est cubique en dimension fixée. Intuitivement, l'analyse de l'algorithme LLL classique correspond à une analyse très naïve de l'algorithme d'Euclide qui donne aussi une borne de complexité cubique (il ne semble pas possible de faire mieux pour l'algorithme LLL classique).

L'algorithme d'Euclide fonctionne comme suit : étant donnés en entrée deux entiers $r_0 > r_1 > 0$, l'algorithme d'Euclide consiste à calculer les quotients q_i et les restes r_i définis par :

$$q_i = \lfloor r_{i-1}/r_i \rfloor \quad \text{et} \quad r_{i+1} = r_{i-1} - q_i r_i,$$

en s'arrêtant quand $r_{\tau+1} = 0$ pour un certain τ. Alors r_τ est le plus grand diviseur commun de r_0 et r_1. Il est classique que les restes décroissent de manière au moins géométrique de telle sorte que le nombre de divisions et de multiplications effectuées est borné par $\tau = O(\log r_0)$. Une analyse naïve de l'algorithme d'Euclide consiste à dire que l'on effectue $O(\log r_0)$ opérations arithmétiques sur des entiers de longueur au plus $O(\log r_0)$, ce qui donne une borne de complexité $O(\log^3 r_0)$. Cette analyse est fine pour chaque étape élémentaire (une étape donnée peut effectivement coûter $\Omega(\log^2 r_0)$ opérations élémentaires), mais pas pour une suite d'étapes : les « mauvaises » étapes ne peuvent pas se suivre indéfiniment. L'analyse amortie classique est plus subtile. Elle consiste à borner le coût du calcul de q_i et r_{i+1} par $O(\log r_{i-1} \cdot (1 + \log q_i)) = O(\log r_0 \cdot (1 + \log r_{i-1} - \log r_i))$. Lorsque l'on somme cette quantité sur l'ensemble des étapes, les termes « $\log r_i$ » disparaissent (sauf le premier et le dernier), ce qui amène à la borne de complexité quadratique.

Cette analyse amortie de l'algorithme d'Euclide ne peut malheureusement pas être étendue à l'algorithme LLL classique, car les coefficients de Gram-Schmidt manipulés sont trop gros (dans l'algorithme d'Euclide, cela reviendrait à calculer les quotients avec des nombres rationnels avant de les arrondir en des entiers). De manière étonnante, l'analyse amortie de l'algorithme d'Euclide peut être étendue à l'algorithme L^2 : cela ne serait pas possible si l'on avait une taille de mantisse qui croît linéairement en $\log B$, d'où l'importance de l'algorithme paresseux de proprification. La difficulté principale est de généraliser le phénomène d'annulation de presque tous les termes de la somme des coûts des itérations de boucle successives, d'une manière similaire à l'analyse de complexité que l'on a décrite au chapitre II-2.

3.4.2 Une analyse amortie en dimension arbitraire.

Dans cette section, nous achevons la preuve du théorème 18. Nous avons déjà montré que le nombre d'itérations de boucle τ est borné par $O(d^2 \log B)$. Grâce au théorème 21, on sait que la t-ième itération de boucle a un coût borné par $O(d(d+n)(d+\log B)(d+\log M(t)))$, où $M(t) = \max_{j<\kappa(t)} |\mu_{\kappa(t),j}(t)|$. Par analogie avec l'algorithme d'Euclide, nous faisons apparaître une annulation dans la somme sur les étapes t des quantités « $\log M(t)$ ». Pour cela, nous définissons l'indice $\alpha(t)$ comme le plus petit indice lié à un échange depuis la dernière fois que l'indice κ a valu au moins sa valeur courante $\kappa(t)$. Le lemme suivant est l'analogue en dimension quelconque du lemme 15, page 69.

Lemme 37. *Soit t une itération de boucle. Soient $\phi(t) = \max(t' < t, \kappa(t') \geqslant \kappa(t))$ si cela existe et 1 sinon, et $\alpha(t) = \min(\kappa(t'), t' \in [|\phi(t), t[|) - 1$. Alors nous avons :*

$$\log M(t) \leqslant O(d) + \log \|\mathbf{b}^{(t)}_{\kappa(t)}\| - \log \|\mathbf{b}^{(t)}_{\alpha(t)}\|.$$

Démonstration. Entre les itérations de boucle $\phi(t)$ et t les vecteurs $\mathbf{b}_1, \ldots, \mathbf{b}_{\alpha(t)-1}$ ne changent pas, et grâce aux proprifications, chaque vecteur créé pendant ces itérations de boucle sera propre vis-à-vis des vecteurs $\mathbf{b}_1, \ldots, \mathbf{b}_{\alpha(t)-1}$. Cela inclut le vecteur $\mathbf{b}^{(t)}_{\kappa(t)}$. Par conséquent, en utilisant le fait que les vecteurs $(\mathbf{b}^{(t)}_1, \ldots, \mathbf{b}^{(t)}_{\kappa(t)-1})$ sont LLL-réduits (d'après le théorème 22, page 121), on a :

$$\begin{aligned}
M(t) = \max_{i<\kappa(t)} |\mu_{\kappa(t),i}| &= \max\left(\max_{i<\alpha(t)} |\mu_{\kappa(t),i}|, \max_{i\in[|\alpha(t),\kappa(t)-1|]} |\mu_{\kappa(t),i}|\right) \\
&\leqslant \max\left(\eta, \max_{i\in[|\alpha(t),\kappa(t)-1|]} \frac{\|\mathbf{b}^{(t)}_{\kappa(t)}\|}{\|\mathbf{b}^{(t)*}_i\|}\right) \\
&\leqslant \eta + (\delta-\eta^2)^{-(\kappa(t)-\alpha(t))/2} \frac{\|\mathbf{b}^{(t)}_{\kappa(t)}\|}{\|\mathbf{b}^{(t)*}_{\alpha(t)}\|} \\
&\leqslant \eta + \sqrt{d}(\delta-\eta^2)^{-d/2} \frac{\|\mathbf{b}^{(t)}_{\kappa(t)}\|}{\|\mathbf{b}^{(t)}_{\alpha(t)}\|},
\end{aligned}$$

car $\|\mathbf{b}^{(t)}_{\alpha(t)}\| \leqslant \sqrt{d} \max_{i\leqslant\alpha(t)} \|\mathbf{b}^{(t)*}_i\| \leqslant \sqrt{d}(\delta-\eta^2)^{-\alpha(t)/2}\|\mathbf{b}^{(t)*}_{\alpha(t)}\|$. □

Nous allons maintenant partager la somme des « $\log M(t)$ » prise sur les itérations de boucle successives en $d-1$ sous-sommes disjointes, suivant la valeur de $\kappa(t)$:

$$\sum_{t\leqslant\tau} \left[O(d) + \log \|\mathbf{b}^{(t)}_{\kappa(t)}\| - \log \|\mathbf{b}^{(t)}_{\alpha(t)}\|\right] \leqslant O(\tau d) + \sum_{k=2}^{d} \sum_{t,\kappa(t)=k} \left[\log \|\mathbf{b}^{(t)}_k\| - \log \|\mathbf{b}^{(t)}_{\alpha(t)}\|\right].$$

Pour chacune de ces sous-sommes, nous gardons $k-1$ termes positifs et $k-1$ termes négatifs, et faisons s'annuler les autres entre eux progressivement. Des termes proportionnels à d apparaissent au cours de ces annulations, mais ils seront inclus dans le « $O(\tau d)$ ». Le point essentiel de l'analyse est le suivant. Il est l'analogue en dimension quelconque du lemme 16, page 69.

Lemme 38. *Soient $k \in [|2, d|]$ et $t_1 < \ldots < t_k$ des itérations de boucle de l'algorithme L^2 telles que pour tout $j \leqslant k$, on a $\kappa(t_j) = k$. Alors il existe $j < k$ tel que :*

$$\|\mathbf{b}_k^{(t_k)}\| \leqslant d(\delta - \eta^2)^d \|\mathbf{b}_{\alpha(t_j)}^{(t_j)}\|.$$

Pour prouver ce lemme, nous avons besoin du résultat technique suivant :

Lemme 39. *Soient T et j des entiers tels que $\kappa(T) \geqslant j \geqslant \kappa(T+1)$. Alors on a :*

$$\max_{i \leqslant j} \|\mathbf{b}_i^{(T+1)*}\| \leqslant \max_{i<j} \|\mathbf{b}_i^{(T)*}\| \quad et \quad \max_{i \leqslant j} \|\mathbf{b}_i^{(T+1)}\| \leqslant \sqrt{d} \cdot \max_{i<j} \|\mathbf{b}_i^{(T)}\|.$$

Démonstration. Si $i \leqslant j$ est différent de $\kappa(T+1) - 1$, alors le vecteur $\mathbf{b}_i^{(T+1)}$ est un vecteur $\mathbf{b}_{i'}^{(T)}$ pour un certain $i' \leqslant j - 1$. Grâce à la proprification et au théorème 22, il suffit de montrer que $\|\mathbf{b}_{\kappa(T+1)-1}^{(T+1)*}\| \leqslant \max_{i<j} \|\mathbf{b}_i^{(T)*}\|$. Puisque le test de la condition de Lovász a échoué à l'itération de boucle T pour l'indice $\kappa(T+1) - 1$, nous avons :

$$\|\mathbf{b}_{\kappa(T+1)-1}^{(T+1)*}\| \leqslant \|\mathbf{b}_{\kappa(T+1)-1}^{(T)*}\|.$$

□

Démonstration du lemme 38. Nous choisissons $j = \max(i \leqslant k, \alpha(t_i) \geqslant i)$. Cette définition est possible car l'ensemble maximisé n'est pas vide (il contient au moins $i = 1$). Puisque $\alpha(t_k) < k$ et $\kappa(t_k) = \kappa(t_{k-1}) = k$, il existe une première itération de boucle $T_k \in [|t_{k-1}, t_k - 1|]$ telle que $\kappa(T_k) \geqslant k \geqslant \kappa(T_k + 1)$. Grâce au théorème 22 (pour la première inégalité) et au lemme 39 (pour la seconde), on a les relations :

$$\|\mathbf{b}_k^{(t_k)}\| \leqslant \sqrt{d} \cdot \max_{i \leqslant k} \|\mathbf{b}_i^{(T_k+1)}\| \leqslant d \cdot \max_{i \leqslant k-1} \|\mathbf{b}_i^{(T_k)}\|.$$

Par définition de T_k, les vecteurs $\mathbf{b}_1, \ldots, \mathbf{b}_{k-1}$ ne changent pas entre les itérations de boucle t_{k-1} et T_k. Ainsi, comme les vecteurs $\mathbf{b}_1^{(t_{k-1})}, \ldots, \mathbf{b}_{k-1}^{(t_{k-1})}$ sont LLL-réduits, on a :

$$\|\mathbf{b}_k^{(t_k)}\| \leqslant d \cdot \max_{i \leqslant k-1} \|\mathbf{b}_i^{(t_{k-1})}\| \leqslant d(\delta - \eta^2)^{d/2} \cdot \max_{i \leqslant k-1} \|\mathbf{b}_i^{(t_{k-1})*}\|.$$

Si $j = k - 1$, nous avons le résultat escompté car dans ce cas :

$$\max_{i \leqslant k-1} \|\mathbf{b}_i^{(t_{k-1})*}\| \leqslant (\delta - \eta^2)^{d/2} \|\mathbf{b}_{\alpha(t_{k-1})}^*\| \leqslant (\delta - \eta^2)^{d/2} \|\mathbf{b}_{\alpha(t_{k-1})}\|.$$

Sinon, il existe une première itération de boucle $T_{k-1} \in [|t_{k-2}, t_{k-1}[|$ telle que $\kappa(T_{k-1}) \geqslant k - 1 \geqslant \kappa(T_{k-1} + 1)$. En utilisant le lemme 39, nous obtenons :

$$\begin{aligned}
\|\mathbf{b}_k^{(t_k)}\| &\leqslant d(\delta - \eta^2)^{d/2} \cdot \max_{i \leqslant k-1} \|\mathbf{b}_i^{(T_{k-1}+1)*}\| \\
&\leqslant d(\delta - \eta^2)^{d/2} \cdot \max_{i \leqslant k-2} \|\mathbf{b}_i^{(T_{k-1})*}\| \\
&\leqslant d(\delta - \eta^2)^{d/2} \cdot \max_{i \leqslant k-2} \|\mathbf{b}_i^{(t_{k-2})*}\|
\end{aligned}$$

Si $j = k-2$, nous avons le résultat escompté, sinon nous continuons ce procédé pour finalement obtenir le résultat. □

On peut maintenant achever l'analyse de complexité. Soient $k \in [|2, d|]$ et $t_1 < \ldots < t_{\tau_k} = \{t \leqslant \tau, k(t) = k\}$. Nous prenons parmi les termes de la somme complète ceux qui correspondent à ces itérations de boucle. Le théorème 22 permet d'obtenir :

$$\sum_{i=1}^{\tau_k} \log \frac{\|\mathbf{b}_k^{(t_i)}\|}{\|\mathbf{b}_{\alpha(t_i)}^{(t_i)}\|} \leqslant (k-1)\log(\sqrt{d}\cdot B) + \sum_{i=k}^{\tau_k} \log \|\mathbf{b}_k^{(t_i)}\| - \sum_{i=1}^{\tau_k} \log \|\mathbf{b}_{\alpha(t_i)}^{(t_i)}\|.$$

Le lemme 38 nous permet de borner finement la partie droite de la quantité ci-dessus. Tout d'abord, nous l'utilisons avec $t_1, \ldots, t_k$. Ceci montre qu'il existe $j < k$ tel que $\|\mathbf{b}_k^{(t_k)}\| \leqslant d(\delta-\eta^2)^d \|\mathbf{b}_{\alpha(t_j)}^{(t_j)}\|$. Les termes d'indices « $i = k$ » dans les termes positifs et « $i = j$ » dans les termes négatifs s'annulent et un terme « $-d\log(\delta-\eta^2) + \log d$ » apparaît à la place. Nous utilisons alors de nouveau le lemme 38 avec t_{k+1} et les $k-1$ premiers t_i pour lesquels le terme « $-\log\|\mathbf{b}_{\alpha(t_i)}^{(t_i)}\|$ » est encore présent (c'est-à-dire tous sauf t_j). On voit que t_{k+1} est supérieur à chacun de ceux-là, et donc on peut créer une nouvelle annulation entre un terme positif et un terme négatif, qui fait apparaître un autre terme « $-d\log(\delta-\eta^2) + \log d$ ». Nous effectuons cela $\tau_k - k + 1$ fois. Il reste donc $k-1$ termes négatifs, que nous majorons en utilisant le fait que l'on manipule un réseau entier. Nous obtenons ainsi :

$$\sum_{i=1}^{\tau_k} \log \frac{\|\mathbf{b}_k^{(t_i)}\|}{\|\mathbf{b}_{\alpha(t_i)}^{(t_i)}\|} \leqslant (k-1)\log(\sqrt{d}\cdot B) + \tau_k\left[-d\log(\delta-\eta^2) + \log d\right].$$

Le fait que $\sum_k \tau_k = \tau$ donne finalement :

$$\sum_{t\leqslant\tau} (O(d) + \log M(t)) = O(\tau d + d^2 \log dB).$$

3.5 Conclusion et problèmes ouverts.

Dans ce chapitre, nous avons décrit l'algorithme L^2. Il s'agit d'un algorithme qui produit une base LLL-réduite d'un réseau euclidien, à partir de vecteurs qui l'engendrent. Il utilise de l'arithmétique flottante avec des tailles de mantisse qui croissent raisonnablement avec la taille de l'entrée. Nous avons montré que son temps d'exécution est borné par $O(d^4(d+n)(d+\log B)\log B)$, où d est le nombre de vecteurs donné en entrée, n est la dimension de ces vecteurs, et B est le maximum de leurs normes euclidiennes. Pour prouver la correction de l'algorithme, l'élément principal a été l'étude de son comportement quand on utilise une arithmétique flottante, et, pour aboutir à la borne de complexité ci-dessus, nous avons mis en œuvre une analyse amortie. Nous étudierons cet algorithme d'un point de vue expérimental au chapitre II-4.

L'algorithme L^2 et l'amélioration de la borne de complexité qu'il induit par rapport à l'algorithme LLL amènent à se poser un certain nombre de questions.

1. La précision des calculs annoncée — une taille de mantisse de $d\lg 3 + o(d)$ bits — est-elle la précision minimale requise pour effectuer une LLL-réduction ? Est-il par exemple possible d'améliorer la présente analyse ? Ou d'isoler les mauvais cas de l'analyse pour les traiter différemment ? Peut-être peut-on utiliser un autre algorithme d'orthogonalisation que celui de Cholesky ? Par exemple, il semble [153] que l'orthogonalisation de Householder se comporte mieux que la factorisation de Cholesky sur les pires cas de notre analyse, mais il n'est pas clair que les mauvais cas soient les mêmes pour les deux algorithmes. Nous revenons sur ce point au chapitre II-4.
2. Les idées utilisées pour notre algorithme peuvent-elles être combinées avec les améliorations de l'algorithme LLL qui reposent sur des économies d'opérations d'algèbre linéaire, comme [94, 156] ? Peut-on construire un algorithme « Segment-L^2 » ?
3. Est-il possible de réutiliser les idées de l'algorithme L^2 avec des algorithmes qui produisent de meilleures bases, en particulier l'algorithme BKZ de Schnorr [150] ?
4. On a vu que l'algorithme LLL généralise l'algorithme d'Euclide et l'algorithme de Lagrange en dimension quelconque. L'algorithme L^2 rend cette généralisation plus complète puisqu'elle devient aussi valide pour les temps d'exécution. Nous avons aussi vu au chapitre II-1 qu'il existe des algorithmes de calcul du plus grand diviseur commun de deux entiers dont la complexité est quasi-linéaire en la taille de l'entrée. Des algorithmes similaires existent aussi en dimension 2 [157, 189]. Il est donc naturel de se demander s'il est possible de construire un algorithme qui LLL-réduit des bases en temps quasi-linéaire en $\log B$. Par exemple, peut-on atteindre une borne de complexité du type[17] $\mathcal{P}\mathrm{ol}(d, n)O(M(\log B)\log\log B)$? Dans [57], Eisenbrand et Rote décrivent un algorithme permettant (entre autres) de LLL-réduire une base en temps quasi-linéaire en toute dimension fixée, mais le coût de cet algorithme est exponentiel en la dimension.

[17] Nous rappelons que $M(n) = O(n \log n \log\log n)$ est le temps requis pour multiplier deux entiers de n bits chacun avec l'algorithme de Schönhage et Strassen [158]

Chapitre II-4

Résultats expérimentaux autour de l'algorithme LLL.

Ce chapitre est un complément expérimental au chapitre II-3. Nous nous autorisons ainsi l'utilisation des notations et des algorithmes introduits alors.

V. Shoup — *I think it is safe to say that nobody really understands how the LLL algorithm works. The theoretical analyses are a long way from describing what "really" happens in practice. Choosing the best variant for a certain application ultimately is a matter of trial and error.*

Sommaire

4.1 Implantation de l'algorithme LLL flottant. 131
4.1.1 La variante prouvée. 132
4.1.2 Les variantes heuristique et rapide. 134
4.1.3 Les bases aléatoires étudiées. 135
4.2 Étude expérimentale de la qualité de la sortie. 136
4.2.1 Étude du défaut d'orthogonalité. 137
4.2.2 Étude des bases locales. 138
4.2.3 L'insertion profonde de Schnorr-Euchner. 141
4.3 Remarques sur le temps d'exécution de l'algorithme L^2. . 142
4.3.1 Mesure du temps d'exécution de l'algorithme L^2. 142
4.3.2 Cas des réseaux sac-à-dos. 145
4.3.3 Évolution du temps d'exécution avec les facteurs (δ, η). . . 147
4.4 Étude expérimentale de la précision numérique nécessaire. 147
4.4.1 Comportement le pire de l'algorithme LLL flottant. 148
4.4.2 Comportement « moyen » de l'algorithme LLL flottant. . . 148
4.4.3 À propos de l'orthogonalisation de Givens et Householder. 150
4.5 Problèmes ouverts. 152

Le but de ce chapitre est en quelque sorte de modérer la constatation de Victor Shoup ci-dessus, qui apparaît dans la documentation de NTL [162], ou, plus précisément, de faire état d'un certain nombre de remarques expérimentales sur l'algorithme LLL en arithmétique flottante. Ces remarques expérimentales viennent sans preuve mais aussi souvent que possible, nous donnons des « justifications » aux phénomènes observés. Rendre ces justifications rigoureuses est selon toute vraisemblance un travail très difficile, car il faut alors définir exactement comment sont créés les réseaux « aléatoires » que nous réduisons, et travailler sur la modification des distributions aléatoires initiales par l'algorithme LLL. Certains travaux vont dans ce sens (voir par exemple la thèse de doctorat d'Ali Akhavi [13]), mais les résultats prouvés demeurent encore très loin de donner une description fine du comportement de l'algorithme.

Pour réaliser ce travail descriptif, nous avons implanté des variantes de l'algorithme L^2 décrit au chapitre II-3. Ce code est disponible en ligne sous licence GNU GPL (*GNU General Public License*). Il est compétitif vis-à-vis des principaux logiciels qui disposent d'algorithmes de réduction de réseaux efficaces : NTL [162], Magma [119], Pari [20] et LiDIA [1]. Il est délicat de comparer précisément différents codes de réduction de réseaux : en effet, le nombre important de paramètres (les paramètres de réduction δ et η, la précision des calculs flottants), la diversité des entrées possibles et les différentes stratégies pour « sauver » les calculs lorsque des comportements suspects apparaissent, rendent certains codes plus efficaces dans certains contextes et d'autres pour d'autres situations. Notre code est, pour les cas que nos avons testés, au moins aussi rapide que les autres et assez souvent plus rapide. Le plus grand gain que nous avons observé concerne des matrices dont les entrées sont tirées uniformément et indépendamment (il s'agit d'un cas très particulier car la base donnée en entrée est en général très proche d'être réduite) : en dimension 750 pour des entrées de 1000 bits, notre code finit environ 15 fois plus rapidement que NTL[18].

Le comportement pratique de l'algorithme LLL apparaît parfois mystérieux, du moins quand il est mis en parallèle avec les analyses théoriques de l'algorithme. Les sources d'étonnement sont de trois types : elles sont liées au temps d'exécution de l'algorithme, à la qualité de sa sortie, et à la précision requise pour avoir un comportement correct — lorsque l'on utilise une arithmétique flottante pour le procédé d'orthogonalisation sous-jacent. On lit assez souvent que l'algorithme LLL se comporterait étonnamment mieux en pratique que ce que ses analyses théoriques prédisent : il finirait plus vite que le garantit la borne dans le cas le pire, il renverrait des bases d'une excellente qualité, et moins de problèmes liés à des pertes de précision apparaîtraient, par rapport aux bornes théoriques sur ces sujets. Nous étudions chacun de ces points en détail dans le présent chapitre. En particulier, nous donnons des justifications heuristiques des constatations expérimentales suivantes :

– la qualité de la base renvoyée est exponentiellement mauvaise, c'est-à-dire correspond à ce qui est annoncé par l'analyse dans le cas le pire, à une constante multi-

[18]Nous comparons la variante rapide de `fplll-1.2` avec $\delta = 0.99, \eta = 0.51$, avec la routine FP_LLL, pour $\delta = 0.99$.

plicative près (dans l'exposant) ;
- à une constante multiplicative près, la précision requise par les calculs est celle que nous avons obtenue pour le cas le pire, au chapitre II-3 ;
- dans le cas des bases « sac-à-dos » (nous expliquons ensuite ce que nous appelons une base « sac-à-dos »), le temps d'exécution de l'algorithme L^2 est meilleur que celui de la borne de complexité dans le cas le pire, mais nous pouvons utiliser ce fait pour construire (heuristiquement) des bases atteignant la borne dans le cas le pire.

Ainsi, pour résumer, nous sommes amenés à penser que l'analyse dans le cas le pire de l'algorithme L^2 donne les bons ordres de grandeur asymptotiques du comportement pratique de l'algorithme, mais que les « constantes » observées sont sensiblement meilleures que celles qui sont garanties dans le cas le pire. Le comportement pratique est meilleur que ce qui est garanti par la théorie, mais il n'y a pas de différentiel surprenant.

Nous essayons d'exploiter certaines constatations expérimentales pour essayer d'améliorer l'algorithme d'un point de vue pratique, pour déterminer les choix les plus « judicieux » de paramètres, et pour commenter certaines de ses variantes. Ainsi, nous discuterons quelques heuristiques proposées par [19] — qui est certainement la référence la plus complète sur le comportement pratique de l'algorithme LLL, ainsi que l'utilisation de procédés d'orthogonalisation alternatifs comme ceux de Givens et Householder, décrits dans [95, 153] et implantés dans NTL, ou encore la stratégie d'insertion profonde de Schnorr et Euchner [154].

Le déroulement du chapitre est le suivant. Dans un premier temps, nous décrivons les diverses variantes de notre implantation de l'algorithme L^2 décrit au chapitre II-3, ainsi que les familles de bases auxquelles nous nous intéresserons. Après avoir ainsi fixé ce contexte expérimental, nous décrirons et tenterons d'expliquer les phénomènes que ce code nous a permis d'observer : à la section 4.2, nous analyserons la qualité des bases renvoyées par l'algorithme ; à la section 4.3, nous ferons quelques remarques sur le temps d'exécution de l'algorithme ; enfin, à la section 4.4, nous ferons état de nos observations sur les pertes de précision de l'algorithme. À la section 4.5, nous donnons quelques problèmes ouverts directement liés aux observations expérimentales.

4.1 Implantation de l'algorithme LLL flottant.

L'algorithme LLL classique utilise une arithmétique rationnelle pour les calculs liés à l'orthogonalisation de Gram-Schmidt. Il a été observé expérimentalement dans les années 1980 [34, 101] que le fait de remplacer cette arithmétique rationnelle par une arithmétique flottante permettait d'accélérer les calculs de manière considérable, sans pour autant affaiblir la qualité des bases renvoyées. L'algorithme que nous avons décrit au chapitre II-3 est très proche de ceux utilisés alors, et son analyse rigoureuse permet d'expliquer certaines observations expérimentales de [34, 101]. Nous nous intéresserons dans ce chapitre uniquement aux variantes flottantes de l'algorithme LLL classique, celles-ci lui étant strictement supérieures (quand elles sont correctes) puisqu'elles renvoient des bases de qualité équivalente, plus rapidement. Pour se convaincre, le lecteur pourra comparer les implantations des algorithmes LLL flottants (le LLL_FP de NTL, le LLL par défaut

de Magma, ou encore notre code, `fplll-1.2`) avec les implantations de l'algorithme LLL classique (celle de NTL par exemple, ou encore celle disponible sur la page internet de Paul Zimmermann[19], qui est une traduction en GNU MP de l'implantation correspondante dans NTL).

Le code `fplll-1.2` contient trois implantations complémentaires de l'algorithme LLL en arithmétique flottante, avec différents compromis entre efficacité et correction : une variante prouvée, une variante heuristique plus rapide mais qui pourrait éventuellement échouer avec une précision de calculs faible, et une variante rapide qui échoue sur certaines entrées mêmes petites en petite précision, mais qui pour de nombreuses entrées est nettement plus efficace que la variante heuristique. Les trois variantes utilisent la bibliothèque GNU MP [73] pour l'arithmétique entière, et la bibliothèque MPFR [147] est indispensable pour une variante prouvée valide en toute dimension.

4.1.1 La variante prouvée.

La variante prouvée est très similaire à l'algorithme L^2 que nous avons décrit au chapitre II-3. Nous en donnons un pseudo-code à la figure 4.1. Il y a deux principales différences avec l'algorithme du chapitre précédent. La première est que nous calculons la matrice de Gram de manière paresseuse : un produit scalaire $\langle \mathbf{b}_i, \mathbf{b}_j \rangle$ ne sera connu qu'à partir du moment où l'indice κ aura dépassé au moins une fois la quantité $\max(i, j)$ (avant ce moment-là, le produit scalaire $\langle \mathbf{b}_i, \mathbf{b}_j \rangle$ n'est jamais utilisé). La seconde est que nous maintenons tout au long de l'algorithme un tableau $(\alpha_i)_i$. La valeur α_i correspond au α courant pour l'indice i, c'est-à-dire la valeur minimale de l'indice κ depuis que celui-ci a valu au moins i, si cette valeur existe, et 1 sinon. Ce tableau permet de conserver (et donc de ne pas recalculer) les $\mu_{i,j}$ pour des petites valeurs de j : si $\alpha_i > j$, alors $\mu_{i,j}$ n'a pas changé depuis la dernière fois que l'indice κ a valu i (voir le chapitre II-3). Ces deux modifications ne changent en rien l'analyse de complexité de l'algorithme, elles permettent tout au plus de diminuer la « constante du $O(\cdot)$ ».

Dans cette variante prouvée, on peut choisir au moment de la compilation trois sortes d'arithmétique flottante sous-jacente :

- des flottants double précision (53 bits de précision) : dans ce cas, l'algorithme n'est correct que jusqu'à une certaine dimension, et uniquement si les entrées ne sont pas trop grosses (vers 500 bits apparaissent des problèmes de dépassement de capacité) ;
- des `dpe`[20] (des flottants double précision avec un entier pour chacun pour pouvoir étendre l'ensemble des exposants possibles) : les problèmes de dépassement de capacité ne devraient plus apparaître, mais les risques d'échec dûs à la faible précision restent les mêmes qu'avec les flottants double précision ;
- des nombres flottants en multiprécision de la bibliothèque MPFR [147] : dans ce cas, la précision est choisie par défaut pour que la correction de l'algorithme soit assurée, grâce aux bornes du chapitre II-3.

Bien entendu, les flottants double précision sont moins coûteux que les `dpe`, qui sont

[19] voir l'URL `http://www.loria.fr/~zimmerma/free`
[20] voir l'URL `http://www.loria.fr/~zimmerma/free`

Entrée : Une paire (δ, η) de facteurs satisfaisant $\delta \in]1/4, 1[$ et $\eta \in]1/2, \sqrt{\delta}[$, une taille de mantisse ℓ et des vecteurs non nuls $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ à coefficients entiers.
Sortie : Une base LLL-réduite de $L(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ pour la paire de facteurs (δ, η).
Variables : Une matrice $d \times d$ entière G contenant les produits scalaires, deux matrices $d \times d$ flottantes $(\bar{r}_{i,j})$ et $(\bar{\mu}_{i,j})$, un vecteur $d \times 1$ flottant $\bar{s}$ et un vecteur $d \times 1$ entier α.

1. Calculer exactement $\|\mathbf{b}_1\|^2$, $\|\mathbf{b}_2\|^2$ et $\langle \mathbf{b}_1, \mathbf{b}_2 \rangle$.
2. $\bar{\delta}$:=$\frac{\delta+1}{2}$, $\bar{\eta}$:=$\frac{\eta+1/2}{2}$, κ:=2, ζ:=0, $\bar{r}_{1,1}$:=$\diamond(\langle \mathbf{b}_1, \mathbf{b}_1 \rangle)$, $\kappa_{\max}$:=2, α:=$[1, \ldots, 1]$.
3. Tant que $\kappa \leqslant d$, faire :
4. Si $\kappa > \kappa_{\max}$, $\kappa_{\max}$:=κ, calculer exactement $\langle \mathbf{b}_\kappa, \mathbf{b}_1 \rangle, \ldots, \langle \mathbf{b}_\kappa, \mathbf{b}_{\kappa-1} \rangle$, $\|\mathbf{b}_\kappa\|^2$.
5. *test*:=0. Tant que *test* = 0, faire :
6. *test*:=1. Pour i de α_κ à $\kappa - 1$, faire :
7. $\bar{r}_{\kappa,i}$:=$\diamond(\langle \mathbf{b}_\kappa, \mathbf{b}_i \rangle)$,
8. Pour j de 1 à $i - 1$, faire : $\bar{r}_{\kappa,i}$:=$\diamond(\bar{r}_{\kappa,i} - \diamond(\bar{\mu}_{i,j} \cdot \bar{r}_{\kappa,j}))$,
9. Si $i \leqslant \zeta$, $\bar{\mu}_{\kappa,i}$:=0, sinon $\bar{\mu}_{\kappa,i}$:=$\diamond(\bar{r}_{\kappa,i} / \bar{r}_{i,i})$.
10. Pour i de $d - 1$ à 1, faire :
11. Si $|\bar{\mu}_{\kappa,i}| \geqslant \bar{\eta}$, alors x_i:=$\lfloor \bar{\mu}_{\kappa,i} \rceil$ et *test*:=0, sinon x_i:=0,
12. Pour j de 1 à $i - 1$, faire : $\bar{\mu}_{\kappa,j}$:=$\diamond(\bar{\mu}_{\kappa,j} - \diamond(x_i \cdot \bar{\mu}_{i,j}))$.
13. $\mathbf{b}_\kappa$:=$\mathbf{b}_\kappa - x_i \mathbf{b}_i$.
14. $\|\mathbf{b}_\kappa\|^2$:=$\|\mathbf{b}_\kappa\|^2 + x_i^2 \|\mathbf{b}_i\|^2 - 2x_i \langle \mathbf{b}_\kappa, \mathbf{b}_i \rangle$.
15. Pour j de 1 à $\kappa_{\max}$ sauf κ, faire : $\langle \mathbf{b}_i, \mathbf{b}_\kappa \rangle$:=$\langle \mathbf{b}_i, \mathbf{b}_\kappa \rangle - x_i \langle \mathbf{b}_i, \mathbf{b}_j \rangle$.
16. $\bar{s}_1$:=$\diamond(\|\mathbf{b}_\kappa\|^2)$. Pour j de 1 à κ, faire : $\bar{s}_j$:=$\diamond(\bar{s}_{j-1} - \diamond(\bar{\mu}_{\kappa,j-1} \cdot \bar{r}_{\kappa,i-1}))$.
17. κ':=κ. Tant que $\kappa \geqslant \zeta + 2$ et $\diamond(\bar{\delta} \cdot \bar{r}_{\kappa-1,\kappa-1}) \geqslant \bar{s}_{\kappa-1}$, faire : κ:=$\kappa - 1$.
18. Pour i de 1 à $\kappa - 1$, faire : $\bar{\mu}_{\kappa,i}$:=$\bar{\mu}_{\kappa',i}$, $\bar{r}_{\kappa,i}$:=$\bar{r}_{\kappa',i}$, $\bar{r}_{\kappa,\kappa}$:=$\bar{s}_\kappa$.
19. Insérer $\mathbf{b}_{\kappa'}$ entre les vecteurs $\mathbf{b}_{\kappa-1}$ et $\mathbf{b}_\kappa$, mettre à jour les produits scalaires.
20. α_κ:=κ. Pour i de $\kappa + 1$ à $\kappa_{\max}$, faire : si $\kappa < \alpha_i$, α_i:=κ.
21. Si $\mathbf{b}_{\kappa'} = \mathbf{0}$, ζ:=$\zeta + 1$.
22. Sinon, κ:=$\max(\zeta + 2, \kappa + 1)$.
23. Renvoyer $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$.

FIG. 4.1 – Pseudo-code de la variante prouvée de `fplll-1.2`.

moins coûteux que les nombres flottants de MPFR. Pour donner un ordre de grandeur, pour les opérations arithmétiques élémentaires avec 53 bits de précisions, les `dpe` sont de l'ordre de quatre fois plus lents que les flottants double précision, et les nombres flottants de MPFR avec 53 bits de précision sont de l'ordre de quinze à vingt fois plus lents que les flottants double précision. Ces données sont à manipuler avec précaution car elles varient d'une machine à l'autre, et surtout, l'arithmétique flottante ne constitue pas la composante dominante du coût de l'algorithme L^2 si les entrées des vecteurs initiaux sont grandes : la composante dominante provient des opérations arithmétiques entières sur les entrées de la matrice représentant la base, et sur celles de la matrice de Gram.

Nous avons aussi créé une adaptation du code pour des formes quadratiques quelconques (non nécessairement positives) : il suffit d'ajouter des valeurs absolues dans les tests de conditions de Lovász, à l'étape 17 de l'algorithme de la figure 4.1, comme cela est suggéré dans [165].

4.1.2 Les variantes heuristique et rapide.

La principale modification entre la variante prouvée et la variante heuristique est de ne pas calculer la matrice de Gram, mais d'évaluer en arithmétique flottante les produits scalaires aux moments où l'on en a besoin. Cela revient donc exactement à partir de l'algorithme LLL classique, et à remplacer l'arithmétique rationnelle par une arithmétique flottante, comme décrit par Schnorr et Euchner [154], et comme cela est implanté dans NTL. De même que pour la variante prouvée, on peut choisir trois types d'arithmétique flottante au moment de la compilation : les flottants double précision, les `dpe` et les nombres flottants de MPFR. La différence est que même avec les flottants de MPFR avec une précision arbitraire, ni la sortie ni le temps d'exécution ne sont garantis. Cela provient du fait qu'il peut y avoir de grosses annulations lors des calculs de produits scalaires. Pour diminuer les effets de ces annulations, une option de compilation peut être activée : elle permet de détecter ces grandes annulations, et le cas échéant de calculer exactement les produits scalaires impliqués. Évidemment, le temps d'exécution peut être fortement allongé. Cette astuce était déjà décrite dans [154], et implantée dans NTL.

Avec des nombres flottants double précision, la variante heuristique correspond à peu près au LLL_FP de NTL et au LLL par défaut de Magma, avec des `dpe` au LLL_XD de NTL, et avec les nombres flottants de MPFR au LLL_RR de NTL. Expérimentalement, nous n'avons jamais rencontré d'échec avec les flottants de MPFR, ni avec le LLL_RR de NTL d'ailleurs.

La variante rapide ne fonctionne qu'avec 53 bits de précision, et a à peu près les mêmes propriétés de correction que la variante heuristique avec des `dpe`, sans l'option de compilation qui détecte les annulations dans les calculs de produits scalaires. Au lieu d'utiliser un entier avec chaque flottant double précision (pour stocker l'exposant), on n'utilise qu'un entier par vecteur, pour stocker l'exposant qui correspond à la plus grande composante du vecteur. L'arithmétique flottante sous-jacente devient quasiment aussi efficace que les flottants double précision, sans qu'il y ait des problèmes de dépassement de capacité.

4.1.3 Les bases aléatoires étudiées.

Lorsqu'il s'agit d'étudier des phénomènes probabilistes avec les réseaux euclidiens, il peut y avoir confusion entre :

- choisir un réseau aléatoire, par exemple uniformément pour l'extension naturelle de la mesure de Lebesgue (voir [8]) ; ou encore uniformément parmi les réseaux entiers d'un déterminant donné, ce qui revient au même quand on fait tendre le déterminant vers $+\infty$ (voir [70]) ;
- choisir des bases aléatoires, sans s'intéresser à la distribution induite sur les réseaux qu'elles engendrent.

Du point de vue des algorithmes de réduction, il semble plus pertinent (si l'on considère les applications les plus classiques) de se placer dans la deuxième optique : pour tester l'efficacité de ces algorithmes, on leur donne en entrée des bases aléatoires très peu réduites. Dans [13, 15, 49], Daudé et Vallée, puis Akhavi choisissent des points uniformément tirés sur la sphère unité de dimension d. Ces points forment une base avec une grande probabilité, mais cette base est très proche d'être réduite : quand on la lui donne en entrée, l'algorithme LLL finit très rapidement car il ne fait quasiment rien. Dans nos expériences, nous prenons des bases dont le défaut d'orthogonalité est très grand. Ainsi, nos expériences sont peu adaptées aux bases provenant du cryptosystème NTRU [80], car celles-ci ont des défauts d'orthogonalité déjà faibles. Il faudrait dans ce cas multiplier la base déjà très orthogonale par une matrice unimodulaire aléatoire avec de grands coefficients.

Ici, nous nous intéressons à deux classes de bases aléatoires. Nous nommons la première « bases sac-à-dos » et la deuxième « bases d'Ajtai ». Les bases de la première classe sont utiles pour casser les cryptosystèmes de type « sac-à-dos », et pour chercher des polynômes minimaux. Elles apparaissent aussi lorsque l'on cherche le polynôme minimal d'un nombre algébrique, et quand on essaie de découvrir des relations linéaires entières entre des nombres réels.

Définition 27 (Bases sac-à-dos). *Soient d et B des entiers. Une base sac-à-dos aléatoire est une base de d vecteurs lignes de dimension $d+1$, de la forme :*

$$\begin{pmatrix} x_1 & 1 & 0 & \dots & 0 & 0 \\ x_2 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{d-1} & 0 & 0 & \dots & 1 & 0 \\ x_d & 0 & 0 & \dots & 0 & 1 \end{pmatrix},$$

où les x_i sont des entiers choisis uniformément et indépendamment dans $[|-B, B|]$.

Une variante proche de « nos » bases d'Ajtai ont permis à Ajtai [9] de montrer que les ordres de grandeur des bornes de Schnorr [151] sur la qualité de la réduction BKZ (réduction Korkine-Zolotarev par blocs) sont fins dans le cas le pire.

Définition 28 (Bases d'Ajtai). *Soient d un entier et $\alpha > 0$ un réel. Une base d'Ajtai*

aléatoire est une base de d vecteurs lignes de dimension d, de la forme :

$$\begin{pmatrix} \lfloor 2^{((2d)^\alpha)} \rceil & 0 & \dots & 0 & 0 \\ x_{2,1} & \lfloor 2^{((2d-1)^\alpha)} \rceil & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{d-1,1} & x_{d-1,2} & \dots & \lfloor 2^{((d+2)^\alpha)} \rceil & 0 \\ x_{d,1} & x_{d,2} & \dots & x_{d,d-1} & \lfloor 2^{((d+1)^\alpha)} \rceil \end{pmatrix},$$

où pour tous $i > j$, l'entier $x_{i,j}$ est choisi uniformément et indépendamment dans l'intervalle $\left[-2^{((2d-j+1)^\alpha)}, 2^{((2d-j+1)^\alpha)}\right]$.

Pour une base d'Ajtai avec le coefficient $\alpha \geqslant 1$, on a pour tout $j \in [|1, d-1|]$:

$$\frac{\|\mathbf{b}_j^*\|}{\|\mathbf{b}_{j+1}^*\|} \approx 2^{(2d-j+1)^\alpha - (2d-j)^\alpha} \approx 2^{\alpha(2d-j)^{\alpha-1}}.$$

Les deux familles de bases que nous avons introduites mettent en valeur un certain nombre de propriétés intéressantes de l'algorithme LLL. Nous pensons en particulier qu'elles sont représentatives du comportement de l'algorithme LLL, vis-à-vis de la qualité de la base renvoyée, et des erreurs dûes aux pertes de précision numérique : on observe expérimentalement que si l'on part d'un réseau donné et qu'on en choisit une base avec un très fort défaut d'orthogonalité, alors le comportement de l'algorithme LLL (qualité de la sortie, pertes de précision, temps d'exécution dans une certaine mesure[21], ...) sera toujours le même. Cela rejoint de manière très informelle l'intuition d'Ajtai comme quoi tous les réseaux se comportent de la même manière [9] vis-à-vis des algorithmes de réduction par blocs de Schnorr [150], dont l'algorithme LLL est un cas particulier :

> *We will present a heuristic argument* [...] *which shows that if we start with a sufficiently random lattice and a random basis in it (random in some undefined heuristic sense) then Schnorr's algorithm will converge to a basis similar to the lattice in our lower bound.*

En particulier, nous pensons que si l'on part d'une base très réduite (par exemple la clé privée du cryptosystème GGH [69], ou la clé publique du cryptosystème NTRU [80]), et qu'on la multiplie par une grande matrice unimodulaire aléatoire, alors le comportement de l'algorithme LLL sur la base obtenue sera « standard ».

4.2 Étude expérimentale de la qualité de la sortie.

Avec le facteur $\delta \in]1/2, 1[$, les algorithmes LLL et L^2 (en fixant η très proche de $1/2$) renvoient des bases LLL-réduites $\mathbf{b}_1, \dots \mathbf{b}_d$ pour le facteur δ, qui satisfont donc les relations suivantes (voir le chapitre I-1) :

- $\|\mathbf{b}_1\| \leqslant \left(\frac{1}{\delta - 1/4}\right)^{\frac{d}{4}} (\text{vol } L)^{\frac{1}{d}}$.

[21] si la quantité $\prod_{i=1}^{d} \|\mathbf{b}_i^*\|^{d-i}$ est de l'ordre de $B^{(d^2)}$, ce qui est le cas des bases d'Ajtai mais pas des bases sac-à-dos.

– Pour tous $1 \leqslant i < d$, $\|\mathbf{b}_{i+1}^*\| \geqslant \left(\frac{1}{\delta - 1/4}\right) \|\mathbf{b}_i^*\|$.

On lit parfois dans la littérature des années 1980 que ces bornes dans le cas le pire ne seraient pas fines en pratique. On peut par exemple citer les propos d'Odlyzko dans [142] :

> *This algorithm* [...] *usually finds a reduced basis in which the first vector is much shorter than guaranteed* [theoretically]. *(In low dimensions, it has been observed empirically that it usually finds the shortest non-zero vector in a lattice.)*

Nous montrons dans ce chapitre que ces bornes théoriques semblent au contraire qualitativement assez représentatives de la réalité.

4.2.1 Étude du défaut d'orthogonalité.

À la figure 4.2, nous étudions l'évolution de la quantité $\frac{1}{d^2}\lg\frac{\|\mathbf{b}_1\|^d}{\det L}$ en fonction de la dimension (cette quantité vaut $-\frac{1}{4}\lg\left(\delta - \frac{1}{4}\right)$ dans le cas le pire). Pour la figure de gauche, chaque point correspond à cette quantité, après que l'algorithme L^2 a réduit une base sac-à-dos pour laquelle on a choisi $B = 2^{100 \cdot d}$. Chaque réduction a été effectuée avec la variante rapide décrite à la section 4.1, pour des facteurs de réduction $(\delta, \eta) = (0.999, 0.501)$. La figure de droite correspond à la même expérience pour des bases d'Ajtai avec $\alpha = 1.2$.

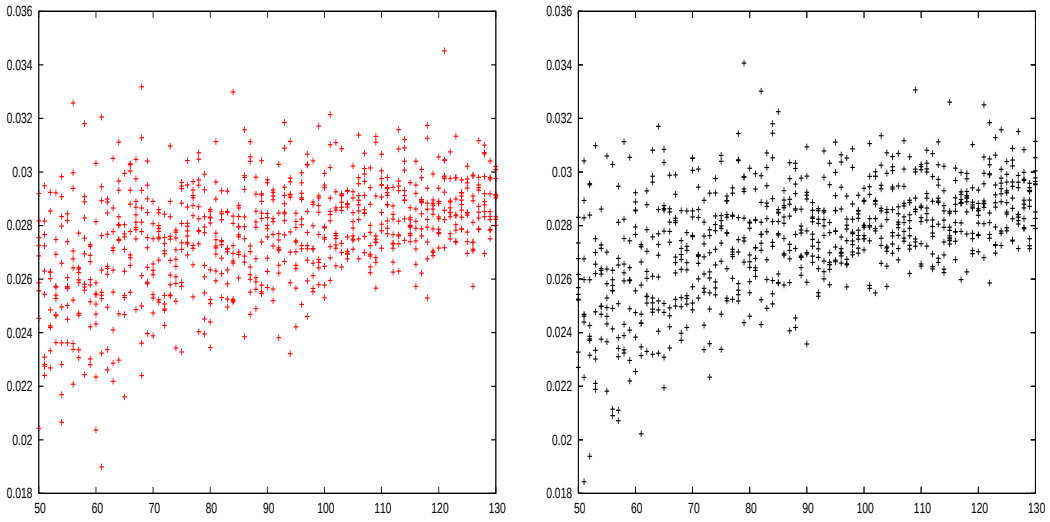


FIG. 4.2 – Évolution de la quantité $\frac{1}{d^2}\lg\frac{\|\mathbf{b}_1\|^d}{\det L}$ en fonction de la dimension, en partant de bases sac-à-dos (à gauche), et en partant de bases d'Ajtai (à droite).

On observe que les deux courbes sont similaires et tendent vers une constante, légèrement inférieure à 0.03 (pour ≈ 0.10 dans le cas le pire). Cela signifie que les premiers vecteurs des bases produites satisfont $\|\mathbf{b}_1\| \approx (1.02)^d (\mathrm{vol}\ L)^{\frac{1}{d}}$. Remarquons que pour des dimensions modérées, la constante multiplicative est très faible (par exemple, on a $(1.02)^{50} \approx 2.7$ et $(1.02)^{100} \approx 7.2$), ce qui explique peut-être pourquoi dans les années 1980 on croyait que l'algorithme LLL renvoyait des vecteurs plus courts que ne le garantissait

l'analyse théorique : on était alors restreint à des réseaux de dimensions modérées, pour des raisons pratiques évidentes.

À la figure 4.3, nous avons dressé l'évolution de la constante « 1.02 » en fonction du paramètre de réduction δ. Les expériences ont été réalisées avec des bases sac-à-dos de dimension 95, avec le facteur de réduction η fixé à 0.501. Chaque point correspond à la valeur moyenne de la quantité $\frac{1}{d^2}\lg\frac{\|\mathbf{b}_1\|^d}{\det L}$ pour 100 expériences.

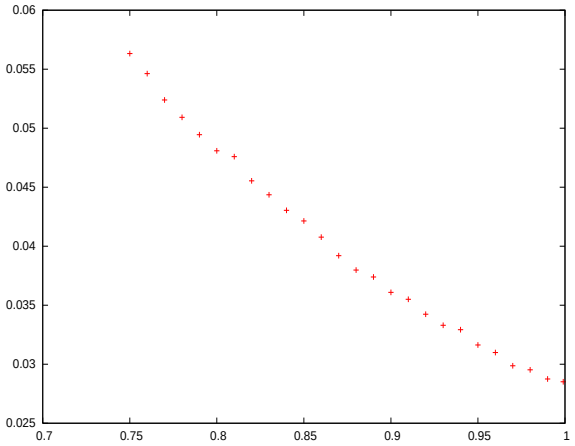


FIG. 4.3 – Évolution de la quantité $\frac{1}{d^2}\lg\frac{\|\mathbf{b}_1\|^d}{\det L}$ pour $d = 95$, en fonction du facteur δ.

4.2.2 Étude des bases locales.

Pour comprendre la forme de la base renvoyée par l'algorithme de réduction, il est intéressant de considérer les bases locales, c'est-à-dire les paires de vecteurs $(\mathbf{b}_i^*, \mathbf{b}_{i+1}^* + \mu_{i+1,i}\mathbf{b}_i^*)$, pour $i \in [|1, d-1|]$. Dans l'algorithme LLL, les opérations les plus importantes sont les échanges dûs aux échecs des conditions de Lovász, et ce phénomène a lieu au niveau de ces bases locales. Nous observons que les bases locales ont une distribution donnée après avoir réduit une base aléatoire, et que ces bases locales semblent se comporter indépendamment les unes des autres. Dans la figure 4.4, chaque point représente une base locale d'un réseau de dimension $d = 71$ réduit par la variante rapide décrite à la section 4.1 à partir d'une base sac-à-dos pour laquelle on a choisi $B = 2^{100 \cdot d}$. Les facteurs de réduction sont $(\delta, \eta) = (0.999, 0.501)$. Il y a 2100 points sur la figure, calculés à partir de 30 réductions de réseaux de dimension 71. Un point a pour coordonnées $\left(\mu_{i+1,i}, \frac{\|\mathbf{b}_{i+1}^*\|}{\|\mathbf{b}_i^*\|}\right)$.

La figure 4.4 peut paraître surprenante. Tout d'abord, les quantités $\mu_{i+1,i}$ ne sont pas du tout uniformément réparties dans $[-0.501, 0.501]$, comme on aurait pu le croire a priori (et comme cela semble être le cas pour les autres $\mu_{i,j}$) : elles sont concentrées vers les extrémités de l'intervalle. Dans la figure 4.5, nous avons dressé un histogramme de la répartition des valeurs des $\mu_{i+1,i}$. La valeur moyenne des $|\mu_{i+1,i}|$ semble être proche de 0.38. Par ailleurs, la valeur moyenne de la quantité $\frac{\|\mathbf{b}_{i+1}^*\|}{\|\mathbf{b}_i^*\|}$ semble proche de 1.04, ce qui est en accord avec la constante « 1.02 » du paragraphe précédent, et l'hypothèse d'indépendance.

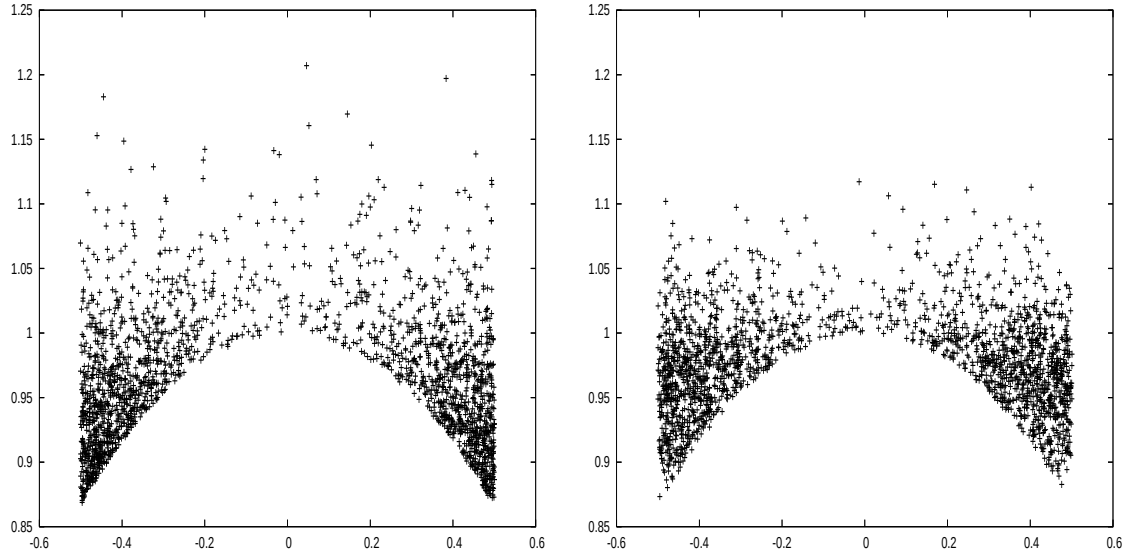


FIG. 4.4 – Forme des bases locales après LLL-réduction, à gauche, et après LLL-réduction avec insertion profonde (voir le paragraphe 4.2.3), à droite.

En effet :

$$\begin{aligned}
\frac{\|\mathbf{b}_1\|^d}{\text{vol } L} &= \frac{\|\mathbf{b}_1\|}{\|\mathbf{b}_1^*\|} \cdot \frac{\|\mathbf{b}_1\|}{\|\mathbf{b}_2^*\|} \cdot \ldots \cdot \frac{\|\mathbf{b}_1\|}{\|\mathbf{b}_d^*\|} \\
&= \prod_{i=1}^{d} \left(\frac{\|\mathbf{b}_i^*\|}{\|\mathbf{b}_{i+1}^*\|} \right)^{d-i+1} \\
&\approx (1.04)^{\frac{d^2}{2}} \\
&\approx (1.02)^{d^2}.
\end{aligned}$$

On peut expliquer heuristiquement la forme des bases locales $(\mathbf{b}_i^*, \mathbf{b}_{i+1}^* + \mu_{i+1,i}\mathbf{b}_i^*)$ de la manière suivante. Au cours de l'algorithme, les $\|\mathbf{b}_i^*\|$ grandissent petit à petit. Quand $\|\mathbf{b}_{i+1}^*\|$ devient plus grand que $\sqrt{4/3}$, soit le vecteur $\mathbf{b}_{i+1}^* + \mu_{i+1,i}\mathbf{b}_i^*$ est « dans l'un des coins » de la figure 4.4, soit « près de l'axe vertical ». S'il est « dans l'un des coins », il est conservé car la base locale $(\mathbf{b}_i^*, \mathbf{b}_{i+1}^* + \mu_{i+1,i}\mathbf{b}_i^*)$ est réduite. Sinon les vecteurs $\mathbf{b}_i$ et $\mathbf{b}_{i+1}$ sont échangés. On voit ainsi que les vecteurs « des coins » ont une probabilité plus grande d'être conservés à la fin de l'exécution de l'algorithme.

Ces expériences nous amènent à la conjecture suivante. Si l'on donne en entrée à l'algorithme LLL une base avec un défaut d'orthogonalité très élevé, alors la distribution de la base de sortie, après avoir divisé tous les vecteurs par la norme du premier, sera identique à la distribution induite par l'expérience suivante :

1. Les $\mu_{i,j}$ pour $i > j + 1$ sont tirés aléatoirement et uniformément dans $[-1/2, 1/2]$.
2. Les paires $\left(\mu_{i+1,i}, \frac{\|\mathbf{b}_{i+1}^*\|}{\|\mathbf{b}_i^*\|}\right)$ sont tirées aléatoirement selon la distribution qui correspond à la figure 4.5. En ordonnée se trouve le pourcentage correspondant aux $\mu_{i,j}$ dans l'intervalle correspondant.

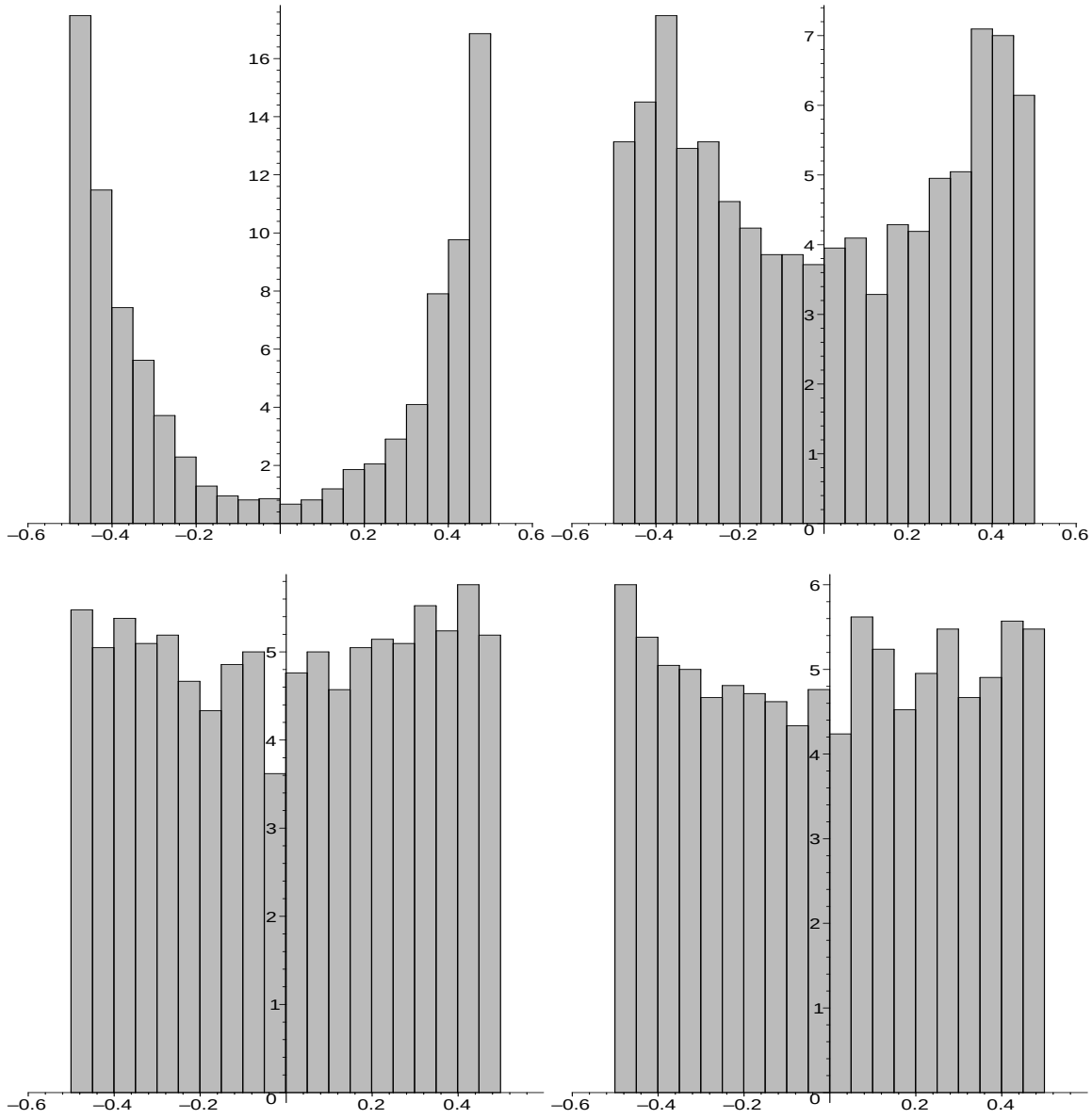


FIG. 4.5 – Répartition des $\mu_{i,i-1}$ (en haut à gauche), $\mu_{i,i-2}$ (en haut à droite), $\mu_{i,i-5}$ (en bas à gauche) et $\mu_{i,i-10}$ (en bas à droite), pour des réseaux de dimension 71, après LLL-réduction.

3. Tous ces tirages sont deux à deux indépendants.

Cette description n'est pas exacte (on voit par exemple à la figure 4.5 que les $\mu_{i,i-2}$ ne sont pas uniformément distribués), il ne s'agit que d'une première approximation. Elle est conforme à la discussion heuristique d'Ajtai [9] sur la géométrie d'une base après lui avoir appliqué l'algorithme de réduction BKZ de Schnorr [150]. Par ailleurs, notre hypothèse d'indépendance est en accord avec la *geometric series assumption* de Schnorr [152] : la quantité $\frac{\|\mathbf{b}_i^*\|}{\|\mathbf{b}_1\|}$ décroît alors géométriquement avec i.

4.2.3 L'insertion profonde de Schnorr-Euchner.

L'étude des bases locales permet d'expliquer pourquoi l'algorithme d'insertion profonde de Schnorr et Euchner [154] renvoie des bases d'une qualité supérieure à celle de l'algorithme LLL. Rappelons tout d'abord en quoi consiste cet algorithme. Celui-ci est donné à la figure 4.6 dans le cas où l'on utilise une arithmétique rationnelle. Il diffère de l'algorithme LLL dans sa stratégie pour tester les conditions de Lovász : plutôt que d'insérer le vecteur $\mathbf{b}_\kappa$ avant le vecteur $\mathbf{b}_{\kappa-1}$ si la condition n'est pas satisfaite, l'insertion profonde consiste à insérer le vecteur $\mathbf{b}_\kappa$ avant le vecteur $\mathbf{b}_i$ où i est l'indice minimal pour lequel, orthogonalement aux vecteurs $\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}$, le vecteur $\mathbf{b}_\kappa$ est plus court que le vecteur $\mathbf{b}_i$.

Entrée : Des vecteurs $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ linéairement indépendants et un facteur $\delta \in]1/4, 1[$.
Sortie : Une base LLL-réduite de $L(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ pour des facteurs $(\delta, 1/2)$.

1. κ:=2, $r_{1,1}$:=$\|\mathbf{b}_1\|^2$.
2. Tant que $\kappa \leqslant d$, faire
3. Calculer les $\mu_{\kappa,j}$ et $r_{\kappa,j}$ pour $j \leqslant \kappa$.
4. Proprifier $\mathbf{b}_\kappa$ vis-à-vis de $(\mathbf{b}_1, \ldots, \mathbf{b}_{\kappa-1})$.
5. κ':=1. Tant que $\kappa' \leqslant \kappa - 1$ et $\delta \cdot r_{\kappa',\kappa'} > \sum_{i=\kappa'}^{\kappa} \mu_{\kappa,i}^2 r_{i,i}$, faire : κ':=$\kappa' + 1$.
6. $r_{\kappa',\kappa'}$:=$\sum_{i=\kappa'}^{\kappa} \mu_{\kappa,i}^2 r_{i,i}$. Pour i de 1 à $\kappa' - 1$, faire : $\mu_{\kappa',i}$:=$\mu_{\kappa,i}$, $r_{\kappa',i}$:=$r_{\kappa,i}$.
7. Insérer $\mathbf{b}_\kappa$ entre $\mathbf{b}_{\kappa'-1}$ et $\mathbf{b}_{\kappa'}$.
8. κ:=max$(2, \kappa' + 1)$.
9. Renvoyer $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$.

FIG. 4.6 – L'algorithme d'insertion profonde.

On ne sait pas si l'algorithme d'insertion profonde a une complexité polynomiale[22]. Par contre, il renvoie des bases d'une qualité supérieure à celle

de l'algorithme LLL. Pour la partie droite de la figure 4.4, nous avons réalisé une expérience proche de celle de la partie gauche. Nous sommes partis de bases sac-à-dos, que nous avons LLL-réduites, avant de les donner en entrée à une variante flottante de l'algorithme de la figure 4.6. Nous LLL-réduisons avant d'appliquer l'algorithme d'insertion

[22]Plus précisément, on ne sait pas borner le nombre d'itérations de boucle polynomialement en la taille de l'entrée, l'argument sur la décroissance de la quantité $D = \Pi_{k \leqslant \dim L} d_k$ échouant (voir la preuve du théorème 17, page 104).

profonde car cela est beaucoup plus efficace (le nombre d'itérations de boucles et donc le temps d'exécution deviennent sinon trop élevés pour réaliser l'expérience). La partie droite de la figure 4.4 (après insertion profonde) est semblable à la partie gauche (sans insertion profonde), mais la moyenne des $\frac{\|\mathbf{b}_{i+1}^*\|}{\|\mathbf{b}_i^*\|}$ est plus élevée. Il semblerait ainsi que la constante « 1.04 » devienne ≈ 1.025. Ces valeurs sont cohérentes avec les observations de [19].

Ce phénomène peut être expliqué informellement de la manière suivante. Quand la base locale $(\mathbf{b}_i^*, \mathbf{b}_{i+1}^* + \mu_{i+1,i}\mathbf{b}_i^*)$ correspond à un « coin » de la partie gauche de la figure 4.4, la condition de Lovász entre les vecteurs $\mathbf{b}_i$ et $\mathbf{b}_{i+2}$ a plus de chances de ne pas être satisfaite. En effet, la composante sur $\mathbf{b}_{i+1}^*$ du vecteur $\mathbf{b}_{i+2}$ sera petite (car $\|\mathbf{b}_{i+1}^*\|$ est petit par rapport à $\|\mathbf{b}_i^*\|$), et donc la quantité $\mu_{i+2,i+1}\mathbf{b}_{i+1}^*$ sera plus petite qu'elle ne l'est sinon. Puisque la norme du projeté de $\mathbf{b}_{i+2}$ orthogonalement à $\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}$ est :

$$\sqrt{\mu_{i+2,i}^2\|\mathbf{b}_i^*\|^2 + \mu_{i+2,i+1}^2\|\mathbf{b}_{i+1}^*\|^2 + \|\mathbf{b}_{i+2}^*\|^2},$$

ce vecteur a plus de chances d'être plus court que $\|\mathbf{b}_i^*\|$. Quand cela arrive, il y a une insertion de profondeur au moins 2 du vecteur $\mathbf{b}_{i+2}$. Puisque les bases locales « dans les coins » apparaissent avec une forte probabilité, l'algorithme d'insertion profonde effectue assez souvent des insertions de profondeur supérieure à 2, qui n'avaient pas été réalisées par l'algorithme LLL.

4.3 Remarques sur le temps d'exécution de l'algorithme L^2.

Dans cette section, nous donnons une explication heuristique du temps d'exécution de l'algorithme LLL pour des bases sac-à-dos. Celle-ci nous permet de construire des bases pour lesquelles la borne de complexité $O(d^4(d+n)(d+\log B)\log B)$ de l'algorithme L^2 serait atteinte dans le cas moyen.

4.3.1 Mesure du temps d'exécution de l'algorithme L^2.

En pratique, pour des dimensions modérées (jusqu'aux dimensions $170-190$, comme cela est expliqué à la section 4.4), il n'est pas intéressant d'utiliser la variante prouvée plutôt que la variante heuristique avec des `dpe` ou la variante rapide. En effet, nous rencontrons avec cette dernière reste peu de problèmes liés à des pertes de précision, et l'arithmétique sur les flottants double précision est nettement plus efficace que l'arithmétique avec des nombres flottants de précision arbitraire. La difficulté avec la variante rapide est que la taille de mantisse est fixe, alors que cette taille de mantisse doit augmenter avec la dimension, non seulement pour garantir la correction de l'algorithme L^2, mais aussi pour garantir sa borne de complexité.

Quand la taille de mantisse est fixée, le nombre d'itérations de la proprification pa-

resseuse peut être multiplié[23] par d ; par ailleurs, puisque la taille de mantisse est fixe, les mises à jour des vecteurs de la base (quand on les multiplie par les entiers trouvés lors de la proprification paresseuse) devraient coûter $O(d \log B)$ à la place de $O(d^2 \log B)$. Ainsi, globalement on devrait conserver une borne de complexité du type $O(d^4(d+n)(d+\log B)\log B)$ tant que les problèmes liés à des pertes de précision ne deviennent pas trop importants. On peut résumer cette discussion en disant qu'il n'est pas simple d'estimer rigoureusement quel devrait être le coût pratique de l'algorithme pour une taille de mantisse qui n'augmente pas avec la dimension.

Pour « retrouver » expérimentalement la borne $O(d^4(d+n)(d+\log B)\log B)$, il faut utiliser la variante prouvée de l'implantation. Pour simplifier, on peut choisir $n = \Theta(d)$ et $\log B$ très grand par rapport à d, pour obtenir une borne de complexité $O(d^5 \log^2 B)$. Nous donnons une analyse heuristique permettant de retrouver cette borne :

1. Il y a $O(d^2 \log B)$ itérations de boucle.
2. Pour une itération de boucle donnée, il y a en général deux itérations de la proprification paresseuse (étapes 5 à 15 de l'algorithme de la figure 4.1, page 133) : la première rend les $|\mu_{\kappa,i}|$ inférieurs à η, et la deuxième recalcule les $\mu_{\kappa,i}$ et les $r_{\kappa,i}$ avec une erreur numérique plus faible. On peut justifier cette simplification en la mettant en parallèle avec le fait que les quotients apparaissant calculés par l'algorithme d'Euclide sont « en général » assez petits. Cette simplification est fausse dans le cas de bases sac-à-dos, comme nous le verrons au paragraphe suivant.
3. Chaque itérations de boucle de la proprification paresseuse fait intervenir $O(d^2)$ opérations arithmétiques.
4. Parmis ces opérations arithmétiques, les plus coûteuses sont celles faisant intervenir les mutiplications entre coefficients entiers de la base ou de la matrice de Gram (de taille $O(\log B)$) et les coefficients x_i calculés à l'étape 11 de l'algorithme de la figure 4.1, page 133 (les x_i sont de taille $O(d)$ si l'on suppose que $|\mu_{\kappa,i}| = O(1)$ pour tout i, avant l'étape 5).

Nous discutons désormais la finesse de chacune de ces bornes. En premier lieu, la borne $O(d^2 \log B)$ semble asymptotiquement fine, comme le suggère la figure 4.7. La partie gauche de la figure correspond à $\alpha = 1.2$: les points ont été obtenu expérimentalement, et la courbe est l'interpolation par une fonction du type $f(d) = a \cdot d^{3.2}$, calculée par `gnuplot`. La partie droite correspond à $\alpha = 1.5$: les points ont été obtenu expérimentalement, et la courbe est l'interpolation par une fonction du type $g(d) = b \cdot d^{3.5}$. Dans le cas de bases d'Ajtai, il y a extrêmement peu d'itérations de boucle pour lesquelles il y a plus de deux passes dans la boucle des étapes 5 à 15 : par exemple, pour $d \leqslant 75$ et $\alpha = 1.5$, cela concerne moins de 0.01% des itérations de boucle. La troisième borne est elle aussi atteinte en pratique. Ainsi, les trois premières bornes semblent atteintes en pratique.

Par contre, les x_i calculés semblent très fréquemment de longueur inférieure à un mot machine (64 bits sur un Opteron), aussi est-il difficile d'observer le facteur $O(d)$ dans la complexité totale, qui devrait provenir de leur taille. Par exemple, sur un réseau d'Ajtai

[23] La quantité M de la preuve du théorème 21, page 120, perd $O(1)$ bits à chaque itération de boucle de l'algorithme paresseux de proprification, au lieu de $O(d)$ bits.

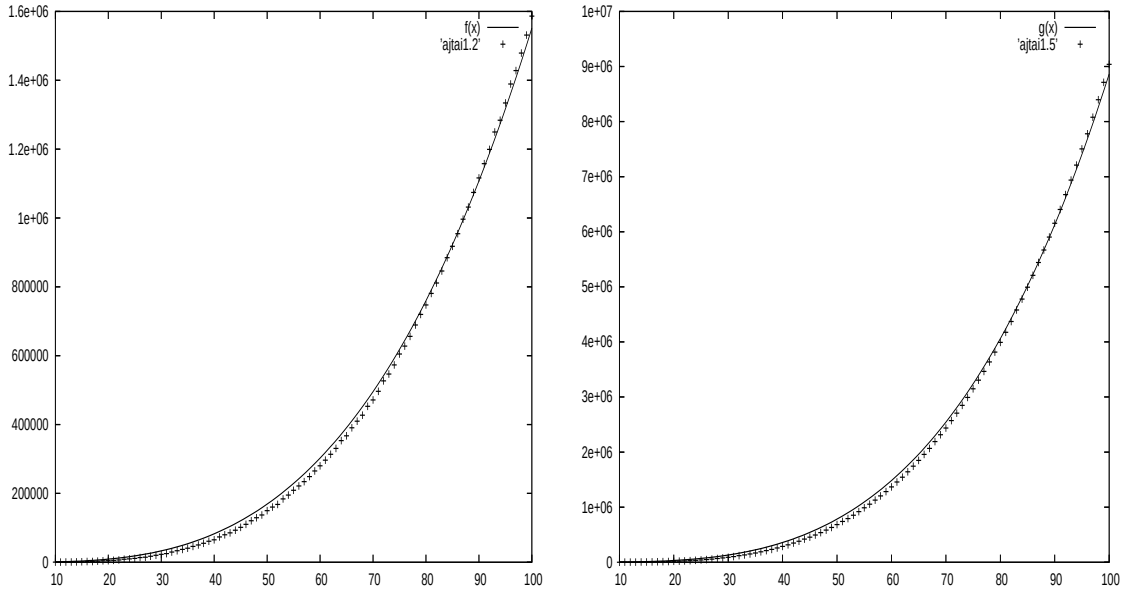


FIG. 4.7 – Croissance du nombre d'itérations de boucle dans l'exécution de l'algorithme LLL en fonction de la dimension, pour des réseaux aléatoires de type « Ajtai ».

pour $d \leqslant 75$ et $\alpha = 1.5$, moins de 0.2% des x_i non nuls calculés ont plus de 64 bits. Nous expliquons cela de la manière suivante. Les x_i calculés sont les entiers les plus proches des $\mu_{\kappa,i}$ calculés et mis-à-jour lors des étapes 5 à 15. Plus précisément :

$$\begin{aligned} \mu_{\kappa,i}^{(\text{final})} &= \mu_{\kappa,i}^{(\text{initial})} - \sum_{j=i+1}^{\kappa-1} x_j \mu_{j,i} \\ &= \mu_{\kappa,i}^{(\text{initial})} - \sum_{j=i+1}^{\kappa-1} \left\lfloor \mu_{\kappa,j}^{(\text{final})} \right\rceil \mu_{j,i}. \end{aligned}$$

Nous modélisons la suite des $\mu_{\kappa,\kappa-i}^{(\text{final})}$ pour $i \in [|1, \kappa|]$ par la suite de variables aléatoires U_i définies par les relations :

$$\begin{aligned} U_0 &= R_0, \\ U_i &= R_i + \sum_{j=1}^{i-1} U_j R'_{i,j}, \text{ pour } i \geqslant 1, \end{aligned}$$

où les R_i sont distribués uniformément dans $[-a, a]$ pour une certaine constante a, et les $R'_{i,j}$ sont distribués aléatoirement dans $[-\eta, \eta]$. On suppose aussi que toutes les variables aléatoires R_i et $R'_{i,j}$ sont indépendantes deux à deux. Il s'agit d'hypothèses extrêmement fortes sur la distribution des $\mu_{i,j}$. En particulier, nous avons vu à la section 4.2 que cette hypothèse n'est pas vérifiée en pratique pour les $\mu_{i,i-1}$. Cependant, cela ne change pas significativement le comportement asymptotique de la suite U_i, et simplifie les calculs.

Par ailleurs, il aurait fallu prendre $\lfloor U_j \rceil$ pour être plus proche de la réalité, mais puisque les U_j semblent croître asymptotiquement, cela simplifie l'analyse sans être déraisonnable d'un point de vue théorique. En utilisant l'hypothèse d'indépendance des R_i et $R'_{i,j}$, ainsi que leur symétrie par rapport à 0, nous avons pour $i \geqslant 1$:

$$\begin{aligned} \mathbb{E}[U_i] &= 0 \\ \mathbb{E}[U_i^2] &= \mathbb{E}[R_i^2] + \sum_{j=1}^{i-1} \mathbb{E}[U_j^2]\mathbb{E}[R_{i,j}'^2] = \frac{2a^3}{3} + \frac{2\eta^3}{3} \sum_{j=1}^{i-1} \mathbb{E}[U_j^2]. \end{aligned}$$

Ainsi, pour i tendant vers $+\infty$, on a $\mathbb{E}[U_i^2] \approx \left(\frac{2\eta^3}{3} + 1\right)^i$. Si on prend $\eta \approx 1/2$, on obtient $\sqrt{\frac{2\eta^3}{3} + 1} \approx \sqrt{\frac{13}{12}} \approx 1.04$. Ainsi, les x_i seraient bien asyptotiquement de longueur $O(d)$, mais la constante du $O(\cdot)$ serait très petite. En effet, la quantité $(1.04)^d$ dépasse 2^{64} (c'est-à-dire les x_i sont souvent plus longs qu'un mot machine) pour $d \geqslant 1100$. Il est pour le moment impossible de réduire des réseaux d'une telle dimension pour lesquels les autres bornes de l'analyse sont elles aussi atteintes.

Cette dernière étude explique que pour des dimensions modérées (entre 100 et 200 par exemple), il est difficile d'observer expérimentalement la borne $O(d^5 \log^2 B$. Il faudrait augmenter sensiblement les paramètres d et B, mais le temps d'exécution devient alors trop important.

4.3.2 Cas des réseaux sac-à-dos.

Les réseaux sac-à-dos se comportent nettement mieux que ce que garantit la borne de complexité théorique. Nous nous proposons ici d'en donner une explication. Tout d'abord, dans le cas des réseaux sac-à-dos, on peut montrer qu'il y a moins d'itérations de boucles que dans le cas le pire : la quantité $D = \prod_{i=1}^{d} \|\mathbf{b}_i^*\|^{d-i+1}$ de l'analyse de l'algorithme LLL (voir le chapitre II-3) satisfait $D = B^{O(d)}$, ce qui garantit que le nombre d'itérations de boucle est borné par $O(d \log B)$. Ainsi, le théorème 18 du chapitre II-3 (page 111) nous donne la borne de complexité suivante :

$$O(d^4(d + \log B) \log(dB)).$$

Nous supposerons dans la suite de ce paragraphe que $\log B = \Omega(d^2)$, ce qui nous permet de simplifier la borne ci-dessus en $O(d^4 \log^2 B)$.

Cette borne de complexité ne semble pas refléter la situation pratique, qui semble plutôt être de l'ordre de $O(d^3 \log^2 B)$, même si cela reste difficile à vérifier, comme nous l'avons expliqué au paragraphe précédent. Nous proposons l'explication suivante. Les $O(d \log B)$ itérations de boucle se partagent de manière « équilibrée » pour les différents indices $\kappa_{\max}$ possibles : on définit l'indice κ_{max} comme le maximum des indices κ depuis le début de l'exécution de l'algorithme. Au début, on a $\kappa_{\max} = 2$, puis $\kappa_{\max}$ est incrémenté jusqu'à $d + 1$, quand l'exécution de l'algorithme s'achève. Le nombre d'itérations de boucle pour chaque valeur de $\kappa_{\max}$ semble être le même, de l'ordre de $O(\log B)$. Nous subdivisons donc l'exécution de l'algorithme en $d - 1$ phases, suivant la valeur de $\kappa_{\max}$.

Il semble (expérimentalement) qu'à l'issue de la phase correspondant à une valeur donnée de l'indice $\kappa_{\max}$, la base ait la forme suivante :

$$\begin{pmatrix}
a_{1,1} & a_{1,2} & \dots & a_{1,\kappa_{\max}+1} & 0 & 0 & \dots & 0 \\
a_{2,1} & a_{2,2} & \dots & a_{2,\kappa_{\max}+1} & 0 & 0 & \dots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
a_{\kappa_{\max},1} & a_{\kappa_{\max},2} & \dots & a_{\kappa_{\max},\kappa_{\max}+1} & 0 & 0 & \dots & 0 \\
A_{\kappa_{\max}+1} & 0 & \dots & 0 & 1 & 0 & \dots & 0 \\
A_{\kappa_{\max}+2} & 0 & \dots & 0 & 0 & 1 & \dots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
A_d & 0 & \dots & 0 & 0 & 0 & \dots & 1
\end{pmatrix},$$

où les entiers $a_{i,j}$ satisfont, pour $i \leqslant \kappa_{\max}$ et $j \leqslant \kappa_{\max+1}$:

$$|a_{i,j}| = O\left(B^{\frac{1}{\kappa_{\max}}}\right).$$

En fait, les $\kappa_{\max}$ premiers vecteurs sont LLL-réduits, ce qui nous garantit que $\prod_{i=1}^{\kappa_{\max}} \|\mathbf{b}_i\| \leqslant 2^{O(d^2)} B$. Il semble expérimentalement que la borne ci-dessus soit fine et que les vecteurs $\mathbf{b}_i$ (pour $i \leqslant \kappa_{\max}$) aient des longueurs similaires.

Nous subdivisons chaque phase en deux sous-phases : dans la première sous-phase, le vecteur $\mathbf{b}_{\kappa_{\max}}$ est proprifié vis-à-vis des vecteurs $\mathbf{b}_1, \dots, \mathbf{b}_{\kappa_{\max}-1}$; la seconde sous-phase est constituée de toutes les itérations de boucle qui suivent, jusqu'à l'incrémentation de l'indice $\kappa_{\max}$. La première sous-phase « raccourcit » le vecteur $\mathbf{b}_{\kappa_{\max}}$: sa longueur était $\approx B$, elle devient inférieure à $\sqrt{\kappa_{\max}}(\max_{i<\kappa_{\max}} \|\mathbf{b}_i\|)+1 \lesssim B^{\frac{1}{\kappa_{\max}-1}}$ (en utilisant l'hypothèse donnée plus haut). Le théorème 21 du chapitre II-3 (page 120) permet de borner le coût de la première sous-phase par $O(d \log^2 B)$: il y a $O\left(\frac{\log B}{d}\right)$ itérations de boucle de l'algorithme paresseux de proprification ; chacune fait intervenir $O(d^2)$ opérations arithmétiques ; les plus coûteuses parmis ces dernières sont des multiplications entre des x_i (de longueurs $O(d)$ bits) avec des entiers correspondants à des coefficients des matrices de la base et de Gram ; ces multiplications sont liées à des coordonnées de vecteurs $\mathbf{b}_i$ pour $i < \kappa$ (et sont donc de longueur $O\left(\frac{\log B}{d}\right)$), à l'exception des multiplications $x_i \cdot \langle \mathbf{b}_\kappa, \mathbf{b}_i \rangle$, qui sont en proportion $O\left(\frac{1}{d}\right)$ des autres et coûtent chacune $O(d \log B)$ opérations élémentaires. Le coût total des premières sous-phases est donc borné par $O(d^2 \log^2 B)$. Nous verrons que cette quantité est asymptotiquement négligeable par rapport au coût total de l'algorithme.

Nous essayons désormais de borner le coût total des deuxièmes sous-phases. Soit $\mathbf{b}'_i$ le vecteur $\mathbf{b}_i$ après la première sous-phase correspondant à la phase $\kappa_{\max} = i$, c'est-à-dire le vecteur $\mathbf{b}_i$ après sa « première proprification ». Soit $C(d, B)$ le coût de l'algorithme en dimension d avec des A_i satisfaisant $|A_i| \leqslant B$, auquel on a ôté les coûts de toutes les premières sous-phases (c'est-à-dire chaque première proprification de chaque phase). L'algorithme commence par réduire une base sac-à-dos constituée des $\lfloor d/2 \rfloor$ premiers vecteurs avant de considérer les suivants. Soient $(\mathbf{b}''_1, \dots, \mathbf{b}''_{\lfloor d/2 \rfloor})$ les vecteurs LLL-réduits correspondants. Si l'on exclut les premières sous-phases des phases suivantes, cela revient à réduire la base $(\mathbf{b}''_1, \dots, \mathbf{b}''_{\lfloor d/2 \rfloor}, \mathbf{b}'_{\lfloor d/2+1 \rfloor}, \dots, \mathbf{b}'_d)$, dont les longueurs des vecteurs sont bornées par $\approx$

$B^{2/d}$. La quantité $C(d,B)$ est inférieure à la somme des coûts de la réduction des vecteurs $(\mathbf{b}'_1,\ldots,\mathbf{b}'_{\lfloor d/2\rfloor})$ et de la LLL-réduction des vecteurs $(\mathbf{b}''_1,\ldots,\mathbf{b}''_{\lfloor d/2\rfloor},\mathbf{b}'_{\lfloor d/2+1\rfloor},\ldots,\mathbf{b}'_d)$:

$$C(d,B) = C\left(\frac{d}{2},B\right) + O\left(d^5\left(\frac{\log B}{d}\right)^2\right) = C\left(\frac{d}{2},B\right) + O(d^3(\log B)^2),$$

ce qui permet d'obtenir $C(d,B) = O(d^3\log^2 B)$, à condition que l'on ait $d^2 = O(\log B)$. Ceci conclut notre analyse heuristique.

De manière amusante, les réseaux sacs-à-dos, qui se comportent nettement mieux que ne le garantit la borne dans le cas le pire de l'algorithme L^2, permettent de construire des bases pour lesquelles la borne est fine : en reprenant les notations ci-dessus, il suffit de prendre comme entrée la base $(\mathbf{b}''_1,\ldots,\mathbf{b}''_{\lfloor d/2\rfloor},\mathbf{b}'_{\lfloor d/2\rfloor+1},\ldots,\mathbf{b}'_d)$. Cela revient à « remplacer artificiellement » $\log B$ par $d\log B$ dans la borne $O(d^3\log^2 B)$.

4.3.3 Évolution du temps d'exécution avec les facteurs (δ,η).

Il est suggéré dans [19] de modifier la paire de facteurs (δ,η) pour améliorer le temps d'exécution de l'algorithme. Puisque la borne sur le nombre d'itérations de boucles est inversement proportionnel à $-\lg\delta$, il est intéressant de prendre δ éloigné de la valeur qui donne les bases de meilleure qualité, c'est-à-dire $\delta = 1$. Cependant, baisser δ fait augmenter la précision nécessaire de l'algorithme L^2 : la taille de mantisse requise pour garantir la correction et la borne de complexité de l'algorithme L^2 est $\approx \lg\frac{(1+\eta)^2}{\delta-\eta^2}$. Ainsi, baisser le facteur de réduction δ sans changer la précision des calculs peut faire échouer l'algorithme en faisant apparaître des boucles infinies. Si l'on modifie la paire de facteurs (δ,η) et la précision des calculs de manière simultanée et cohérente, on obtient (voir le chapitre II-3) une borne de complexité proportionnelle à la quantité :

$$\frac{\lg^2\frac{(1+\eta)^2}{\delta-\eta^2}}{-\lg\delta},$$

dont le minimum est atteint pour $\eta = 1/2$ et $\delta \approx 0.59$. Cette valeur n'a peut-être aucun sens en pratique, puisqu'elle est calculée à partir de la borne dans le cas le pire de l'algorithme LLL, qui n'est vraisemblablement pas correcte dans le cas moyen. Elle permet cependant de montrer que l'on ne peut pas diminuer le temps d'exécution en choisissant δ très petit (c'est-à-dire proche de $1/4$).

4.4 Étude expérimentale de la précision numérique nécessaire.

Dans cette section, nous évoquons fréquemment des problèmes de pertes de précision. Deux principaux problèmes peuvent arriver : le processus de proprification paresseuse boucle (c'est le cas le plus fréquemment observé), ou les tests de Lovász bouclent. Il se peut aussi que la sortie calculée soit incorrecte, mais cela est moins fréquent : en général, si le calcul est incorrect, il ne finit pas.

4.4.1 Comportement le pire de l'algorithme LLL flottant.

Nous avons donné au chapitre II-3 de mauvais cas pour les pertes de précision dans l'algorithme LLL en arithmétique flottante. Pour une paire de paramètres (δ, η), nous considérons les vecteurs lignes de la matrice $d \times d$ définie par les équations suivantes :

$$\begin{array}{lcll} L_{j,j} & = & (\delta - \eta^2)^{j-d} & \\ L_{i,j} & = & (-1)^{i-j+1} L_{j,j} \cdot \text{random}[0.49, 0.5] & \text{si} \quad j < i \\ L_{i,j} & = & 0 & \text{si} \quad j > i. \end{array}$$

Pour observer plus facilement les anomalies, nous rajoutons une dernière ligne dont les entrées sont grandes et aléatoires. Expérimentalement, ces matrices font boucler[24] les implantations de l'algorithme LLL flottant pour des paramètres (δ, η), des dimensions d et des tailles ℓ de mantisses pour lesquels on a $\ell \approx d \cdot \lg \frac{(1+\eta)^2}{\delta - \eta^2}$. Nous donnons par exemple à l'URL `http://www.loria.fr/~stehle/these.html` un réseau de dimension 83 pour lequel la version prouvée de `fplll-1.2` semble boucler (lors de la proprification du dernier vecteur) avec 113 bits de précision.

4.4.2 Comportement « moyen » de l'algorithme LLL flottant.

Nous donnons dans ce paragraphe une analyse heuristique permettant d'estimer le comportement numérique moyen de l'algorithme LLL flottant. Plus exactement, nous estimons l'erreur moyenne commise lors de l'application de l'algorithme de Cholesky en arithmétique flottante. Pour une taille de mantisse fixée, cela nous permettra d'estimer à partir de quelle dimension l'erreur commise sur les coefficients $\mu_{i,j}$ et $r_{i,j}$ devient significative. Pour ensuite transposer les résultats à l'algorithme LLL, il faut faire intervenir le nombre d'itérations de boucles — si l'on calcule de nombreuses fois des coefficients de Gram-Schmidt, on augmente la probabilité que l'un d'entre eux soit erroné et que l'exécution échoue à ce moment là — ainsi que les indices des vecteurs qui les concernent — les itérations concernant les derniers vecteurs sont plus sujettes à des problèmes de pertes de précision que les itérations concernant les premiers car l'acuité des $\mu_{i,j}$ et des $r_{i,j}$ décroît avec j (voir le théorème 19, page 113, pour la borne dans le cas le pire).

Nous prenons le modèle aléatoire suivant, qui est une version simplifiée de celui présenté à la section 4.2 (il est possible d'adapter l'analyse au modèle exact de la section 4.2, mais cela rend l'étude plus complexe, avec un résultat asymptotique relativement équivalent) :

- Nous tirons des $\mu_{i,j}$ pour $i > j$ aléatoirement et indépendamment dans $[-\eta, \eta]$ (où η est fixé), suivant une distribution commune symétrique par rapport à 0. On a ainsi $\mathbb{E}[\mu] = 0$. On définit $\mu_2 = \mathbb{E}[\mu^2]$ et $\mu_{i,i} = 1$.
- Nous tirons des $\frac{r_{i+1,i+1}}{r_{i,i}}$ aléatoirement et indépendamment dans $[\delta, +\infty[$ (où $\delta < 1$ est fixé). Ces tirages sont indépendants de ceux des $\mu_{i,j}$. On définit $\delta_{-1} = \mathbb{E}\left[\frac{r_{i,i}}{r_{i+1,i+1}}\right]$.
- Les variables aléatoires $\mu_{i,j}$ et $\frac{r_{i+1,i+1}}{r_{i,i}}$ permettent de définir la matrice de Gram. Soit $r_{1,1}$ une constante arbitraire. La variable aléatoire $\langle \mathbf{b}_i, \mathbf{b}_j \rangle$ pour $i > j$ est définie

[24] lors de la proprification de la dernière ligne, à cause des coefficients de Gram-Schmidt erronés

par :

$$\langle \mathbf{b}_i, \mathbf{b}_j \rangle = r_{1,1} \cdot \left(\sum_{k=1}^{j} \mu_{j,k}\mu_{i,k} \prod_{l=1}^{k-1} \frac{r_{l+1,l+1}}{r_{l,l}} \right).$$

On définit de même les variables aléatoires $\|\mathbf{b}_i\|^2$:

$$\|\mathbf{b}_i\|^2 = r_{1,1} \cdot \left(\sum_{k=1}^{i} (\mu_{i,k})^2 \prod_{l=1}^{k-1} \frac{r_{l+1,l+1}}{r_{l,l}} \right).$$

- On définit les variables aléatoires $r_{i,j} = r_{1,1}\mu_{i,j} \prod_{l=1}^{j-1} \frac{r_{l+1,l+1}}{r_{l,l}}$, pour tous $i \geqslant j$.
- Nous supposons que nous avons commis une erreur relative ε de l'ordre de $2^{-\ell}$ (où ℓ est la précision des calculs) lors de la traduction de la valeur exacte $\|\mathbf{b}_1\|^2$ en un nombre flottant : $\Delta\|\mathbf{b}_1\|^2 = \varepsilon\|\mathbf{b}_1\|^2$.

Nous avons choisi une façon de choisir aléatoirement la matrice de Gram, et de commettre une erreur d'arrondi sur $\|\mathbf{b}_1\|^2$. Pour simplifier, nous supposons qu'il n'y a pas d'erreur d'arrondi sur les autres produits scalaires $\langle \mathbf{b}_i, \mathbf{b}_j \rangle$. Notre but est d'évaluer l'amplification de l'erreur d'arrondi $\Delta\|\mathbf{b}_1\|^2$ quand on calcule les $r_{i,j}$ et les $\mu_{i,j}$, en négligeant les termes d'erreur d'ordres supérieurs. Plus formellement, on cherche à étudier les variables aléatoires suivantes, définies récursivement, pour $i > j$:

$$\begin{aligned}
\Delta r_{1,1} &= \Delta\|\mathbf{b}_1\|^2 = \varepsilon\|\mathbf{b}_1\|^2, \\
\Delta r_{i,j} &= -\sum_{k=1}^{j-1} \left(\frac{\Delta r_{i,k} \cdot r_{j,k} + \Delta r_{j,k} \cdot r_{i,k}}{r_{k,k}} - \frac{\Delta r_{k,k} \cdot r_{i,k} \cdot r_{j,k}}{r_{k,k}^2} \right) \quad \text{pour } i > j, \\
\Delta r_{i,i} &= -\sum_{k=1}^{i-1} \left(\frac{2r_{i,k} \cdot \Delta r_{i,k}}{r_{k,k}} - \frac{\Delta r_{k,k} \cdot r_{i,k}^2}{r_{k,k}^2} \right) \quad \text{pour } i > 1.
\end{aligned}$$

Puisque $\Delta r_{i,k}$ et $\mu_{j,k}$ sont des variables aléatoires indépendantes (les seuls $\mu_{a,b}$ dont dépend $\Delta r_{i,k}$ sont tels que $b < k$), puisque $\Delta r_{j,k}$ et $\mu_{i,k}$ sont indépendantes (par symétrie), et $\Delta r_{k,k}, \mu_{i,k}$, et $\mu_{j,k}$ sont deux-à-deux indépendantes, on a :

$$\begin{aligned}
\mathbb{E}\left[\frac{\Delta r_{i,j}}{r_{j,j}}\right] = -\sum_{k=1}^{j-1} \left(\prod_{l=k}^{j-1} \mathbb{E}\left[\frac{r_{l,l}}{r_{l+1,l+1}}\right] \right) \Bigg(& \mathbb{E}\left[\frac{\Delta r_{i,k}}{r_{k,k}}\right] \mathbb{E}[\mu_{j,k}] + \mathbb{E}\left[\frac{\Delta r_{j,k}}{r_{k,k}}\right] \mathbb{E}[\mu_{i,k}] \\
& - \mathbb{E}\left[\frac{\Delta r_{k,k}}{r_{k,k}}\right] \mathbb{E}[\mu_{i,k}]\mathbb{E}[\mu_{j,k}] \Bigg).
\end{aligned}$$

Puisque $\mathbb{E}[\mu_{j,k}] = \mathbb{E}[\mu_{i,k}] = 0$, on a $\mathbb{E}\left[\frac{\Delta r_{i,j}}{r_{j,j}}\right] = 0$, pour tous $i > j$. De la même manière, on obtient, pour $i > 1$:

$$\begin{aligned}
\mathbb{E}\left[\frac{\Delta r_{i,i}}{r_{i,i}}\right] &= -\sum_{k=1}^{j-1} \left(\prod_{l=k}^{i-1} \mathbb{E}\left[\frac{r_{l,l}}{r_{l+1,l+1}}\right] \right) \left(2\mathbb{E}\left[\mu_{i,k} \cdot \frac{\Delta r_{i,k}}{r_{k,k}}\right] - \mathbb{E}\left[\frac{\Delta r_{k,k}}{r_{k,k}}\right] \mathbb{E}[\mu_{i,k}^2] \right) \\
&= \mu_2 \sum_{k=1}^{j-1} \delta_{-1}^{i-k} \left(\mathbb{E}\left[\frac{\Delta r_{k,k}}{r_{k,k}}\right] \right).
\end{aligned}$$

On en déduit que $\mathbb{E}\left[\frac{\Delta r_{i,i}}{r_{i,i}}\right]$ est de l'ordre de $(\delta_{-1}(1+\mu_2))^i\varepsilon$. Par exemple, si l'on suppose les $\mu_{i,j}$ uniformément répartis dans $[-1/2, 1/2]$, que $\delta_{-1} = 1.04$ (conformément aux observations relatées à la section 4.2), et que $\varepsilon \approx 2^{-53}$, on obtient : $\mathbb{E}\left[\frac{\Delta r_{i,i}}{r_{i,i}}\right] \approx 1.13^i \cdot 2^{-53}$. Pour $i = 200$, cette valeur est de l'ordre de 2^{-17}.

Nous avons analysé de manière grossière l'amplification de l'erreur d'arrondi effectuée sur $\|\mathbf{b}_1\|^2$, pour l'algorithme de factorisation de Cholesky. Si on veut adapter cette analyse à l'algorithme LLL, il faut prendre en compte le nombre de coefficients $r_{i,j}$ et $\mu_{i,j}$ calculés au cours de l'algorithme. Pour simplifier, on ne considère que les coefficients $r_{d,d}$, c'est-à-dire ceux pour lesquels l'erreur numérique est a priori la plus grande. On suppose aussi que ces calculs soient indépendants. Soit K le nombre d'itérations de boucles de l'algorithme LLL pour lesquelles $\kappa = d$. On considère qu'une erreur sur $r_{d,d}$ est significative si $\frac{\Delta r_{d,d}}{r_{d,d}}$ est de l'ordre de 2^{-3}. Si une telle erreur a lieu, alors il est vraisemblable que le test de Lovász correspondant soit erroné, et il peut y avoir un échange non désiré, ou l'inverse. Dans ce cas, la probabilité qu'il y ait au moins un calcul défectueux est de l'ordre de $1 - (1 - 2^{-17+3})^K \approx K2^{-14}$. S'il y a plusieurs millions d'échanges, il est donc tout à fait vraisemblable qu'il y aura une erreur numérique faisant dévier l'algorithme LLL de son comportement désiré. Par exemple, l'exécution bouclera lors d'une proprification à cause de coefficients de Gram-Schmidt incorrects : les coefficients de l'indice κ courant sont recalculés à chaque itération de la proprification paresseuse, mais si les coefficients d'indices inférieurs sont faux, il n'y a pas de raison pour que les coefficients recalculés soient plus corrects que les anciens.

L'analyse qui précède est tout à fait heuristique, et repose sur des hypothèses grossières et partiellement fausses, mais elle permet d'obtenir des ordres de grandeur que l'on peut appréhender en pratique. Pour des réseaux « aléatoires », nous observons expérimentalement des boucles infinies dûes à une précision trop faible autour des dimensions 175 à 185, avec des nombres d'itérations de boucle de l'ordre de 3 à 5 millions.

4.4.3 À propos de l'orthogonalisation de Givens et Householder.

Koy et Schnorr [95, 153] suggèrent de changer le procédé d'orthogonalisation sous-jacent à l'algorithme LLL pour améliorer le comportement pratique de ses variantes flottantes. Plus précisément, ils proposent de remplacer l'orthogonalisation de Gram-Schmidt par les algorithmes de Givens et Householder, connus pour être plus stables numériquement en toute généralité [71, 104]. Nous ne sommes pas ici dans le « cas général » puisque chaque orthogonalisation d'un vecteur est effectuée par rapport à des vecteurs LLL-réduits. Cette remarque est centrale dans la preuve de l'algorithme L^2. Il est cependant possible que les orthogonalisations de Givens et Householder soient plus stables, dans le sens où elles nécessiteraient en général des tailles de mantisse plus faibles à dimension fixée. L'article [153] va dans ce sens : Schnorr y montre que les pires cas pour Gram-Schmidt (donnés au début de cette section) peuvent être traités par l'orthogonalisation de Householder avec une précision nettement plus faible que celle requise par l'algorithme L^2.

L'exemple suivant montre une faiblesse des orthogonalisations de Givens et Household-

der : la précision requise semble devoir être proportionnelle à $\log B$ dans le cas le pire.

$$\begin{pmatrix} 1 & 1 \\ 2^{54}+1 & -2^{54}+1 \end{pmatrix} \qquad \begin{pmatrix} 1 & 1 & 0 \\ 2^{54}+1 & -2^{54}+1 & 0 \\ 2^{54} & 2^{54}-1 & 1 \end{pmatrix}.$$

Les vecteurs lignes de la matrice de gauche ne sont pas LLL-réduits ($\mu_{2,1} = 1$ est trop grand), mais la variante de l'algorithme LLL utilisant les rotations de Givens qui est implantée dans NTL, G_LLL_FP, affirme le contraire. Ce code boucle lorsque les vecteurs lignes de la matrice de droite lui sont donnés en entrée :

```
G_LLL_FP: too much loss of precision...stop!
Abort
```

Ce problème n'est pas lié à NTL, mais plutôt à l'algorithme d'orthogonalisation : la matrice est construite pour que le produit scalaire des deux premiers vecteurs ne tienne pas compte de la petite perturbation. Cette petite perturbation fausse la valeur de $\mu_{2,1}$, ce qui rend inconsistants les calculs visant à « proprifier » le troisième vecteur par rapport aux deux premiers. Ces entrées ne posent aucun problème à l'algorithme L^2, car celui-ci travaille sur la matrice de Gram, qui est toujours exacte. Par contre, elles sont gênantes pour les variantes rapide et heuristique de notre code. Pour parer cela, il faut tester si les produits scalaires calculés en arithmétique flottante ne sont pas très petits par rapport au produit des normes des vecteurs considérés. Si c'est le cas, on calcule alors le produit scalaire avec une arithmétique entière avant de l'arrondir. Le problème de ce procédé est que l'on ne peut plus garantir que l'algorithme a une complexité quadratique quand la dimension est fixée. Cette idée se trouve déjà dans [154] et dans NTL.

Les orthogonalisations de Givens et de Householder pourraient être intéressantes dans le contexte suivant. Supposons que l'on veuille réduire un réseau dont la dimension exclut l'orthogonalisation de Gram-Schmidt, par exemple en dimension 200 (voir le paragraphe précédent). Il y a alors deux alternatives. On peut tout d'abord choisir d'utiliser une bibliothèque de flottants en précision arbitraire (comme MPFR ou la classe RR de NTL), mais cela induit un surcoût tout à fait considérable par rapport à l'utilisation des flottants double précision, car ceux-ci utilisent directement les instructions du processeur. Une autre alternative est d'utiliser l'orthogonalisation de Givens ou celle de Householder, qui nécessiteraient des précisions plus faibles, et donc qui permettraient d'utiliser encore des flottants double précision. L'inconvénient de cette deuxième solution est que ces orthogonalisations font intervenir plus d'opérations arithmétiques de base que l'orthogonalisation de Gram-Schmidt : en dimension d, l'orthogonalisation de Gram-Schmidt telle que décrite au chapitre II-3 nécessite $\frac{d^3}{6} + O(d^2)$ opérations arithmétiques (auxquelles il faut rajouter $\frac{d^3}{2} + O(d^2)$ si l'on prend en compte les produits scalaires), alors que l'orthogonalisation de Householder telle que décrite dans [153] nécessite $d^3 + O(d^2)$ opérations arithmétiques (dont des racines carrées, mais en nombre négligeable). L'orthogonalisation de Householder est donc plus coûteuse à dimension fixée, mais cette différence est sûrement compensée par le surcoût entre double précision et précision arbitraire.

4.5 Problèmes ouverts.

Les remarques expérimentales faites dans ce chapitre amènent naturellement à plusieurs problèmes. Pour chacun de ces problèmes, il s'agirait de déterminer comment exploiter les réponses pour accélérer les variantes pratiques de l'algorithme LLL en arithmétique flottante.

- Prouver la borne de complexité heuristique $O(d^3 \log^2 B)$ pour les bases sac-à-dos.
- Expliquer la géométrie des bases locales, comprendre comment elle se généralise pour l'algorithme de réduction BKZ de Schnorr [150], et essayer d'en tirer partie pour réduire plus fortement les bases (c'est-à-dire par exemple diminuer le défaut d'orthogonalité, ou le facteur d'approximation du premier minimum).
- Programmer une variante de l'algorithme L^2 qui change dynamiquement la précision et les facteurs de réduction (δ, η) pour aboutir plus rapidement à une base LLL-réduite de la qualité exigée : le principe serait par exemple de commencer avec une précision faible, de détecter si celle-ci n'est pas suffisante, puis de l'augmenter.

Du point de vue de l'implémentation de l'algorithme, un problème crucial est de déterminer si l'on peut ne pas avoir besoin de calculer la matrice de Gram pour la variante prouvée de `fplll-1.2`. Dans la variante heuristique, on calcule les produits scalaires en arithmétique flottante et on les recalcule en arithmétique entière si on détecte une trop grande annulation. Cela affecte la complexité de l'algorithme, qui a priori devient cubique en $\log B$. Une solution pourrait être de calculer les produits scalaires en arithmétique flottante, en conservant une information sur la taille de ceux-ci au cours de l'exécution : si l'on connaît la taille du produit scalaire, on sait le calculer de manière précise en arithmétique flottante, en ne considérant dans les nombres de départ que leurs parties pouvant influer sur le résultat.

Troisième partie

Pires cas pour l'arrondi de fonctions.

Chapitre III-1

Recherche des pires cas de fonctions mathématiques.

Le travail de recherche correspondant à ce chapitre a fait l'objet d'une publication dans les actes de la conférence ARITH'16 (Sixteenth Symposium on Computer Arithmetic) [167], dont une version corrigée et étendue a été publiée dans la revue IEEE Transactions on Computers [168]. Le présent chapitre est une version très étendue de ces deux publications.

Sommaire

1.1 Méthode de Coppersmith. 160
1.1.1 Petites racines de polynômes modulaires univariés. 160
1.1.2 Petites racines de polynômes modulaires multivariés. . . . 161
1.2 La méthode de Lefèvre et sa généralisation. 162
1.2.1 Motivations. 162
1.2.2 La méthode de Lefèvre. 164
1.2.3 Généralisation de la méthode de Lefèvre. 165
1.2.4 Principaux résultats. 168
1.3 Une première analyse de l'algorithme. 171
1.3.1 Deux bornes dans l'analyse de complexité. 171
1.3.2 Première analyse de complexité. 172
1.3.3 Le cas $d = \alpha = 2$. 173
1.4 Le cas $d = 2$. 176
1.4.1 Borne sur le déterminant du réseau. 176
1.4.2 Borne de Coppersmith et conclusion. 179
1.5 Le cas $d \geqslant 3$. 180
1.5.1 La famille de polynômes choisie. 180
1.5.2 Une méthode pour borner le déterminant d'une matrice. . 181
1.5.3 Calcul du déterminant de la matrice B_3 (introduite au paragraphe 1.5.1). 182
1.5.4 Borne de Coppersmith et conclusion. 188
1.6 Le cas de fonctions de plusieurs variables. 189

1.6.1 L'analyse naïve. 190
1.6.2 Analyse du cas $d = 2$. 191
1.6.3 Analyse du cas $d \geqslant 3$. 193
1.7 Données expérimentales. 195
1.7.1 Recherche des pires cas d'une fonction d'une variable. . . . 196
1.7.2 Recherche de pires cas de fonctions de deux variables. . . . 198
1.7.3 Calculs de bornes supérieures pour l'arrondi correct, et inversion d'une suite de bits. 199
1.8 Problèmes ouverts. 200

Comme nous l'avons vu au chapitre I-2, le standard IEEE-754 [2] concernant l'arithmétique flottante en base 2 spécifie que les quatre opérations arithmétiques élémentaires ($+, -, \times$ et $\div$) et la racine carrée soient correctement arrondies. Cela signifie que pour une taille de mantisse et un mode d'arrondi donnés, le résultat renvoyé pour (a op b) ou pour $\sqrt{a}$ doit être l'arrondi du résultat en précision infinie. Il serait tentant d'étendre ce standard aux fonctions mathématiques élémentaires (les fonctions trigonométriques $\sin, \cos, \tan$ et leurs inverses, les fonctions exponentielles et logarithmiques népériennes et en base 2, et les fonctions trigonométriques hyperboliques et leurs inverses). Cela permettrait la portabilité et la reproductibilité des programmes les utilisant, d'une machine à l'autre. Le principal obstacle pour la standardisation est le coût algorithmique d'une implantation correcte de ces fonctions, c'est-à-dire prouvée — dans un sens très général. Plusieurs difficultés se posent pour implanter ces fonctions correctement et efficacement. Une question naturelle est de savoir quelle taille de mantisse il faut utiliser pour les calculs intermédiaires pour pouvoir décider de l'arrondi correct pour toutes les entrées. Nous nous intéressons à cette question dans le présent chapitre. Dans le chapitre III-2, on utilisera des techniques similaires pour accélérer une étape importante des implantations de certaines fonctions mathématiques élémentaires.

La question que nous abordons ici est celle du dilemme du fabricant de tables (de calcul). Le but est de trouver les nombres représentables x avec une taille de mantisse fixée tels que le réel $f(x)$ soit très difficile à arrondir, pour le mode d'arrondi spécifié. Plus précisément, pour le mode d'arrondi au plus proche, on va chercher les nombres représentables x tels que $f(x)$ est extrêmement proche du milieu de deux nombres représentables (de telle sorte qu'il faudra beaucoup de bits de $f(x)$ pour l'arrondir correctement), et pour les arrondis dirigés, on cherchera les nombres représentables x pour lesquels $f(x)$ est très proche d'un nombre représentable. Si la précision d'entrée-sortie de la fonction f est de n bits et si l'on se restreint à l'intervalle $[1/2, 1[$ et qu'on suppose $f([1/2, 1[) \subset [1/2, 1[$, cela signifie que l'on cherche à trouver les $x \in [|2^{n-1}, 2^n - 1|]$ tels que[25] :

$$\left| \left[2^{n+e_1} f\left(\frac{x}{2^n}\right) + e_2 \right] \bmod 1 \right| \leqslant \frac{1}{2^m},$$

avec $e_1 = e_2 = 0$ dans le cas des arrondis dirigés et $e_1 = 1, e_2 = 1/2$ dans le cas de l'arrondi au plus proche, et où m mesure la « qualité » du mauvais cas x. Dans ce chapitre, nous

[25]Nous rappelons que $a \bmod b$ est le réel $c \in \left]\frac{b}{2}, \frac{b}{2}\right]$ tel qu'il existe un entier k pour lequel $a = c + bk$. En particulier, le réel $a \bmod 1$ est la partie fractionnaire centrée de a.

chercherons à résoudre le problème plus général d'inverser une fonction quand les premiers bits de la sortie sont perdus, ou inconnus.

Définition 29 (Problème d'inversion d'une fonction quand les premiers bits sont inconnus). *Soit* $f : [1/2, 1[\to [1/2, 1[$. *En entrée sont donnés quatre nombres réels* N_1, N_2, M *et* c. *Le but est de trouver toutes les solutions* $x \in [|N_1/2, N_1 - 1|]$ *de l'équation ci-dessous si une telle solution existe, et sinon de dire qu'il n'y a pas de solution :*

$$\left| \left[N_2 f\left(\frac{x}{N_1}\right) + c \right] \mod 1 \right| \leqslant \frac{1}{M}. \tag{1.1}$$

Remarquons que dans cette définition, les entrées N_1, N_2, M et c sont des nombres réels. On supposera qu'ils n'ont qu'un nombre fini de bits après la virgule et qu'ils sont donnés par la suite finie de leurs bits (sans exposant). On supposera que $c < 1$, et que $N_1, N_2, M > 1$. S'ils ont au plus k bits après la virgule, la taille de l'entrée est alors de l'ordre de $k + \lg N_1 + \lg N_2 + \lg M$. En prenant $c = 0$ et $N_1 = N_2 = 2^n$, on retrouve bien le dilemme du fabricant de tables en précision n, pour les arrondis dirigés. En effet, si x satisfait l'équation (1.1), cela signifie qu'il est très proche d'un mot machine : quand il s'agit d'arrondir correctement $f(x)$, il faut alors calculer une approximation très fine pour déterminer si la valeur réelle est juste au-dessus ou juste au-dessous du nombre flottant proche. Par exemple avec $n = 7$ et $f = \sin$, on a :

$$\sin(0.1011001)_2 = 0.\underbrace{1010010}_{7}\underbrace{00000000000000}_{14}111\ldots$$

En prenant $c = 0$ et M extrêmement grand, on cherche en fait les valeurs exactes de la fonction f, c'est-à-dire les nombres représentables x pour lesquels $f(x)$ est un nombre représentable. En prenant $c = 1/2, N_1 = 2^n, N_2 = 2^{n+1}$ et M extrêmement grand, on cherche les nombres représentables x pour lesquels $f(x)$ est exactement le milieu de deux nombres représentables. Une autre motivation du problème ci-dessus est d'inverser une fonction f quand on n'a qu'une partie des bits de la sortie. Imaginons par exemple que l'on a calculé l'image d'un nombre représentable avec une précision d'entrée de n_1 bits, que cette image a été calculée avec une précision infinie, ou tout du moins très grande, mais que les n_2 premiers bits de l'image ont été perdus. Alors, en lisant les m premiers bits que l'on connaît, et en les appelant c, on a une instance du problème ci-dessus, avec $N_1 = 2^{n_1}, N_2 = 2^{n_2}$ et $M = 2^m$. Ce type de questions est intéressant d'un point de vue cryptographique : si le problème est difficile, alors on a un générateur de bits pseudo-aléatoire efficace. En effet, la graine serait l'entrée x, et la sortie serait la suite des bits de $f\left(\frac{x}{N_1}\right)$, sauf les n_2 premiers. Résoudre l'équation (1.1) est alors le rôle de l'attaquant. Cette question a déjà été soulevée par Blum, Blum et Shub [23] dans le célèbre article où ils décrivent le générateur par carrés modulaires successifs : avant de décrire ce générateur cryptographiquement sûr (il est calculatoirement équivalent à la factorisation du module), ils décrivent le « générateur $1/N$ », qui consiste à prendre $f(x) = 1/x$ dans notre situation. Ce « générateur $1/N$ » n'est pas sûr car on peut facilement retrouver la graine avec l'algorithme d'Euclide : $2\lg N + O(1)$ bits consécutifs suffisent pour obtenir N. Dans [89], Kannan, Lenstra et Lovász s'intéressent aux suites de bits produites par des

fonctions algébriques, et montrent qu'elles ne sont pas aléatoires calculatoirement, en utilisant l'algorithme LLL : si d est le degré d'algébricité de la fonction — le degré de son polynôme minimal — et H sa hauteur — le maximum des valeurs absolues des coefficients de son polynôme minimal, alors $O(d^2 + d \log H)$ bits suffisent. Ici, nous nous intéressons à la même question, mais pour des fonctions f plus générales, comme par exemple $f(x) = \exp x$. Il nous faudra cette fois-ci de l'ordre de n^2 bits pour retrouver la graine en temps polynomial, si $n = n_1 = n_2$.

Revenons au dilemme du fabricant de table. Dans le cas des quatre opérations arithmétiques élémentaires, il est relativement simple à résoudre (voir [4] par exemple). Par ailleurs, plusieurs auteurs [83, 103] ont montré que dans le cas des fonctions algébriques, l'image d'un nombre représentable ne peut pas être arbitrairement proche d'un nombre représentable ou du milieu de deux nombres représentables, sauf si cette image est exactement un nombre représentable ou le milieu de deux nombres représentables. Cependant, ces bornes deviennent trop grandes quand le degré d'algébricité de la fonction augmente. De telles bornes existent aussi pour certaines fonctions transcendantes [135], mais sont encore plus grandes, et inutilisables en pratique. De façon intéressante, la borne de [135] pour la fonction exponentielle dans le cas d'entrées du type $\frac{x}{2^n}$ avec $x \in [|2^{n-1}, 2^n - 1[|$ est asymptotiquement $m = O(n^2)$, c'est-à-dire que l'équation (1.1) ne peut avoir de solution pour $N_1 = N_2 = 2^n, c = 0$ et $M = 2^{kn^2}$ pour une certaine constante k. Pour les paramètres $N_1 = N_2 = 2^n$ et $M = 2^{n^2}$, notre algorithme a une complexité polynomiale heuristique, ce qui en fait en quelque sorte une version efficace de [135]. Dans une proposition de normalisation des fonctions mathématiques élémentaires, Defour *et al.* [52] proposent de définir plusieurs degrés de garanties pour les implantations des fonctions mathématiques élémentaires. Dans les deux degrés les plus contraignants, le dilemme du fabricant de tables doit être résolu (au moins pour les sous-domaines les plus utilisés des fonctions).

D'un point de vue algorithmique, le meilleur algorithme pour résoudre le dilemme du fabricant de tables était celui de Lefèvre [105, 106, 107, 108], qui est une variante de l'algorithme d'Euclide, et qui résout l'équation (1.1) pour $N_1 = N_2 = 2^n, c = 0, M = 2^{\frac{n}{3}}$ en temps $2^{\frac{2n}{3}+o(n)}$ (cette borne de complexité est heuristique). À l'aide de cet algorithme, un travail systématique a été réalisé par Lefèvre et Muller [109], qui ont publié de nombreux pires cas pour l'arrondi de fonctions élémentaires en double précision (53 bits de mantisse), pour tous les exposants possibles pour certaines fonctions. Quand la précision augmente, leur approche devient trop coûteuse. Les méthodes exhaustives de recherche des mauvais cas — dont la méthode de Lefèvre et celle que nous présentons dans ce chapitre font partie — permettent de trouver le pire cas, mais sont en général très onéreuses. Si n est la précision, elles ont une complexité de l'ordre de $2^{cn+o(n)}$, où c est une constante qui dépend de la méthode : avec la recherche naïve, on a $c = 1$, avec l'algorithme de Lefèvre on a $c = 2/3$, et nous montrons dans ce chapitre comment obtenir $c = 1/2$. L'algorithme que nous décrivons généralise de façon très naturelle l'algorithme de Lefèvre.

Si l'on exclut la méthode de Lefèvre, il y a peu de travaux antérieurs sur la résolution du dilemme du fabricant de tables. Dans les travaux connexes, on peut mentionner un algorithme d'Elkies [58] qui utilise la réduction de réseaux pour trouver des points rationnels de petite hauteur qui sont proches d'une courbe. Son exemple

$$5853886516781223^3 - 44788492842840204230791 8^2 = 1641843$$

correspond à un mauvais cas de la fonction $x^{3/2}$ pour une précision d'entrée de 53 bits et une précision de sortie de 79 bits. Son autre exemple

$$2220422932^3 - 283059965^3 - 2218888517^3 = 30$$

correspond à un mauvais cas de la fonction $(x^3 + y^3)^{1/3}$ pour une précision d'entrée et de sortie de 32 bits. Gonnet [72] utilise lui aussi de la réduction de réseaux pour trouver des mauvais cas pour l'arrondi des fonctions mathématiques. Ces deux algorithmes s'apparentent en fait à la méthode de Lefèvre.

Le chapitre s'organise de la manière suivante. Nous décrivons la méthode de Coppersmith pour trouver les petites racines de polynômes modulaires à la section 1.1. L'algorithme décrit dans ce chapitre utilise cette méthode dans le cas de polynômes modulaires multivariés. Dans la section 1.2, nous rappelons la méthode de Lefèvre, décrivons notre algorithme, et donnons nos résultats en expliquant ce qu'ils impliquent dans les cas particuliers de la recherche de mauvais cas et de la cryptanalyse du générateur pseudo-aléatoire décrit plus haut. Notre algorithme fait intervenir deux paramètres : d et α. En fait, l'algorithme se décompose en deux phases principales : une première phase d'approximation polynomiale, qui fait intervenir le paramètre d — il s'agit du degré des polynômes approximateurs ; et une deuxième phase qui utilise la méthode de Coppersmith et qui fait intervenir le paramètre α. Dans la section 1.3, nous donnons une première analyse de l'algorithme et expliquons en détail comment améliorer celle-ci dans le cas où $d = \alpha = 2$, ce qui correspond au choix de paramètres le plus simple qui permet d'améliorer l'algorithme de Lefèvre. Dans la section 1.4, nous analysons l'algorithme dans le cas où $d = 2$ et où α est quelconque, puis dans la section 1.5 nous nous intéressons à un paramètre $d \geqslant 3$ avec α quelconque. La section 1.6 explique brièvement ce que deviennent l'algorithme et son analyse de complexité lorsque l'on considère des fonctions de plusieurs variables. À la section 1.7 nous discutons du comportement pratique de l'algorithme, et enfin, à la section 1.8 nous décrivons quelques questions ouvertes.

Le lecteur désireux d'aboutir rapidement aux résultats principaux de ce chapitre peut ne lire que les sections 1.2 et 1.5.

Remarque : Pour simplifier l'exposé de la méthode, nous fixons une fonction $f : [1/2, 1[\to [1/2, 1[$ qui est $\mathcal{C}^\infty$ sur $[1/2, 1[$ et pour laquelle il existe une constante K telle que :

$$\forall i \geqslant 1, \forall x \in [1/2, 1[, |f^{(i)}(x)| \leqslant i!K,$$

où $f^{(i)}$ est la dérivée i-ième de la fonction f. Dans le cas où les dérivées sont plus grandes, les bornes peuvent être adaptées : nous avons gardé explicitement les termes dépendants de K quasiment jusqu'à la fin des preuves.

1.1 Méthode de Coppersmith.

Nous rappelons ici la méthode de Coppersmith pour trouver les petites racines de polynômes modulaires multivariés. Nous utiliserons de manière intensive la généralisation de cette méthode aux polynômes multivariés tout au long de la partie III de cette thèse. Pour une introduction plus complète sur ce sujet, nous renvoyons le lecteur aux articles de Coppersmith sur ce sujet [43, 44, 45, 46], et aux thèses de doctorat de Howgrave-Graham [81] et May [124]. Dans un premier temps, nous décrivons la méthode dans le cas de polynômes univariés, avant de nous intéresser au cas de polynômes multivariés, pour lequel plus de difficultés se posent.

1.1.1 Petites racines de polynômes modulaires univariés.

Pour des polynômes univariés, la méthode de Coppersmith est rigoureuse. Comme nous le verrons au paragraphe suivant, cela ne sera plus le cas pour des polynômes de plusieurs variables.

Théorème 23 (Coppersmith [44]). *Soient N un entier positif et P un polynôme unitaire (son coefficient dominant est 1) de degré d, écrit comme une liste de $d+1$ entiers en base 2. Alors en temps déterministe polynomial en $\log N$ et d, on peut trouver tous les entiers x tels que :*

$$P(x) = 0 \mod N \quad et \quad |x| \leqslant N^{\frac{1}{d}} \quad (*).$$

Démonstration. La méthode de Coppersmith fonctionne de la manière suivante. Elle dépend d'un paramètre $\alpha \geqslant 1$, qui est fixé de sorte à optimiser la borne X sur les valeurs absolues des racines de P que l'on saura calculer. Le but ici est d'atteindre $X = N^{1/d}$, et pour cela, nous ferons tendre α vers l'infini en prenant $\alpha = \Theta(\log N)$. La méthode comprend quatre étapes principales.

1. Construction d'une famille d'équations polynomiales. Tout d'abord, à partir de l'équation $P(x) = 0$, on dérive un grand ensemble d'équations modulo N^α satisfaites par x. Plus précisément, nous considérons la famille de polynômes $\{P_{i,j}(x), 0 \leqslant i \leqslant d-1, 0 \leqslant j \leqslant \alpha\}$, avec $P_{i,j}(x) = N^{\alpha-j}x^i P^j(x)$. Si x est racine de P modulo N, elle est racine de tous les $P_{i,j}$ modulo N^α.

2. Construction du réseau correspondant aux polynômes. À chaque polynôme $P_{i,j}$ on fait correspondre un vecteur de dimension $d(\alpha+1)$, qui est constitué des coefficients de $P_{i,j}(xX)$, où les puissances de x sont rangées par ordre croissant, et X sera fixé ultérieurement. On ordonne les polynômes $P_{i,j}$ par ordre croissant pour j, puis par ordre croissant pour i en cas d'égalité. En transformant chaque $P_{i,j}$ en un vecteur ligne, on obtient une matrice carrée triangulaire inférieure de dimension $d(\alpha+1)$. Nous considérons le réseau engendré par ses lignes. La matrice suivante est la forme de la matrice que l'on obtiendrait pour $d = 2$ et $\alpha = 3$, les entrées notées « $-$ » étant celles qui peuvent être non nulles. Les colonnes correspondent

aux monômes $1, x, \ldots, x^6$, et les lignes aux polynômes $N^3, N^3x, \ldots, (P(x))^3$.

$$\left(\begin{array}{c|ccccccc} & 1 & x & x^2 & x^3 & x^4 & x^5 & x^6 \\ \hline N^3 & N^3 & & & & & & \\ N^3x & & XN^3 & & & & & \\ N^2P(x) & - & - & X^2N^2 & & & & \\ N^2xP(x) & & - & - & X^3N^2 & & & \\ NP^2(x) & - & - & - & - & X^4N & & \\ NxP^2(x) & & - & - & - & - & X^5N & \\ P^3(x) & - & - & - & - & - & - & X^6 \end{array}\right).$$

3. La réduction du réseau. On réduit le réseau L engendré par les lignes de la matrice précédemment construite, avec l'algorithme LLL. Comme α sera choisi au plus polynomial en d et $\log N$ et qu'on aura $X \leqslant N$, cette étape de réduction a un coût polynomial. Il s'agit maintenant d'estimer la longueur du premier vecteur renvoyé, noté $\mathbf{b}_1$. Avec le théorème 3, page 15, on sait que :

$$\begin{aligned} \|\mathbf{b}_1\| &\leqslant 2^{\frac{\dim L}{4}} (\text{vol } L)^{\frac{1}{\dim L}} \\ &\leqslant 2^{d(\alpha+O(1))} \left(X^{\frac{d^2}{2}(\alpha^2+O(\alpha))} N^{\frac{d}{2}(\alpha^2+O(\alpha))} \right)^{\frac{1}{d(\alpha+O(1))}} \\ &\leqslant 2^{d(\alpha+O(1))} X^{\frac{d}{2}(\alpha+O(1))} N^{\frac{\alpha}{2}+O(1))}, \end{aligned}$$

quand α tend vers l'infini.

4. Calcul des petites racines sur $\mathbb{Z}$. Le polynôme correspondant au vecteur $\mathbf{b}_1$ est une combinaison linéaire entière de polynômes pour lesquels toute racine éventuelle de P modulo N est racine modulo N^α. Par conséquent, si x est racine de P modulo N, elle est racine modulo N^α du polynôme Q associé au vecteur $\mathbf{b}_1$. Si la norme ℓ_1 du vecteur $\mathbf{b}_1$ est suffisamment petite, alors l'inégalité triangulaire nous assure qu'en fait x est racine du polynôme R sur $\mathbb{Z}$. Pour cela, il nous suffit d'avoir l'inégalité :

$$2^{d(\alpha+O(1))} X^{\frac{d}{2}(\alpha+O(1))} N^{\frac{\alpha}{2}+O(1))} \leqslant N^\alpha.$$

En prenant $X = N^{\frac{1}{d}\left(1+O\left(\frac{1}{\alpha}\right)\right)}$, cette inégalité est satisfaite. Il ne reste plus qu'à chercher les racines du polynôme R sur $\mathbb{Z}$ qui sont plus petites que X, ce qui peut être réalisé en temps polynomial, par exemple à l'aide d'une recherche dichotomique.

En prenant $\alpha = O(\log N)$, on obtient toutes les racines x qui satisfont $|x| \leqslant \frac{1}{2}N^{1/d}$. En réitérant toute la méthode pour les polynômes $P(x+X)$ et $P(x-X)$, on trouve toutes les racines x qui satisfont $|x| \leqslant N^{1/d}$. □

1.1.2 Petites racines de polynômes modulaires multivariés.

Dans toute la partie III de la thèse, nous utiliserons la technique de Coppersmith avec des polynômes multivariés. Le déroulement général des étapes est le même que dans le cas univarié. Les différences principales sont les suivantes :

- Le choix de la famille de polynômes est moins automatique. Par exemple, dans le cas d'un polynôme en deux variables $P(x, y)$, on peut utiliser les puissances de P, et les multiplier par tous les monômes $x^k y^l$. Cela donne souvent des familles de cardinalités trop grandes pour que les vecteurs associés soient linéairement indépendants (et donc pour que les volumes des réseaux associés soient aisément calculables). Le choix de la famille de polynômes optimisant la borne de calculabilité des racines dépend fortement de la forme du polynôme P : ce choix peut dépendre en particulier des monômes apparaissant dans P, ainsi que des coefficients.
- La matrice construite à partir de la famille de polynômes a des colonnes indicées par des monômes en plusieurs variables. Cela complexifie surtout la description de la méthode, en faisant intervenir des multi-indices.
- Si le polynôme P a k variables, on a besoin de garder au moins k vecteurs courts à l'issue de la réduction de réseaux. En effet, l'algorithme LLL nous permet d'obtenir des polynômes en k variables dont il suffit ensuite de chercher les racines sur $\mathbb{Z}$. Pour pouvoir éliminer ces variables, à l'aide de résultants ou toute autre méthode, on a besoin d'au moins k polynômes. Il se peut que lors de l'élimination des variables, on aboutisse au polynôme nul, ou que l'on n'arrive pas à éliminer suffisamment de variables.

Pour que la méthode fonctionne, il faut que les petits polynômes sur $\mathbb{Z}$ calculés soient algébriquement indépendants, ce que l'on ne sait pas aujourd'hui garantir à partir du polynôme initial. La méthode est donc heuristique dans le cas (modulaire) multivarié : elle fonctionne sous l'hypothèse que les polynômes correspondants aux vecteurs courts trouvés sont algébriquement indépendants.

La méthode de Coppersmith s'est avérée très utile en cryptographie, que ce soit la méthode univariée prouvée [27, 45, 125], ou la méthode heuristique multivariée [26, 123].

1.2 La méthode de Lefèvre et sa généralisation.

Dans cette section, nous décrivons quelques motivations de l'équation (1.1), expliquons la méthode de Lefèvre qui la résout pour un jeu de paramètres restreint, et décrivons notre généralisation.

1.2.1 Motivations.

On se donne ici une fonction $f : [1/2, 1[\to [1/2, 1[$, et une précision d'entrée-sortie de n bits : les nombres $x \in [1/2, 1]$ donnés en entrée à f et renvoyés par l'implantation de f sont des flottants de n bits de mantisse. Notre but est de trouver les mauvais cas pour l'arrondi de la fonction f.

Définition 30 (m-mauvais cas). *Soit $m \geqslant 1$. On dit que $x \in \frac{1}{2^n}[|2^{n-1}, 2^n - 1|]$ est un m-mauvais cas pour f en précision n si :*

$$| \left[2^{n+e_1} f(x) + e_2\right] \mod 1| \leqslant 2^{-m},$$

avec $e_1 = 2_2 = 0$ dans le cas des arrondis dirigés, et $e_1 = 1, e_2 = 1/2$ dans le cas de l'arrondi au plus proche.

Il est évident que la recherche des mauvais cas d'une fonction pour une précision donnée est directement liée à l'équation (1.1) avec $c = 0$. La figure 1.1 illustre graphiquement la définition précédente. Les mauvais cas pour l'arrondi correspondent aux points de la grille avec une petite distance verticale à la courbe.

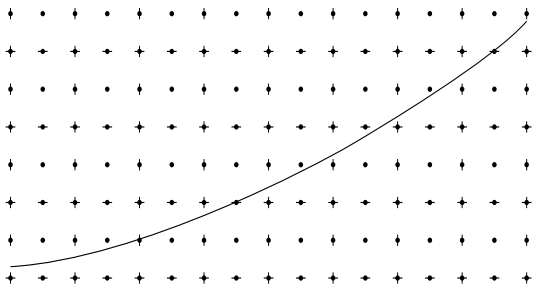


FIG. 1.1 – Le graphe d'une fonction et la grille des nombres représentables.

Si l'on suppose qu'au-delà du n-ième les bits de f se comportent comme des variables aléatoires indépendantes uniformes dans $\{0, 1\}$, alors on s'attend à obtenir un pire cas avec $m \approx n$. En effet, la probabilité qu'un x soit un n-mauvais cas est exactement la probabilité que $f(x)$ ait une suite de n zéros consécutifs ou n uns consécutifs après le $(n+e)$-ième bit. Comme il y a 2^{n-1} entrées x possibles, on a une probabilité non-négligeable que l'une d'entre elles soit un n-mauvais cas. Ce fait s'observe très bien en pratique [107].

Lorsque l'on cherche le pire cas pour une fonction $f : [1/2, 1[\to [1/2, 1[$ et une précision données, on ne peut pas choisir M nettement plus grand que 2^n dans l'équation (1.1), car il se peut alors qu'il n'y ait pas de solution. Dans ce cas, nous obtiendrons une borne sur la qualité du pire cas, mais pas la borne la plus fine possible. Dans la situation de la recherche du pire cas pour l'arrondi, nous avons donc $N_1 = 2^n, N_2 = 2^{n+e}$ et $M \leqslant N_1$.

On peut remarquer qu'il peut tout de même être intéressant de choisir $M \geqslant 2^n$ pour déterminer les mauvais cas pour l'arrondi : les valeurs exactes de la fonction f — c'est-à-dire les entrées x qui sont des m-mauvais cas pour tout m — sont solutions de l'équation (1.1) pour $c = 0$ et toute valeur de $M = 2^m$. Ainsi, si l'on cherche les valeurs exactes, il peut être intéressant de choisir M plus grand que N_1, si cela rend l'algorithme plus efficace. Si cela ne rend pas la recherche des solutions plus rapide, alors cela n'a aucun intérêt puisqu'une valeur exacte est de toute façon un mauvais cas pour tout $m \leqslant n$.

D'un point de vue pratique, on a une connaissance précise des pires cas des fonctions mathématiques élémentaires en double précision[26] ($n = 53$), et, en double précision étendue ($n = 64$), les pires cas de la fonction 2^x entre $1/2$ et 1 ont été calculés par Paul Zimmermann et Vincent Lefèvre à l'aide d'une implantation de l'algorithme décrit dans ce chapitre[27].

Une autre motivation de l'équation (1.1) est d'inverser une fonction f quand on a perdu les premiers bits de $f(x)$. Cette question s'exprime très naturellement sous la forme

[26] voir l'URL `http://perso.ens-lyon.fr/jean-michel.muller/Intro-to-TMD.htm`

[27] voir l'URL `http://www.loria.fr/equipes/spaces/slz.fr.html`

d'un générateur pseudo-aléatoire :

Définition 31. *On dispose de deux paramètres de sécurité n_1 et n_2. On choisit un entier x aléatoirement dans $[|2^{n_1-1}, 2^{n_1}-1|]$. La suite de bits pseudo-aléatoire est constituée de l'expansion binaire de $f(x)$, où les n_2 premiers bits ont été supprimés.*

Dans cette situation, on choisit $N_1 = 2^{n_1}$ et $N_2 = 2^{n_2}$ dans l'équation (1.1). On dira que le générateur est cassé si avec un paramètre M de la forme $2^{\mathcal{P}\mathrm{ol}(n_1,n_2)}$ on sait résoudre l'équation (1.1) en temps polynomial.

1.2.2 La méthode de Lefèvre.

Lefèvre [106, 107, 108] a décrit sa méthode dans le contexte de la recherche de mauvais cas pour l'arrondi des fonctions mathématiques. On approche la fonction f par des droites sur de petits sous-intervalles de $[1/2, 1[$, de longueur $\frac{2T}{N}$ (on a $N = 2^n$, où n est la précision choisie). Plus précisément, on prend $\frac{N}{4T}$ petits intervalles de longueurs $\frac{2T}{N}$ (on considérera leurs centres, ce qui explique la constante « 2 »). Si l'on considère $N \cdot [1/2, 1[$ au lieu de $[1/2, 1]$, les entrées deviennent des entiers, et on considère alors $\frac{N}{4T}$ intervalles de longueurs $2T$. Dans la suite, par défaut nous considérerons l'intervalle réel $[1/2, 1[$ plutôt que l'intervalle entier $N[1/2, 1[$. Les approximations linéaires sont calculées à partir d'approximations de plus grand degré sur des intervalles plus larges, en utilisant une méthode de « table de différences ». La façon dont sont construites les approximations de degré un n'a pas d'importance pour l'analyse de l'algorithme. On peut par exemple supposer qu'il s'agisse des débuts des approximations de Taylor calculées au milieu des sous-intervalles. Sur chaque petit sous-intervalle, les mauvais cas pour la droite sont calculés à l'aide d'une variante de l'algorithme d'Euclide. Le coût de l'algorithme vient en fait du nombre de sous-intervalles à considérer, tous les autres coûts étant négligeables. La complexité est ainsi de l'ordre de $\frac{N}{T}$.

Si les sous-intervalles sont suffisamment nombreux, les approximations sont de qualité suffisante pour que les mauvais cas de f soient tous des mauvais cas des droites (en affaiblissant un peu la borne m). Plus précisément, supposons que $f(x) = f(x_0) + f'(x_0)(x - x_0) + O((x - x_0)^2)$ pour x proche de x_0. Puisque nous négligeons les termes d'ordre 2 et plus dans le développement de Taylor, nous devons avoir, d'après la formule de Taylor-Lagrange, $N|f''(x_0)|(x - x_0)^2 \lesssim 1/M$, ce qui est impliqué par :

$$K \cdot \frac{T^2}{N} \lesssim \frac{1}{M}.$$

Ainsi, l'erreur d'approximation linéaire ne sera pas beaucoup plus grande que la distance $\frac{1}{M}$ (sinon, il y aurait beaucoup trop de faux positifs parmi les mauvais cas trouvés pour les droites). Comme $\max_{x\in[1/2,1]} |f''(x)| \leqslant K$ et K est une constante, on trouve donc la condition $\frac{T^2}{N} = O\left(\frac{1}{M}\right)$. On appellera plus tard cette condition la condition de Taylor.

Par ailleurs, si T est grand par rapport à M, alors il y a trop de solutions dans un sous-intervalle donné pour qu'on puisse les trouver en temps polynomial (heuristiquement, il y en a $\frac{T}{M}$). On obtient ainsi une deuxième condition $T = O(M)$, que l'on généralisera sous le nom de condition de Coppersmith (car on utilisera la méthode de Coppersmith

décrite à la section précédente). Si l'on considère les deux conditions simultanément, on s'aperçoit que le choix de paramètres optimal est

$$T \approx M \approx N^{1/3},$$

ce qui permet d'aboutir à une complexité heuristique de l'ordre de $N^{2/3+\varepsilon}$ pour l'intervalle complet $[1/2, 1[$.

En pratique, l'algorithme de Lefèvre est coûteux mais reste réaliste en double précision étendue (c'est-à-dire $n = 64$).

1.2.3 Généralisation de la méthode de Lefèvre.

Le principe de notre généralisation de la méthode de Lefèvre est de considérer des approximations polynomiales de plus grand degré, à la place d'approximations par des droites. Il faut donc être capable de trouver les mauvais cas de polynômes. Pour cela, nous utilisons la méthode de Coppersmith pour trouver les petites racines de polynômes modulaires bivariés (décrite à la section 1.1). Deux phénomènes se compensent vis-à-vis de la complexité de l'algorithme : la méthode de Coppersmith est plus coûteuse que la variante de l'algorithme d'Euclide utilisée par Lefèvre (cela nous donnera la borne de Coppersmith), mais le fait de prendre des polynômes de plus grand degré permet de diminuer significativement le nombre de sous-intervalles à considérer (cela nous donnera la borne de Taylor). La méthode fait intervenir divers paramètres : $\frac{2T}{N}$ est la longueur des sous-intervalles, d est le degré des polynômes approximateurs, et α est l'ordre de la méthode de Coppersmith. L'algorithme est décrit aux figures 1.2 et 1.3.

L'algorithme de la figure 1.2 consiste à subdiviser l'intervalle $[1/2, 1[$ en $\approx \frac{N_1}{4T}$ sous-intervalles de longueur $\approx \frac{2T}{N_1}$. Pour chaque intervalle, on approche la fonction f par un polynôme de degré d. Bien que le développement de Taylor puisse ne pas être le meilleur choix du point de vue de la qualité de l'approximation, il a l'avantage d'être très rapide à calculer, mais on pourrait par exemple choisir de prendre une approximation minimax, ou encore adapter la méthode à base de « tables de différences » de Lefèvre [107]. Ensuite, on cherche à résoudre l'équation (1.1) pour le polynôme approximateur et $c = 0$, ce que l'on fait en appelant l'algorithme de la figure 1.3, en prenant en compte l'erreur d'approximation ε dans la variable M'. Ce dernier algorithme peut échouer — s'il échoue, il le fait en temps fini (il ne peut pas boucler). Pour parer à cela, on le rappelle en divisant la longueur de l'intervalle par deux, jusqu'à ce que soit l'algorithme de la figure 1.3 réussisse, soit on obtienne un intervalle avec un seul nombre représentable, auquel cas on teste l'élément unique de l'intervalle. Cette procédure permet de garantir que l'algorithme est correct : seule sa borne de complexité sera heuristique.

L'algorithme de la figure 1.3 cherche à trouver les solutions $x \in [|-T, T|]$ de l'équation $|P(x) \bmod 1| \leqslant \frac{1}{M}$. Pour cela, il utilise la méthode de Coppersmith pour trouver des petites racines de polynômes modulaires bivariés. Les petites racines contiendront les couples (x, y) où $x \in [|-T, T|]$ est solution de l'équation à résoudre et $y = P(x) \bmod 1$, avec $|y| \leqslant \frac{1}{M}$. Le module est 1. Cet algorithme fait intervenir le paramètre α, qui détermine la plus grande puissance du polynôme $Q(x, y) = P(x) + y$ qui sera utilisée pour

Entrée : Quatre réels N_1, N_2, M et c avec un nombre fini de bits après la virgule, et trois paramètres entiers T, d et α.
Sortie : Tous les $x \in [|N_1/2, N_1 - 1|]$ solutions de l'équation (1.1).

1. $t_0 := \lceil \frac{N_1}{2} \rceil$, $T' := T$, $Res := \emptyset$.
2. Tant que $t_0 \leqslant N_1 - 1$, faire
3. Si $T' = 0$,
4. Ajouter $x = t_0$ dans *Res* s'il est solution de l'équation (1.1),
5. $t_0 := t_0 + 1, T' := T$.
6. Sinon,
7. Si $t_0 + 2T' \geqslant N_1 - 1$, $T' := \lfloor \frac{N_1 - t_0 - 1}{2} \rfloor$.
8. $t_m := t_0 + T', P(x) := c + f\left(\frac{t_m}{N_1}\right) + f'\left(\frac{t_m}{N_1}\right) x + \ldots + \frac{1}{d!} f^{(d)}\left(\frac{t_m}{N_1}\right) x^d$.
9. $\varepsilon := \left(\max_{x \in [1/2, 1[} |f^{(d+1)}(x)|\right) \cdot \frac{N_2}{(d+1)!} \left(\frac{T'}{N_1}\right)^{d+1}$, $M' := \frac{1}{\varepsilon + \frac{1}{M}}$.
10. Utiliser l'algorithme de la figure 1.3 pour trouver les solutions de l'équation $\left| N_2 P\left(\frac{x}{N_1}\right) \bmod 1 \right| \leqslant \frac{1}{M'}$, avec $x \in [|-T', T'|]$.
11. Si l'algorithme de la figure 1.3 échoue, $T' := \lfloor T'/2 \rfloor$,
12. Sinon,
13. ajouter à *Res* les solutions trouvées qui satisfont l'équation (1.1),
14. $t_0 := t_0 + 2T' + 1, T' := T$.
15. Renvoyer *Res*.

FIG. 1.2 – L'algorithme de résolution de l'équation (1.1).

Entrée : Deux réels N_1 et M avec un nombre fini de bits après la virgule, un polynôme $P(x)$ et trois paramètres entiers T, d et α.
Sortie : Tous les $x \in \frac{1}{N_1}[|-T, T|]$ tels que $|P(x) \bmod 1| \leqslant \frac{1}{M}$.

1. $n := \frac{(\alpha+1)(d\alpha+2)}{2}$.
2. Soit $\{e_1, \ldots, e_n\} = \{x^i y^j, 0 \leqslant i + dj \leqslant d\alpha\}$.
3. Soit $\{g_1, \ldots, g_n\} = \{x^i (P(x) + y)^j, 0 \leqslant i + dj \leqslant d\alpha\}$.
4. Créer la matrice B de dimensions $n \times n$ pour laquelle $B_{k,l}$ est le coefficient du monôme e_l dans le polynôme $g_k\left(Tx, \frac{1}{M} y\right)$.
5. LLL-réduire le réseau engendré par les lignes de la matrice B.
6. Soient $\mathbf{b}_1, \mathbf{b}_2$ deux plus courts vecteurs pour la norme ℓ_1.
7. Si $\|\mathbf{b}_1\|_1 \geqslant 1$ ou $\|\mathbf{b}_2\|_1 \geqslant 1$, renvoyer ÉCHEC.
8. Soient $Q_1(x, y), Q_2(x, y)$ les polynômes correspondant à $\mathbf{b}_1$ et $\mathbf{b}_2$. $R(x) := \mathrm{Res}_y(Q_1(x, y), Q_2(x, y))$. Si $R(x) = 0$, renvoyer ÉCHEC.
9. Renvoyer tous les $x \in \frac{1}{N_1}[|-T, T|]$ racines de $R(x)$.

FIG. 1.3 – L'algorithme de recherche de mauvais cas du polynôme $P(x)$.

construire le réseau euclidien. À l'étape 5, nous réduisons ce réseau. Pour effectuer cela, on peut utiliser l'algorithme LLL [112], ou n'importe laquelle de ses variantes, par exemple celle décrite au chapitre II-3.

L'algorithme peut échouer de deux façons différentes : la première source d'échec est que la réduction de réseau ne renvoie pas assez de vecteurs suffisamment courts, et la seconde est que les deux polynômes correspondants aux deux vecteurs courts trouvés ne sont pas algébriquement indépendants. En faisant croître arbitrairement la dimension et en fixant les paramètres convenablement, on peut faire en sorte de ne jamais être dans la première situation. En fait, à l'aide du théorème 3 (page 15), nous fixerons toujours les paramètres de telle sorte que l'on obtienne au moins un vecteur suffisamment court. Cela ne garantit pas qu'il y aura deux vecteurs assez courts. Pour ce deuxième point, si l'on fait croître arbitrairement la dimension du réseau, la même condition pour le premier vecteur (à des facteurs négligeables près) assurera aussi que le deuxième vecteur renvoyé par l'algorithme LLL sera suffisamment court. En dimension fixée, deux choix sont possibles : soit renforcer la condition sur le premier vecteur pour qu'on soit certain d'avoir deux vecteurs suffisamment courts, soit supposer que l'algorithme LLL renvoie des vecteurs de longueurs similaires quand on lui donne en entrée une base « aléatoire », ce qui s'observe assez bien expérimentalement[28]. Nous avons choisi la deuxième solution, car nous sommes obligés, dans l'état actuel des connaissances sur la méthode de Coppersmith multivariée, de faire des hypothèses heuristiques pour nous prémunir contre la seconde source d'échec. En effet, on ne sait pas comment écarter la possibilité que le résultant calculé soit nul. Ainsi, les analyses de complexité que nous effectuerons seront correctes sous l'hypothèse qu'à l'étape 8, le résultant calculé n'est jamais nul, ou qu'il l'est très rarement, et que, dans le cas où la dimension est fixe, les deux premiers vecteurs renvoyés par l'algorithme LLL à l'étape 5 sont de longueurs similaires.

À l'étape 6, la construction des polynômes Q_1 et Q_2 à partir des vecteurs $\mathbf{b}_1$ et $\mathbf{b}_2$ est simple à effectuer. Pour une colonne qui correspond à un monôme $x^i y^j$, on multiplie le coefficient par M^j et on le divise par T^i. On reconstruit ensuite le polynôme en lisant ses coefficients et en utilisant la correspondance entre coordonnées et monômes donnée par les e_k. À l'étape 9, il s'agit de trouver les racines entières d'un polynôme à une seule variable. Cela peut être réalisé par exemple en les cherchant modulo p, où p est un nombre premier supérieur à $2T$.

Dans les algorithmes des figures 1.2 et 1.3, les calculs sont décrits avec des nombres réels. Il est possible de les remplacer par des nombres réels avec un nombre fini de bits non nuls après la virgule. En effet, si l'on remplace le polynôme P de l'étape 8 de l'algorithme de la figure 1.2 par un polynôme $\tilde{P}$ dont les coefficients approchent ceux de P, il suffit que pour tout $x \in \left[-\frac{T'}{N_1}, \frac{T'}{N_1}\right]$, on ait $|N_2 \cdot [P(x) - \tilde{P}(x)]| \lesssim \frac{1}{M'}$. On peut donc

[28]Pour un réseau « aléatoire » L, les minimas sont tous du même ordre de grandeur à des facteurs négligeables près : $(\det L)^{\frac{1}{\dim L}}$. Il en est donc de même pour les vecteurs renvoyés par l'algorithme LLL. Nous ne savons pas a priori si les bases générées sont aléatoires dans un sens qui serait à déterminer, mais l'heuristique sur les longueurs des vecteurs renvoyés par l'algorithme LLL semble vérifiée expérimentalement dans notre situation, pour des fonctions f transcendantes.

par exemple arrondir tous les coefficients de $P(x)$ vers un réel le plus proche qui n'a que $\lg M' + \lg N_2 + \lg(d+1)$ bits après la virgule, et remplacer M' par $\lfloor M'/2 \rfloor$. Si l'on veut ne manipuler que des entiers, il convient alors de multiplier tous les coefficients de $\tilde{P}$ par une puissance de 2 adéquate. À l'étape 3 de l'algorithme de la figure 1.3, on remplacera les polynômes g_k par les polynômes $x^i \left(M' N_2 N_1^d P\left(\frac{x}{N_1}\right) + y\right)^j (M' N_2 N_1^d)^{\alpha - j}$ pour $0 \leqslant i + dj \leqslant d\alpha$, modulo $M' N_2 N_1^d$, et avec $|x| \leqslant T$ et $|y| \leqslant N_2 N_1^d$. À l'étape 7, au lieu de comparer les normes avec 1, il faudra alors les comparer avec $(M' N_1)^\alpha$. Nous garderons la description avec des nombres réels, pour ne pas introduire de difficultés techniques superflues.

Théorème 24. *Pour tout choix de paramètres T, d et α, l'algorithme de la figure 1.2 est correct. Cela signifie qu'il renvoie bien toutes les solutions de l'équation (1.1) dans l'intervalle $[1/2, 1[$, pour les quatre réels N_1, N_2, M et c.*

Démonstration. Comme on teste toutes les solutions renvoyées, il suffit de vérifier que toutes les solutions de l'équation (1.1) sont bien calculées. Soit $x \in [|N_1/2, N_1 - 1|]$ une telle solution. L'entier x fait partie de l'un des sous-intervalles considérés. Soit $[|t_m - T', t_m + T'|]$ l'intervalle contenant x qui ne fait pas échouer l'algorithme de la figure 1.3 et qui comporte plus d'un élément. On a, d'après le théorème de Taylor-Lagrange :

$$\begin{aligned}
\left| N_2 P\left(\frac{x}{N_1}\right) \mod 1 \right| &\leqslant \left| N_2 P\left(\frac{x}{N_1}\right) - N_2 f\left(\frac{x}{N_1}\right) - c \right| + \left| N_2 f\left(\frac{x}{N_1}\right) - c \mod 1 \right| \\
&\leqslant \varepsilon + \frac{1}{M} \\
&\leqslant \frac{1}{M'}.
\end{aligned}$$

Ainsi, il nous suffit de prouver que l'algorithme de la figure 1.3 fonctionne correctement lorsqu'il ne renvoie pas ÉCHEC. Soit $y = -N_2 P\left(\frac{x}{N_1}\right) \mod 1$. Le couple (x, y) est une racine du polynôme $N_2 P\left(\frac{x}{N_1}\right) + y$, modulo 1. Par conséquent, il est aussi racine de tous les polynômes g_k modulo 1, et donc de leurs combinaisons linéaires entières. En particulier, il est racine des polynômes Q_1 et Q_2 modulo 1. Par ailleurs, on a :

$$|x| \leqslant T' \quad \text{et} \quad |y| \leqslant \frac{1}{M'}.$$

Comme les vecteurs $\mathbf{b}_1$ et $\mathbf{b}_2$ sont de normes ℓ_1 plus petites que 1, on en déduit que le couple (x, y) est racine des polynômes Q_1 et Q_2 sur $\mathbb{Z}$. Ainsi, si R n'est pas nul, alors x est une racine entière de R sur $\mathbb{Z}$, et sera trouvé à l'étape 9 de l'algorithme de la figure 1.3. □

1.2.4 Principaux résultats.

Tous nos résultats de complexité sont heuristiques, et sont énoncés sous l'hypothèse où l'algorithme de la figure 1.3 n'échoue jamais à cause d'un résultant nul. Des hypothèses similaires sont usuelles dans le domaine de la cryptographie [26, 123, 124]. Dans notre

contexte, cette heuristique s'avère valide très souvent dans le cas de fonctions transcendantes, mais elle peut échouer complètement pour certaines fonctions algébriques. Nous évoquerons ce problème à la section 1.7. Le résultat ci-dessous est prouvé à la section 1.5.

Théorème 25. *Soient M, N_1, N_2 et c quatre réels avec $MN_2 \geqslant N_1$. On définit $m = \lg M$, $n_1 = \lg N_1$ et $n_2 = \lg N_2$. Soient α et d deux entiers tels que $\alpha \geqslant \log d$. Supposons que l'on fasse tendre α vers $+\infty$ lorsque le produit MN_1N_2 croît. Supposons enfin qu'aucun résultant calculé ne soit nul. Alors l'algorithme décrit aux figures 1.2 et 1.3 résout l'équation (1.1) en temps $\mathcal{P}\mathrm{ol}(n_1, n_2, m, d, \alpha)\frac{N_1}{T}$, tant que :*

$$t(1+\varepsilon_1) \leqslant \min\left(n_1 - \frac{m+n_2+O(1)}{d+1}, n_1 - \frac{(n_1+n_2)^2}{4(m+n_2)} + \varepsilon_2\right),$$

quand α tend vers $+\infty$, avec $t = \lg T$ et :

$$\begin{aligned}
\varepsilon_1 &= \frac{1}{m+n_2}O(1) + O\left(\frac{1}{\alpha}\right), \\
\varepsilon_2 &= \frac{1}{m+n_2}O(\alpha^2) + \frac{n_1}{m+n_2}O(\alpha) + (n_1+n_2)O\left(\frac{1}{\alpha}\right) + (m+n_2)O\left(\frac{1}{\alpha^2}\right).
\end{aligned}$$

Dans l'énoncé précédent, les quantités $\varepsilon_1, \varepsilon_2$ et « $O(1)$ » sont vouées à être négligeables quand les différents paramètres augmentent correctement les uns relativement aux autres. Si on les néglige, le théorème donne deux bornes supérieures à T, c'est-à-dire à la taille des sous-intervalles que l'on sait traiter en une seule réduction de réseaux, pour un jeu de paramètre donné. Puisque la complexité est polynomiale quand on se restreint à un sous-intervalle (et croît polynomialement avec les paramètres d, m, α), le but est de choisir les paramètres de telle sorte que l'on puisse prendre T le plus grand possible. La première borne supérieure de T permet de garantir l'acuité des approximations polynomiales, alors que la seconde assure que suffisamment de vecteurs courts sont renvoyés lors de l'utilisation de la méthode de Coppersmith.

Dans le cas de la recherche des mauvais cas pour l'arrondi des fonctions mathématiques, nous sommes limités par la condition $M = O(2^n)$, et nous obtenons ainsi le résultat suivant :

Théorème 26. *Supposons que $N_1 = 2^n$ et $N_2 = 2^{n+e}$ avec $e \in \{0, 1\}$. Soit $\varepsilon > 0$. Supposons que $d = 3, M = 2^n$, et $t = \frac{n}{2}(1-\varepsilon)$. Supposons enfin qu'aucun résultant calculé ne soit nul. Alors on peut choisir α de telle sorte que l'algorithme décrit aux figures 1.2 et 1.3 résout l'équation (1.1) en temps $\mathcal{P}\mathrm{ol}(n)2^{\frac{n}{2}(1+\varepsilon)}$.*

Démonstration. Si l'on fait tendre α vers $+\infty$, on a :

$$\begin{aligned}
\varepsilon_1 &= \frac{1}{n}O(1) + O\left(\frac{1}{\alpha}\right), \\
\varepsilon_2 &= \frac{1}{n}O(\alpha^2) + O(\alpha) + nO\left(\frac{1}{\alpha}\right).
\end{aligned}$$

On fixe α suffisamment grand pour que les « $O\left(\frac{1}{\alpha}\right)$ » de ε_1 et ε_2 soient plus petits que ε en valeur absolue. On obtient ainsi, pour n tendant vers $+\infty$:

$$\begin{aligned} |\varepsilon_1| &\leqslant O\left(\frac{1}{n}\right) + \varepsilon, \\ |\varepsilon_2| &\leqslant O(1) + n\varepsilon. \end{aligned}$$

Finalement, l'équation à vérifier devient :

$$t\left(1 + O\left(\frac{1}{n}\right) + \varepsilon\right) \leqslant \min\left(\frac{n}{2} - O(1), \frac{n}{2}(1-2\varepsilon) + O(1)\right),$$

pour n tendant vers $+\infty$. Quand t et n sont plus grands que certaines constantes, alors en prenant $t = \frac{n}{2}(1-4\varepsilon)$, on obtient le résultat escompté. □

Dans le cas du générateur pseudo-aléatoire de la définition 31, on obtient le résultat suivant :

Théorème 27. *Soient $N_1 = 2^{n_1}$ et $N_2 = 2^{n_2}$. Supposons que $M = 2^{(n_1+n_2)^2}$. Nous choisissons $d = \Theta((n_1+n_2)^2), \alpha = \Theta(n_1+n_2)$, et $T = \frac{N_1}{c}$ pour une certaine constante c. Alors, sous l'hypothèse qu'aucun résultant calculé ne soit nul, l'algorithme décrit aux figures 1.2 et 1.3 calcule les solutions de l'équation (1.1) en temps polynomial en n_1 et n_2.*

Démonstration. Avec ce choix de paramètres, on a, pour $n_1 + n_2$ tendant vers $+\infty$:

$$\begin{aligned} \varepsilon_1 &= O\left(\frac{1}{(n_1+n_2)^2}\right) + O\left(\frac{1}{n_1+n_2}\right) = O\left(\frac{1}{(n_1+n_2)^2}\right) \\ \varepsilon_2 &= O(1) + O\left(\frac{1}{n_1+n_2}\right) + O(1) + O(1) = O(1) \end{aligned}$$

L'équation que $t = \lg T$ doit satisfaire devient donc :

$$t\left(1 + O\left(\frac{1}{(n_1+n_2)^2}\right)\right) \leqslant n_1 - O(1).$$

Ceci nous donne le résultat annoncé. □

On peut remarquer que même si la complexité annoncée dans le précédent résultat est polynomiale, le calcul reste extrêmement coûteux. Si nous prenons l'algorithme décrit au chapitre II-3 pour réaliser la réduction de réseau, la complexité « $O(d^4(d+n)(d+\log B)\log B)$ » devient ici :

$$O(((n_1+n_2)^2)^4 \cdot (n_1+n_2)^4 \cdot ((n_1+n_2)^4)^2) = O((n_1+n_2)^{20}).$$

Pour aboutir à cette borne, nous avons borné la dimension du réseau par $O(\alpha^2)$ car on peut se contenter de réduire le réseau engendré par les lignes pour lesquelles $0 \leqslant i+j \leqslant \alpha$ (voir la section 1.5). Par contre, cela ne diminue pas le nombre de colonnes, bien qu'il soit vraisemblable que l'on puisse en supprimer. Enfin, pour la borne sur les longueurs des vecteurs au début de la réduction, nous prenons un réseau dont les entrées sont entières, comme nous l'avons expliqué plus haut (à la fin du paragraphe 1.2.3).

On peut noter qu'en fait $O\left(\frac{(n_1+n_2)^2}{\log n_1}\right)$ bits suffisent au lieu de $(n_1+n_2)^2$ pour aboutir à une complexité polynomiale. Plus généralement, quand on considère une fonction de ν variables, il est semble que la borne devienne $\Omega((n_1+n_2)^{\nu+1})$ bits (voir la section 1.6).

1.3 Une première analyse de l'algorithme.

Dans cette section, nous donnons une première analyse naïve de notre généralisation de l'algorithme de Lefèvre, qui ne permet pas d'obtenir les bornes de complexité annoncées à la section 1.2, mais permet d'introduire simplement les idées que nous utiliserons aux sections 1.4 et 1.5 pour améliorer ces bornes de complexité. Dans cette optique nous considérons en détail dans la présente section le choix de paramètres $d = \alpha = 2$.

1.3.1 Deux bornes dans l'analyse de complexité.

À l'étape 8 de l'algorithme de la figure 1.2, nous approchons la fonction f par un polynôme de degré d. Ce polynôme est le développement de Taylor à l'ordre d de la fonction f au point $\frac{t_m}{N_1}$. Par conséquent, pour tout x dans cet intervalle, grâce à la formule de Taylor-Lagrange, on a la borne d'erreur :

$$|f(x) - P(x)| \leqslant \left(\max_{y \in [1/2,1[} |f^{(d+1)}(y)| \right) \cdot \frac{T^{d+1}}{(d+1)! N_1^{d+1}} \leqslant K \left(\frac{T}{N_1} \right)^{d+1}.$$

Pour que la quantité M' (calculée à l'étape 9 de l'algorithme de la figure 1.2) ne soit pas beaucoup plus petite que la quantité M, il suffit d'avoir :

$$K N_2 \left(\frac{T}{N_1} \right)^{d+1} \leqslant \frac{1}{M}. \tag{1.2}$$

Nous appelons cette condition la condition de Taylor. Elle implique que $M' \in [M/2, M]$. La raison pour laquelle nous prenons M' proche de M est que la borne sur T liée à l'étape de réduction de réseau augmente avec M'. Ainsi, si M' est petit, la borne sur T sera plus petite, et il y aura plus de sous-intervalles à considérer, ce qui augmentera le coût de la méthode. On peut interpréter cela de la manière suivante. Avoir un grand ε revient à prendre un petit M, à cause de la somme $\varepsilon + \frac{1}{M}$. Par ailleurs, si M est petit, l'équation (1.1) impose moins de contraintes, et donc elle devient plus difficile à résoudre. Moralement, quand M est fixé, mieux vaut prendre M' proche de M, sinon cela revient à résoudre l'équation (1.1) avec un M plus petit.

La deuxième borne vient de la réduction de réseau. Si on lui donne en entrée une base d'un réseau L, l'algorithme LLL renvoie une base réduite, dont le premier vecteur satisfait $\|\mathbf{b}_1\| \leqslant 2^{\dim L} (\det L)^{\frac{1}{\dim L}}$. Ainsi, pour être certain que la réduction de réseau de l'étape 5 de l'algorithme de la figure 1.3 renverra au moins un vecteur suffisamment court pour la norme ℓ_1, il faut s'assurer que $\sqrt{\dim L} \cdot 2^{\dim L} (\det L)^{\frac{1}{\dim L}} < 1$. En réalité, on veut s'assurer que l'algorithme LLL renverra au moins deux vecteurs de normes ℓ_1 plus petites que 1. Il convient donc en fait d'utiliser l'inégalité suivante, qui provient du théorème 3, page 15, et du deuxième théorème de Minkowski, page 13 :

$$\|\mathbf{b}_2\| \leqslant 2^{\dim L} \lambda_2(L) \leqslant 2^{\dim L} \sqrt{\dim L} \left(\frac{\det L}{\lambda_1(L)} \right)^{\frac{1}{\dim L - 1}}.$$

Par conséquent, on cherchera à satisfaire l'équation suivante, que nous appelons condition de Minkowski :

$$\sqrt{\dim L} \cdot 2^{\dim L} \left(\frac{\det L}{\lambda_1(L)} \right)^{\frac{1}{\dim L - 1}} < 1. \tag{1.3}$$

Comme $\dim L$ va tendre vers $+\infty$, le terme « $\sqrt{\dim L}$ » deviendra négligeable, de même que le « -1 » en exposant[29]. Si l'équation ci-dessus est satisfaite, la seule cause d'échec possible dans l'algorithme de la figure 1.3 est la nullité du résultant calculé à l'étape 8.

1.3.2 Première analyse de complexité.

La condition de Minkowski est relativement simple à expliciter. En effet, nous disposons d'une base triangulaire du réseau L à réduire, ce qui rend aisé les calculs de sa dimension et de son volume, ainsi que la minoration de son premier minimum. La forme triangulaire de la base est apparente si l'on ordonne les monômes $(x^i y^j)_{0 \leqslant i+dj \leqslant d\alpha}$ par valeur de $i + dj$ croissante, et par valeur de j croissante en cas d'égalité. On ordonne les lignes de la même manière. Par exemple, pour $d = 2$ et $\alpha = 3$, nous avons une matrice de la forme :

$$\begin{pmatrix}
1 & x & x^2 & y & x^3 & xy & x^4 & x^2y & y^2 & x^5 & x^3y & xy^2 & x^6 & x^4y & x^2y^2 & y^3 \\
\hline
1 & & & & & & & & & & & & & & & \\
 & T & & & & & & & & & & & & & & \\
 & & T^2 & & & & & & & & & & & & & \\
- & - & - & \frac{1}{M'} & & & & & & & & & & & & \\
 & & & & T^3 & & & & & & & & & & & \\
 & - & - & & - & \frac{T}{M'} & & & & & & & & & & \\
 & & & & & & T^4 & & & & & & & & & \\
 & & - & & - & & - & \frac{T^2}{M'} & & & & & & & & \\
- & - & - & - & - & - & - & - & \frac{1}{(M')^2} & & & & & & & \\
 & & & & & & & & & T^5 & & & & & & \\
 & & & & - & & - & & & - & \frac{T^3}{M'} & & & & & \\
 & - & - & & - & - & - & - & & - & - & \frac{T}{(M')^2} & & & & \\
 & & & & & & & & & & & & T^6 & & & \\
 & & & & & & - & & & - & & & - & \frac{T^4}{M'} & & \\
 & & - & & - & & - & - & & - & - & & - & - & \frac{T^2}{(M')^2} & \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & \frac{1}{(M')^3}
\end{pmatrix}.$$

Les entrées de la matrice avec des « $-$ » sont celles qui peuvent être non nulles.

On obtient assez aisément la dimension du réseau :

$$\dim L = \frac{(\alpha+1)(d\alpha+2)}{2} = \frac{d}{2}(\alpha^2 + O(\alpha)).$$

[29]Dans la suite, on utilisera implicitement cet argument sur le deuxième vecteur renvoyé par l'algorithme LLL. Il se généralise sans difficulté à un nombre fini quelconque de vecteurs courts, quand la dimension du réseau tend vers $+\infty$.

De même, son volume est facile à calculer puisqu'il suffit de prendre le produit des termes diagonaux de la matrice :

$$\det L = T^{\frac{d^2}{6}(\alpha^3+O(\alpha^2))}(M')^{-\frac{d}{6}(\alpha^3+O(\alpha^2))}.$$

On minore $\lambda_1(L)$ en utilisant le fait que pour toute base $\mathbf{b}_1, \ldots, \mathbf{b}_n$, on a $\lambda_1(L) \geqslant \min_{i \leqslant n} \|\mathbf{b}_i^*\|$ (voir le chapitre I-1). Ici, les $\|\mathbf{b}_i^*\|$ sont simplement les termes diagonaux de la matrice. Par conséquent, on obtient :

$$\lambda_1(L) \geqslant (M')^{-\alpha}.$$

Cette étude du réseau permet d'expliciter la condition de Coppersmith, c'est-à-dire l'équation (1.3) :

$$2^{\frac{d}{2}(\alpha^2+O(\alpha))}T^{\frac{d}{3}(\alpha+O(1))}M^{-\frac{1}{3}(\alpha+O(1))} < 1,$$

où l'on a utilisé les relations $M' \geqslant M/2$ et $\sqrt{\dim L} = 2^{\frac{d}{2}O(\alpha)}$.

Soit $\varepsilon > 0$. On choisit α suffisamment grand pour que les deux termes « $O(1)$ » de l'équation ci-dessus satisfassent $O(1) \leqslant \frac{\varepsilon}{2}\alpha$. L'équation (1.3) est donc impliquée par :

$$O(1)T^{\frac{d}{3}\left(1+\frac{\varepsilon}{2}\right)}M^{-\frac{1}{3}\left(1+\frac{\varepsilon}{2}\right)} < 1,$$

Par conséquent, si $T^{d+\varepsilon} \leqslant M^{1-\varepsilon}$ et si T et M sont suffisamment grands (plus grands qu'une certaine constante), alors la condition de Coppersmith est satisfaite. On en déduit que si la condition suivante est satisfaite, il existe un algorithme de complexité en temps heuristique $\mathcal{P}\mathrm{ol}(n_1, n_2, m, d)\frac{N_1}{T}$ pour résoudre l'équation (1.1) :

$$T \leqslant \min\left(\frac{N_1}{(MN_2K)^{\frac{1}{d+1}}}, M^{\frac{1-\varepsilon}{d+\varepsilon}}\right).$$

Si on peut choisir M, alors, en prenant $M = \left(\frac{N_1^{d+1}}{N_2K}\right)^{\frac{d}{2d+1}+\varepsilon}$ et $T = \left(\frac{N_1^{d+1}}{N_2K}\right)^{\frac{1}{2d+1}-\varepsilon}$, on aboutit à un algorithme de complexité :

$$\mathcal{P}\mathrm{ol}(n_1, n_2, d)N_1^{\frac{d}{2d+1}+\varepsilon}(N_2K)^{\frac{1}{2d+1}+\varepsilon}.$$

1.3.3 Le cas $d = \alpha = 2$.

Dans ce paragraphe, nous analysons en détail le choix de paramètres $d = \alpha = 2$, car il s'agit des paramètres les plus simples qui permettent d'améliorer l'algorithme de Lefèvre. Ces paramètres sont aussi particulièrement intéressants en pratique pour la recherche des mauvais cas pour l'arrondi de fonctions.

Commençons par donner la forme explicite du réseau à réduire. On écrit $P(x) = a_0 + a_1x + a_2x^2$ (le polynôme P correspond au début du développement de Taylor de la fonction f en $\frac{t_m}{N_1}$). Alors le réseau à réduire est celui engendré par les lignes de la matrice B suivante :

$$\begin{pmatrix}
1 & T & T^2 & & T^3 & & T^4 & & \\
N_2a_0 & N_2a_1\frac{T}{N_1} & N_2a_2\frac{T^2}{N_1^2} & \frac{1}{M'} & & & & & \\
 & N_2a_0T & N_2a_1\frac{T^2}{N_1} & & N_2a_2\frac{T^3}{N_1^2} & \frac{T}{M'} & & & \\
 & & N_2a_0T^2 & & N_2a_1\frac{T^3}{N_1} & & N_2a_2\frac{T^4}{N_1^2} & \frac{T^2}{M'} & \\
N_2^2a_0^2 & 2N_2^2a_0a_1\frac{T}{N_1} & N_2^2(2a_0a_2+a_1^2)\frac{T^2}{N_1^2} & 2N_2a_0\frac{1}{M'} & 2N_2^2a_1a_2\frac{T^3}{N_1^3} & 2N_2a_1\frac{T}{N_1M'} & N_2^2a_2^2\frac{T^4}{N_1^4} & 2N_2a_2\frac{T^2}{N_1^2M'} & \frac{1}{(M')^2}
\end{pmatrix}.$$

La dimension de ce réseau est 9, et son déterminant est $\frac{T^{13}}{(M')^5}$. Ainsi, pour obtenir un vecteur de norme ℓ_1 plus petite que 1, il suffit que $T^{13} = O(M^5)$. En général, les vecteurs renvoyés par l'algorithme LLL sont de longueurs similaires, de telle sorte que si $T^{13} = O(M^5)$, alors on s'attend à obtenir deux vecteurs suffisamment courts. Ainsi, en utilisant à la fois les bornes de Taylor et de Coppersmith, on obtient la condition :

$$T = O\min\left(\frac{N_1}{(MN_2)^{1/3}}, M^{5/13}\right).$$

Le choix optimal pour M est donc $M \approx \frac{N_1^{39/28}}{N_2^{13/28}}$, ce qui permet de choisir $T \approx \frac{N_1^{15/28}}{N_2^{5/28}}$. Dans le cas où $N_1 = N_2 = N$, ces équations se simplifient en $M \approx N^{13/14}$ et $T \approx N^{5/14}$.

En fait, on peut exhiber un sous-ensemble de lignes de la matrice ci-dessus pour lequel la borne de Minkowski $\lambda_1(L) \leqslant \sqrt{\dim L} \cdot (\det L)^{\frac{1}{\dim L}}$ est beaucoup plus avantageuse : le réseau que nous avons considéré contient un sous-réseau de borne de Minkowski beaucoup plus faible. Cela vient du fait que des coefficients de la matrice B sont nettement plus petits que le coefficient diagonal qui se situe sur leur colonne. On supprime désormais les lignes 5, 7 et 8, pour obtenir la matrice B' ci-dessous :

$$\begin{pmatrix}
1 & T & T^2 & & T^3 & & & & \\
N_2a_0 & N_2a_1\frac{T}{N_1} & N_2a_2\frac{T^2}{N_1^2} & \frac{1}{M'} & & & & & \\
 & N_2a_0T & N_2a_1\frac{T^2}{N_1} & & N_2a_2\frac{T^3}{N_1^2} & \frac{T}{M'} & & & \\
N_2^2a_0^2 & 2N_2^2a_0a_1\frac{T}{N_1} & N_2^2(2a_0a_2+a_1^2)\frac{T^2}{N_1^2} & 2N_2a_0\frac{1}{M'} & 2N_2^2a_1a_2\frac{T^3}{N_1^3} & 2N_2a_1\frac{T}{N_1M'} & N_2^2a_2^2\frac{T^4}{N_1^4} & 2N_2a_2\frac{T^2}{N_1^2M'} & \frac{1}{(M')^2}
\end{pmatrix}.$$

La dimension du réseau engendré par les lignes de la matrice B' est 6 car les vecteurs en question sont linéairement indépendants. Par contre, le déterminant du réseau engendré est moins facile à évaluer que précédemment car la matrice n'est plus carrée. Supposons pour simplifier que la borne de Taylor soit fine, c'est-à-dire $MN_2T^3 \approx N_1^3$. Nous rappelons que le déterminant du réseau engendré par des vecteurs $\mathbf{b}_1, \ldots, \mathbf{b}_k$ linéairement indépendants est $\det L(\mathbf{b}_1, \ldots, \mathbf{b}_k) = \prod_{i=1}^k \|\mathbf{b}_i^*\|$, où les $\mathbf{b}_i^*$ sont les orthogonalisés de Gram-Schmidt des vecteurs $\mathbf{b}_i$. Soient $\mathbf{b}_1, \ldots, \mathbf{b}_6$ les lignes de la matrice B'. On a directement les égalités :

$$\|\mathbf{b}_1^*\| = 1, \|\mathbf{b}_2^*\| = T, \|\mathbf{b}_3^*\| = T^2, \|\mathbf{b}_4^*\| = \frac{1}{M'}.$$

Ensuite, on a :

$$\|\mathbf{b}_5^*\|^2 = N_2^2a_2^2\frac{T^6}{N_1^4} + \frac{T^2}{(M')^2} \lesssim N_2^2a_2^2\frac{T^6}{N_1^4},$$

car le deuxième terme est plus petit que le premier (on s'est placé dans le cas où la borne de Taylor est fine). Pour finir, on voit que :

$$\mathbf{b}_6^* = \left(0, 0, 0, 0, 0, 0, N_2^2 a_2^2 \frac{T^4}{N_1^4}, 2N_2 a_2 \frac{T^2}{N_1^2 M'}, \frac{1}{(M')^2}\right),$$

ce qui donne $\|\mathbf{b}_6^*\| \lesssim N_2^2 a_2^2 \frac{T^4}{N_1^4}$ puisque l'on s'est placé dans le cas où la borne de Taylor est fine. En utilisant le fait que $|a_2| = O(1)$, on obtient :

$$\det L(B') \lesssim \frac{N_2^3 T^{10}}{M N_1^6}.$$

En utilisant la borne de Taylor, on obtient alors le choix de paramètres optimal $M \approx \frac{N_1^{12/13}}{N_2^{1/13}}$ et $T \approx \frac{N_1^{9/13}}{N_2^{4/13}}$. Quand on prend $N_2 = N_1 = N$, on obtient alors $M \approx N^{11/13}$ et $T \approx N^{5/13}$, ce qui est meilleur que les bornes précédentes.

En pratique, on peut réduire le « grand » réseau : le sous-réseau engendré par les lignes de la matrice B' sera lui aussi réduit, et on obtiendra un vecteur dont la longueur est celle annoncée par l'analyse fine ci-dessus. Par contre, du point de vue de l'efficacité pratique de la réduction, il vaut mieux supprimer les lignes superflues. On peut aussi supprimer les colonnes $6, 8$ et 9 de la matrice B' avant de la réduire, car ces colonnes sont des multiples d'autres colonnes dont les coefficients sont plus gros (la colonne 6 est un multiple de la colonne 5, et les colonnes 8 et 9 sont des multiples de la colonne 7). Il faudra par contre « reconstruire » ces colonnes avant de calculer les polynômes Q_1 et Q_2 à l'étape 7 de l'algorithme de la figure 1.3. Pour faire cela, il suffit d'utiliser la matrice de transformation calculée lors de la réduction de réseau de l'étape 6. Cette matrice de transformation peut être calculée très simplement à partir de la base réduite car la base de départ est triangulaire. Finalement, on peut se contenter de réduire la matrice 6×6 suivante :

$$B'' = \begin{pmatrix} 1 & & & & & \\ & T & & & & \\ & & T^2 & & & \\ N_2 a_0 & N_2 a_1 \frac{T}{N_1} & N_2 a_2 \frac{T^2}{N_1^2} & \frac{1}{M'} & & \\ & N_2 a_0 T & N_2 a_1 \frac{T^2}{N_1} & & N_2 a_2 \frac{T^3}{N_1^2} & \\ N_2^2 a_0^2 & 2N_2^2 a_0 a_1 \frac{T}{N_1} & N_2^2 (2a_0 a_2 + a_1^2) \frac{T^2}{N_1^2} & 2N_2 a_0 \frac{1}{M'} & 2N_2^2 a_1 a_2 \frac{T^3}{N_1^3} & N_2^2 a_2^2 \frac{T^4}{N_1^4} \end{pmatrix}.$$

Avec le choix de paramètres $N_1 = N_2 = N, M = N^{11/13}$ et $T = N^{5/13}$, la matrice qui précède a la forme suivante :

$$B'' \approx \begin{pmatrix} 1 & & & & & \\ & N^{5/13} & & & & \\ & & N^{10/13} & & & \\ N & N^{5/13} & N^{-3/13} & N^{-11/13} & & \\ & N^{18/13} & N^{10/13} & & N^{2/13} & \\ N^2 & N^{18/13} & N^{10/13} & N^{2/13} & N^{2/13} & N^{-6/13} \end{pmatrix}.$$

Dans cette matrice, les entrées sont des ordres de grandeur des entrées de la matrice B''.

1.4 Le cas $d = 2$.

Dans cette section, nous généralisons l'étude fine de la section précédente pour n'importe quel paramètre $\alpha \geqslant 1$, pour un degré d'approximation polynomiale égal à 2. Dans la section 1.2, nous avions choisi de prendre les polynômes $Q_{i,j}(x,y) = x^i Q(x,y)^j$ pour $0 \leqslant i + dj \leqslant d\alpha$, ce qui donne $0 \leqslant i + 2j \leqslant 2\alpha$ quand on prend $d = 2$. Ici, nous considérons la famille restreinte des $Q_{i,j}(x,y)$ pour $0 \leqslant i + j \leqslant \alpha$. Cela revient à considérer la matrice carrée triangulaire inférieure qui était associée aux $(Q_{i,j})_{0 \leqslant i+2j \leqslant 2\alpha}$ et à en conserver un certain nombre de lignes. On obtient ainsi une matrice rectangulaire avec $\frac{(\alpha+1)(\alpha+2)}{2} \approx \frac{\alpha^2}{2}$ lignes et $\approx \alpha^2$ colonnes (puisque nous conservons la dernière ligne de la matrice triangulaire inférieure de départ, toutes les colonnes de la nouvelle matrice sont en général non nulles). La difficulté est désormais d'évaluer le volume du réseau engendré par les lignes de cette nouvelle matrice : nous allons utiliser la formule $\det L = \prod_{i \leqslant \dim L} \|\mathbf{b}_i^*\|$, où les vecteurs $\mathbf{b}_i$ sont les vecteurs lignes d'une matrice qui représente une base de L. Une conséquence importante de cette formule est que pour évaluer le volume du réseau, on peut effectuer des opérations élémentaires sur les lignes et les colonnes, avec des coefficients réels.

1.4.1 Borne sur le déterminant du réseau.

Nous gardons le même ordre sur les lignes que dans la section 1.2 : nous ordonnons les lignes par valeur de $i + 2j$ croissante, puis par valeur de j croissante en cas d'égalité. Soit B la matrice correspondante. Celle-ci est de la forme :

$$B = \begin{pmatrix} B_1 & 0 \\ B_2 & B_3 \end{pmatrix},$$

où la matrice B_1 est carrée triangulaire inférieure et correspond aux lignes et aux colonnes telles que $0 \leqslant i+2j \leqslant \alpha$, et la matrice B_3 est rectangulaire et correspond aux lignes telles que $\alpha < i+2j \leqslant 2\alpha$ et $0 \leqslant i+j \leqslant \alpha$. Ainsi, on a $\det B = \det B_1 \cdot \det B_3$. Le déterminant de la matrice B_1 est simple à évaluer :

$$\begin{aligned} \det B_1 &= \prod_{0 \leqslant i+2j \leqslant \alpha} (xT)^i \cdot Q_{i,j}\left(Tx, \frac{1}{M'}y\right)[x^i y^j] \\ &= \prod_{0 \leqslant i+2j \leqslant \alpha} T^i \left(\frac{1}{M'}\right)^j \\ &= T^{\frac{\alpha^3}{12}+O(\alpha^2)}(M')^{-\frac{\alpha^3}{24}+O(\alpha^2)}. \end{aligned}$$

Il s'agit désormais de borner le déterminant de la matrice B_3. Écrivons $Q(x,y) = a_0 + a_1 x + a_2 x^2 + y$. Nous considérons les polynômes $\bar{Q}_j(x,y)$ définis par : $\bar{Q}_0 = 1$ et $\bar{Q}_j(x,y) = Q(x,y)^j - \sum_{k=0}^{j-1} \binom{j}{k}(a_0+a_1x)^{j-k}\bar{Q}_k(x,y)$ pour $j > 0$ (nous rappelons que $\binom{j}{k}$ est

le coefficient binomial pour k éléments parmi j). On vérifie aisément l'identité $\bar{Q}_j(x,y) = (a_2x^2+y)^j$. En effet, elle est vraie pour $j=0$, et pour $j>0$, on a :

$$\begin{aligned}\bar{Q}_j(x,y) &= (a_0+a_1x+a_2x^2+y)^j - \sum_{k=0}^{j-1}\binom{j}{k}(a_0+a_1x)^{j-k}(a_2x^2+y)^k \\ &= \sum_{k=0}^{j}\binom{j}{k}(a_0+a_1x)^{j-k}(a_2x^2+y)^k - \sum_{k=0}^{j-1}\binom{j}{k}(a_0+a_1x)^{j-k}(a_2x^2+y)^k \\ &= (a_2x^2+y)^j.\end{aligned}$$

On considère la matrice $\bar{B}$ qui correspond aux polynômes $x^i\bar{Q}_j(x,y)$ pour $0 \leqslant i+j \leqslant \alpha$. La raison pour laquelle nous avons introduit cette matrice est la suivante : son déterminant est plus facile à calculer que celui de la matrice B — comme on va le voir plus bas — et en fait, les deux matrices ont le même déterminant :

Lemme 40. *Avec les notations ci-dessus, nous avons :* $\det \bar{B} = \det B$.

Démonstration. Il suffit de montrer que chaque ligne de la matrice $\bar{B}$ est l'addition de la ligne correspondante de la matrice B avec une combinaison linéaire des lignes précédentes de la matrice $\bar{B}$. Nous montrons cela par récurrence sur les lignes. C'est vrai pour la première ligne. Prenons la ligne de la matrice $\bar{B}$ associée à un polynôme $x^i\bar{Q}_j(x,y)$. Par définition des polynômes $\bar{Q}_j(x,y)$ et par hypothèse de récurrence, il suffit de s'assurer que les polynômes $x^{i'}\bar{Q}_{j'}(x,y)$ sont dans la matrice $\bar{B}$ et au-dessus de la ligne associée au polynôme $x^i\bar{Q}_j(x,y)$, avec $j' \in [|0, j-1|]$ et $i' \in [|0, i+j-j'|]$. Tout d'abord, on a $0 \leqslant i'+j' \leqslant i+j \leqslant \alpha$, donc le polynôme $x^{i'}\bar{Q}_{j'}(x,y)$ est bien dans la matrice $\bar{B}$. De plus, on a $0 \leqslant i'+2j' \leqslant i+j+j' < i+2j$, ainsi le polynôme $x^{i'}\bar{Q}_{j'}(x,y)$ est « au-dessus » du polynôme $x^i\bar{Q}_j(x,y)$. □

La matrice $\bar{B}$ est de la forme suivante :

$$\bar{B} = \begin{pmatrix} \bar{B}_1 & 0 \\ 0 & \bar{B}_3 \end{pmatrix},$$

où la matrice $\bar{B}_1$ est carrée de mêmes dimension et déterminant que la matrice B_1 (on le déduit facilement de la preuve du lemme qui précède), et la matrice $\bar{B}_3$ est de la forme :

$$\bar{B}_3 = \begin{bmatrix} T^e \cdot A_1^{(\lceil \alpha/2 \rceil)} & & & & & & & & \\ & \ddots & & & & & & & \\ & & A_{\alpha-2i}^{(\alpha-i)} & & & & & & \\ & & & T \cdot A_{\alpha-2i+1}^{(\alpha-i)} & & & & & \\ & & & & \ddots & & & & \\ & & & & & A_{\alpha-2}^{(\alpha-1)} & & & \\ & & & & & & T \cdot A_{\alpha-1}^{(\alpha-1)} & & \\ & & & & & & & A_\alpha^{(\alpha)} \end{bmatrix},$$

où $A_k^{(l)}$ (pour $l \geqslant k$) est la matrice triangulaire inférieure $\left(\binom{i}{j} T^{2(l-j)} a_2^{i-j} (M')^{-j}\right)_{0 \leqslant j \leqslant i \leqslant l}$, dont les k premières lignes ont été effacées. La matrice $A_k^{(l)}$ a $l-k+1$ lignes et $l+1$ colonnes. On a $e=1$ si α est pair, et $e=0$ sinon. Pour $\alpha = 5$:

$$\bar{B}_3 = \begin{bmatrix} A_1^{(3)} & & & & & \\ & T \cdot A_2^{(3)} & & & & \\ & & A_3^{(4)} & & & \\ & & & T \cdot A_4^{(4)} & & \\ & & & & A_5^{(5)} & \end{bmatrix},$$

avec par exemple :

$$A_1^{(3)} = \begin{pmatrix} T^6 a_2 & T^4 \frac{1}{M'} & 0 & 0 \\ T^6 a_2^2 & 2T^4 a_2 \frac{1}{M'} & T^2 \left(\frac{1}{M'}\right)^2 & 0 \\ T^6 a_2^3 & 3T^4 a_2^2 \frac{1}{M'} & 3T^2 a_2 \left(\frac{1}{M'}\right)^2 & \left(\frac{1}{M'}\right)^3 \end{pmatrix},$$

$$A_3^{(4)} = \begin{pmatrix} T^8 a_2^3 & 3T^6 a_2^2 \frac{1}{M'} & 3T^4 a_2 \left(\frac{1}{M'}\right)^2 & T^2 a_2 \left(\frac{1}{M'}\right)^3 & 0 \\ T^8 a_2^4 & 4T^6 a_2^3 \frac{1}{M'} & 6T^4 a_2^2 \left(\frac{1}{M'}\right)^2 & 4T^2 a_2 \left(\frac{1}{M'}\right)^3 & \left(\frac{1}{M'}\right)^4 \end{pmatrix}.$$

Dans le lemme suivant, nous bornons le déterminant de chaque matrice $A_k^{(l)}$.

Lemme 41. *Avec les notations précédentes, nous avons :*

$$\det A_k^{(l)} \leqslant 2^{2\alpha^2} |a_2|^{k(l-k+1)} T^{(l-k+1)(l+k)} M^{-\frac{(l-k)(l-k+1)}{2}}.$$

Démonstration. Nous effectuons la suite suivante d'opérations élémentaires sur la matrice $A_k^{(l)}$: tant que cela est possible, prendre la ligne la plus basse dans la matrice qui commence par le même nombre de zéros que la précédente, et lui soustraire la ligne précédente multipliée par a_2. Cela ne change pas le déterminant de la matrice, et nous donne une nouvelle matrice dont la première ligne L_1 correspond aux coefficients du polynôme $T^{2(l-k)} \left(a_2 T^2 x^2 + \frac{1}{M'} y\right)^k$, et dont la i-ème ligne est $\frac{1}{(M'T^2)^{i-1}} \cdot L_1$, décalée de $i-1$ colonnes vers la droite. On borne le déterminant de cette nouvelle matrice par le produit des longueurs de ses lignes :

$$\begin{aligned} \det A_k^{(l)} &\leqslant \prod_{i=k}^{l} \left[\left(\frac{1}{M'T^2}\right)^{i-k} \left(\sqrt{k+1} \cdot 2^k T^{2l} |a_2|^k\right) \right] \\ &\leqslant 2^{2\alpha^2} |a_2|^{k(l-k+1)} T^{(l-k+1)(l+k)} (M')^{-\frac{(l-k)(l-k+1)}{2}}, \end{aligned}$$

ce qui donne le résultat, en bornant k et l par α, et en minorant M' par $M/2$. □

Nous bornons maintenant le déterminant de la matrice $\bar{B}_3$ en utilisant ce dernier

lemme :

$$\begin{aligned}\det \bar{B}_3 &= T^{\lfloor \alpha/2 \rfloor} \prod_{i=1}^{\alpha} \det A_i^{\left(\left\lfloor \frac{\alpha+i}{2} \right\rfloor\right)} \\ &\leqslant 2^{O(\alpha^3)} T^{\alpha/2} \prod_{i=1}^{\alpha} \left(|a_2|^{i\left(\frac{\alpha-i}{2}+1\right)} T^{\left(\frac{\alpha-i}{2}+1\right)\frac{\alpha+3i}{2}} M^{-\frac{(\alpha-i)\left(\frac{\alpha-i}{2}+1\right)}{4}} \right) \\ &\leqslant 2^{O(\alpha^3)} |a_2|^{\frac{\alpha^3}{12}+O(\alpha^2)} T^{\frac{\alpha^3}{4}+O(\alpha^2)} M^{-\frac{\alpha^3}{24}+O(\alpha^2)}.\end{aligned}$$

En utilisant la borne sur $\det B_1$, on obtient ainsi :

$$\det B \leqslant 2^{O(\alpha^3)} |a_2|^{\frac{\alpha^3}{12}+O(\alpha^2)} T^{\frac{\alpha^3}{3}+O(\alpha^2)} M^{-\frac{\alpha^3}{12}+O(\alpha^2)}.$$

1.4.2 Borne de Coppersmith et conclusion.

Nous rappelons que la dimension du réseau L considéré est $\frac{\alpha^2}{2} + O(\alpha)$. Puisque le plus court vecteur de L est de longueur au moins $(M')^{-\alpha}$ (cela provient de la relation $\lambda_1(L) \geqslant \min_i \|\mathbf{b}_i^*\|$, où les vecteurs $\mathbf{b}_i$ forment une base de L, comme expliqué au chapitre I-1), l'algorithme LLL renvoie deux vecteurs de normes strictement plus petites que 1 si $2^{\dim L}\left(\left(\frac{2}{M}\right)^{\alpha} \det L\right)^{\frac{1}{\dim L}} < (M')^{-\alpha}$. On peut récrire cette relation de la manière suivante :

$$2^{\frac{\alpha^2}{2}+O(\alpha)} |a_2|^{\frac{\alpha}{6}+O(1)} T^{\frac{2\alpha}{3}+O(1)} M^{-\frac{\alpha}{6}+O(1)} \leqslant 1.$$

Puisque $a_2 \leqslant K\frac{N_2}{N_1^2}$ (a_2 est le coefficient du monôme x^2 dans le polynôme $Q(x,y) = N_2 P\left(\frac{x}{N_1}\right) + y$), nous obtenons la borne de Coppersmith suivante :

$$2^{3\alpha+O(1)} K^{1+O\left(\frac{1}{\alpha}\right)} N_2^{1+O\left(\frac{1}{\alpha}\right)} T^{4+O\left(\frac{1}{\alpha}\right)} \leqslant M^{1+O\left(\frac{1}{\alpha}\right)} N_1^{2+O\left(\frac{1}{\alpha}\right)}.$$

Soit $\varepsilon > 0$. On choisit α suffisamment grand pour que les « $O\left(\frac{1}{\alpha}\right)$ » soient plus petits que ε. L'équation ci-dessus est impliquée par :

$$O(1) K^{1+\varepsilon} N_2^{1+\varepsilon} T^{4+\varepsilon} \leqslant M^{1-\varepsilon} N_1^{2-\varepsilon}.$$

Si $T^{4+2\varepsilon} \leqslant \left(\frac{MN_1^2}{N_2 K}\right)^{1-\varepsilon}$, et si T, M, N_1 et N_2 sont suffisamment grands, alors la condition de Coppersmith est satisfaite. Nous rappelons par ailleurs la borne de Taylor, que nous avons obtenue à la section 1.2 :

$$KMN_2T^3 \leqslant N_1^3.$$

Par conséquent, sous l'hypothèse que l'algorithme LLL renvoie deux vecteurs dont les polynômes associés sont algébriquement indépendants, on peut trouver toutes les solutions de l'équation (1.1) en temps $\mathcal{P}\mathrm{ol}(n_1, n_2, m, d)\frac{N_1}{T}$ tant que :

$$T \leqslant \min\left(\frac{N_1}{(MN_2K)^{1/3}}, \left(\frac{MN_1^2}{N_2K}\right)^{\frac{1-\varepsilon}{4+2\varepsilon}}\right).$$

En prenant ε très proche de 0, on voit que le choix optimal pour M est $M \approx \left(\frac{N_1^6}{N_2}\right)^{1/7}$, avec $T \approx \left(\frac{N_1^5}{N_2^2}\right)^{1/7}$, ce qui permet d'obtenir un algorithme de complexité heuristique $\approx (N_1 N_2)^{2/7}$. En particulier, dans le cas où $N_1 = N_2 = N$, la valeur optimale de M est $\approx N^{5/7}$, et il y a $\approx N^{4/7}$ sous-intervalles à considérer.

1.5 Le cas $d \geqslant 3$.

Nous nous intéressons désormais au cas où l'on prend un polynôme d'approximation P de degré $d \geqslant 3$. Pour analyser finement le comportement de l'algorithme dans cette situation, il nous faut développer une autre technique que celle que nous avons décrite pour le cas où $d = 2$. En effet, dans le cas où $d = 2$, le réseau avait une forme très particulière : après un peu d'algèbre linéaire, on s'aperçoit que la matrice de la base du réseau à réduire est triangulaire par blocs, et le déterminant de chaque bloc est facile à borner. Une telle structure « triangulaire par blocs » paraît difficile à faire apparaître [30] quand $d \geqslant 3$. Dans cette section, nous montrons comment analyser bien plus finement l'algorithme que nous l'avons fait à la section 1.2. Pour permettre cette analyse plus fine, nous donnons une borne générale sur les déterminants des matrices d'une certaine forme assez courante. Cette borne générale peut aussi être utilisée dans le cas où $d = 2$, mais elle donne des résultats moins intéressants que ceux que nous avons obtenus à la section 1.4. Nous commençons cette section par la description de la famille de polynômes que nous choisissons, puis nous donnons notre technique pour borner le déterminant de certaines matrices, et enfin nous appliquons cette technique à notre contexte.

1.5.1 La famille de polynômes choisie.

Comme dans le cas $d = 2$, parmi les polynômes $Q_{i,j}(x, y) = x^i Q(x, y)^j$ pour $0 \leqslant i + dj \leqslant d\alpha$, nous prenons ceux pour lesquels $0 \leqslant i + j \leqslant \alpha$. Cela fait décroître la dimension du réseau que nous considérons de $\approx \frac{d\alpha^2}{2}$ à $\frac{(\alpha+1)(\alpha+2)}{2} \approx \frac{\alpha^2}{2}$. La matrice B représentant la base correspondante a ainsi $\approx \frac{\alpha^2}{2}$ lignes et $\approx \frac{d\alpha^2}{2}$ colonnes. Nous ordonnons les lignes et les colonnes par valeur de $i + dj$ croissante, puis par valeur de j croissante lorsqu'il y a égalité. La matrice B est de la forme :

$$B = \begin{pmatrix} B_1 & 0 \\ B_2 & B_3 \end{pmatrix},$$

où la matrice B_1 est carrée triangulaire inférieure et correspond aux lignes telles que $0 \leqslant i + dj \leqslant \alpha$, et la matrice B_3 est rectangulaire et correspond aux lignes telles que $\alpha < i + dj \leqslant d\alpha$ et $0 \leqslant i + j \leqslant \alpha$. Ainsi, on a $\det B = \det B_1 \cdot \det B_3$. Nous commençons par

[30] plus exactement, il est possible de généraliser l'étude réalisée pour $alpha = 2$ en utilisant les $\bar{Q}_j(x, y) = (a_2 x^2 + \ldots + a_d x^d + y)^j$, mais cela ne fournit pas des bornes aussi intéressantes que l'analyse que nous allons donner dans cette section. L'utilisation d'autres polynômes auxiliaires comme par exemple les $(a_d x^d + y)^j$ ne fait pas apparaître de structure aisément exploitable dans la matrice associée.

calculer le déterminant de la matrice B_1 :

$$\begin{aligned}\det B_1 &= \prod_{0 \leqslant i+dj \leqslant \alpha} \left((xT)^i \cdot Q_{i,j}\left(Tx, \frac{1}{M'}y\right)[x^i y^j] \right) \\ &= \prod_{0 \leqslant i+dj \leqslant \alpha} T^i (M')^{-j} \\ &= T^{\frac{1}{6d}(\alpha^3+O(\alpha^2))} (M')^{-\frac{1}{6d^2}(\alpha^3+O(\alpha^2))}.\end{aligned}$$

Il s'agit désormais de borner le déterminant de la matrice B_3. Pour faire cela, nous décrivons au paragraphe suivant une méthode permettant de borner les déterminants de réseaux donnés par des matrices de ce type.

1.5.2 Une méthode pour borner le déterminant d'une matrice.

La méthode décrite ici est générale, et pourrait être utilisée dans d'autres situations. Nous supposons que nous avons une matrice $d \times n$ (avec $d \leqslant n$), dont on considère le réseau engendré par les lignes. Soit B cette matrice. Nous supposons aussi que les entrées de la matrice B sont bornées en valeurs absolues par des produits de poids sur les lignes et les colonnes : $|B_{i,j}| \leqslant L_i \cdot C_j$, où les L_i sont les poids sur les lignes et les C_j les poids sur les colonnes. Nous dirons qu'une telle matrice est bornée par les produits des L_i et C_j. Ce type de matrice apparaît en particulier lorsque l'on utilise la technique de Coppersmith pour trouver de petites racines de polynômes.

Théorème 28. *Soit B une matrice $d \times n$ qui est bornée par le produit de quantités L_i et C_j. On rappelle que le déterminant de la matrice B est le produit des $\|\mathbf{b}_i^*\|$, où les vecteurs $\mathbf{b}_i$ sont les lignes de la matrice B. On a :*

$$\det B \leqslant 2^{\frac{d(d-1)}{2}} \cdot \sqrt{n} \cdot \left(\prod_{i=1}^{d} L_i \right) \cdot P,$$

où P est le produit des d plus grands C_j.

Démonstration. Si l'un des L_i est nul, le déterminant de la matrice B est nul aussi car au moins l'une de ses lignes est nulle. Nous supposons donc maintenant que tous les L_i sont strictement positifs. Soit B' la matrice B après avoir divisé la i-ème ligne par L_i pour tout $i \in [|1, d|]$. Sans perte de généralité, nous pouvons supposer que $C_1 \geqslant C_2 \geqslant \ldots \geqslant C_n$: en effet, si ce n'est pas le cas, il suffit de permuter les colonnes, ce qui ne change pas le déterminant de la matrice B. Il s'agit de montrer que le déterminant de la matrice B' est borné par $2^{\frac{d(d-1)}{2}} \cdot \sqrt{n} \cdot C_1 \cdot \ldots \cdot C_d$.

Nous prouvons cela par récurrence sur d et sur n. Si $n < d$, c'est évident car les lignes sont linéairement dépendantes, et donc le déterminant est nul. Si $d = 1$, le résultat est simple à vérifier. Supposons désormais que $n \geqslant d \geqslant 2$. Nous effectuons une permutation des lignes de la matrice B' de telle sorte que $|B'_{1,1}| \geqslant |B'_{i,1}|$ pour tout $i \in [|1, d|]$. Cela ne change pas le déterminant de la matrice B'. Nous appliquons désormais un pivot de Gauss

partiel à la matrice B'. Pour tout $i \geqslant 2$, nous effectuons la transformation élémentaire suivante :

$$B'_i \leftarrow B'_i - \frac{B'_{i,1}}{B'_{1,1}} B'_1,$$

ce qui ne modifie pas le déterminant de la matrice B'. On peut en effet supposer que $B'_{1,1}$ est non-nul car sinon tous les $B'_{i,1}$ sont nuls, et on obtient le résultat par récurrence sur n (on enlève la première colonne). Une fois ces transformations effectuées, on obtient une nouvelle matrice B' pour laquelle $|B'_{1,1}| \leqslant C_1$, $B'_{i,1} = 0$ pour tout $i \geqslant 2$ et $|B'_{i,j}| \leqslant 2C_j$, pour tous $i, j \geqslant 2$. Nous utilisons l'hypothèse de récurrence sur la matrice B'' qui a $d-1$ lignes et $n-1$ colonnes et qui correspond aux dernières lignes et dernières colonnes de la matrice B' :

$$\det B'' \leqslant 2^{\frac{(d-1)(d-2)}{2}} \cdot \sqrt{n-1} \cdot (2C_2) \cdot \ldots \cdot (2C_d) \leqslant 2^{\frac{d(d-1)}{2}} \cdot \sqrt{n} \cdot C_2 \cdot \ldots \cdot C_d.$$

À partir de ce point, on obtient facilement le résultat. □

Ce résultat doit être accompagné de plusieurs remarques. En premier lieu, il n'est pas optimal pour de nombreux réseaux. Par exemple, en utilisant ce théorème pour la situation $d = 2$ de la section 1.4, nous obtenons des bornes plus grandes sur les déterminants impliqués, c'est-à-dire plus faibles. Cela s'explique par des spécificités du réseau dont le théorème ne tire pas de bénéfice : d'une part, il n'utilise pas le fait que les matrices considérées sont triangulaires inférieures (c'est-à-dire qu'elles peuvent être complétées en des matrices carrées triangulaires inférieures en rajoutant des lignes), et d'autre part, il n'utilise pas les liens éventuels que les coefficients de la matrice ont entre eux. De même, on peut utiliser le théorème 28 pour analyser l'attaque de Boneh et Durfee [26] contre le cryptosystème RSA avec un petit exposant privé, mais la borne que l'on obtient alors est légèrement moins bonne que celle prouvée dans [26].

Par ailleurs, ce théorème a l'avantage d'être effectif. Supposons que l'on veuille réduire un réseau donné par une matrice bornée par le produit de quantités L_i et C_j. Avant de réduire le réseau, en particulier si la dimension est importante, il est tentant d'essayer de trouver le sous-ensemble des lignes de la matrice qui minimise la borne de Minkowski. Ce problème peut ne pas être facile à résoudre, mais nous pouvons y apporter une solution grâce au théorème 28 : celle-ci ne sera peut-être pas optimale, mais elle sera obtenue efficacement. On procède comme suit : on range les C_j par ordre décroissant et les L_i par ordre croissant ; pour d allant de la dimension de la matrice de départ jusqu'à 1, on calcule $\prod_{i=1}^{d} (L_i C_i)^{1/d}$; on garde l'ensemble des lignes qui minimise cette quantité, et on réduit le réseau engendré par ces lignes.

1.5.3 Calcul du déterminant de la matrice B_3 (introduite au paragraphe 1.5.1).

Nous cherchons maintenant à borner $(\det B_3)$ en utilisant le théorème 28. Dans ce but, nous commençons par borner la matrice B_3 par le produit de quantités L_i et C_j, ou plus précisément par le produit de quantités $L_{i,j}$ et $C_{i',j'}$ car nous décrivons la matrice B_3 à l'aide de doubles indices.

Lemme 42. *La matrice B_3 est bornée par le produit des quantités :*

$$\begin{aligned} L_{i,j} &= 2^j (d+1)^j K^j N_1^i N_2^j, \\ C_{i',j'} &= \frac{2^{j'} T^{i'}}{M^{j'} N_1^{i'} N_2^{j'}}. \end{aligned}$$

Démonstration. La ligne (i,j) de la matrice B_3 pour $i+j \in [|0,\alpha|]$ et $i+dj \in [|\alpha+1, d\alpha|]$ correspond au polynôme $(Tx)^i Q\left(Tx, \frac{1}{M'}y\right)^j$. Sa colonne $(i'+j')$ pour $i'+j' \in [|\alpha+1, d\alpha|]$ correspond au coefficient de ce polynôme pour le monôme $x^{i'}y^{j'}$. Ainsi, en utilisant la notation $Q(x,y) = P(x) + y$, et en définissant $B_3[(i,j);(i',j')]$ comme l'entrée à la ligne (i,j) et à la colonne (i',j') de la matrice B_3, on a :

$$\begin{aligned} |B_3[(i,j);(i',j')]| &\leqslant (Tx)^i \left(Q\left(xT, \frac{1}{M'}y\right)\right)^j [x^{i'}y^{j'}] \\ &\leqslant T^i \left(Q\left(xT, \frac{1}{M'}y\right)\right)^j [x^{i'-i}y^{j'}] \\ &\leqslant T^i \left[\sum_{k=0}^{j} \binom{j}{k} (P(xT))^{j-k} (M')^{-k} y^k\right] [x^{i'-i}y^{j'}] \\ &\leqslant \binom{j}{j'} T^i (M')^{-j'} (P(xT))^{j-j'} [x^{i'-i}] \\ &\leqslant 2^j T^i (M')^{-j'} \left[\sum_{k=0}^{d} a_k T^k x^k\right]^{j-j'} [x^{i'-i}]. \end{aligned}$$

Par ailleurs, nous avons $a_k \leqslant K \frac{N_2}{N_1^k}$, ce qui nous permet d'écrire :

$$\begin{aligned} |B_3[(i,j);(i',j')]| &\leqslant 2^j K^{j-j'} T^i N_2^{j-j'} (M')^{-j'} \left[\sum_{k=0}^{d} \left(\frac{T}{N_1}\right)^k x^k\right]^{j-j'} [x^{i'-i}]. \\ &\leqslant 2^j K^{j-j'} T^i N_2^{j-j'} (M')^{-j'} \left(\frac{T}{N_1}\right)^{i'-i} \left[\sum_{k=0}^{d} x^k\right]^{j-j'} [x^{i'-i}] \\ &\leqslant \left(2^j (d+1)^j K^j N_1^i N_2^j\right) \left(\frac{2^{j'} T^{i'}}{M^{j'} N_1^{i'} N_2^{j'}}\right). \end{aligned}$$

Dans la dernière inégalité, nous avons utilisé la minoration $M' \geqslant M/2$. □

Grâce au lemme ci-dessus, nous pouvons désormais utiliser le théorème 28. Cela nous permet d'obtenir :

$$\det B_3 \leqslant 2^{\frac{\alpha^4}{8} + O(\alpha^3)} \cdot \sqrt{d} \cdot \mathbf{L} \cdot \mathbf{C},$$

où $\mathbf{L}$ est le produit des $L_{i,j}$ pour $i+j \in [|0,\alpha|]$ et $i+dj \in [|\alpha+1, d\alpha|]$, et $\mathbf{C}$ est le produit des $(\dim B_3)$ plus grands $C_{i,j}$ pour $i+dj \in [|\alpha+1, d\alpha|]$. Dans la borne supérieure ci-dessus, certains termes sont faciles à évaluer : nous commençons par ceux-là.

Lemme 43. *Avec les notations ci-dessus, on a :*

$$\begin{aligned}\dim B_3 &= \left(\frac{1}{2}-\frac{1}{2d}\right)(\alpha^2+O(\alpha)),\\ \mathbf{L} &= 2^{O(\alpha^3)}(d+1)^{O(\alpha^3)}K^{O(\alpha^3)}N_1^{\frac{d-1}{6d}(\alpha^3+O(\alpha^2))}N_2^{\frac{d^2-1}{6d^2}(\alpha^3+O(\alpha^2))}.\end{aligned}$$

Démonstration. Pour la première relation, on écrit :

$$\dim B_3 = \dim B - \dim B_1 = \frac{1}{2}(\alpha^2+O(\alpha)) - \frac{1}{2d}(\alpha^2+O(\alpha)).$$

On s'inspire de cela pour montrer la deuxième relation :

$$\begin{aligned}\prod_{i+j\in[|0,\alpha|],i+dj\in[|\alpha+1,d\alpha|]}(N_1^iN_2^j) &= \left(\prod_{i+j\leqslant\alpha}(N_1^iN_2^j)\right)\cdot\left(\prod_{i+dj\leqslant\alpha}(N_1^{-i}N_2^{-j})\right)\\ &= (N_1N_2)^{\frac{1}{6}(\alpha^3+O(\alpha^2))}N_1^{\frac{-1}{6d}(\alpha^3+O(\alpha^2))}N_2^{-\frac{1}{6d^2}(\alpha^3+O(\alpha^2))}\\ &= N_1^{\frac{d-1}{6d}(\alpha^3+O(\alpha^2))}N_2^{\frac{d^2-1}{6d^2}(\alpha^3+O(\alpha^2))}.\end{aligned}$$

□

Il nous reste donc à évaluer le dernier terme, c'est-à-dire $\mathbf{C}$. Nous faisons cela de la manière suivante : nous écrivons $\mathbf{C} \leqslant 2^{O(\alpha^3)}\prod_{k=\tau\alpha}^{k_{\max}} 2^{kc_k}$, où c_k est le nombre de paires (i,j) telles que $i+dj \in [|\alpha+1, d\alpha|]$ et :

$$(t-n_1)i-(m+n_2)j \in [k,k+1[,$$

avec $t=\lg T, n_1=\lg N_1, m=\lg M$ et $n_2=\lg N_2$. Enfin, le paramètre de sommation τ est choisi maximal ($k_{\max}$ et τ sont négatifs) tel que $\sum_{k=\tau\alpha}^{k_{\max}} c_k \geqslant \dim B_3$, c'est-à-dire de telle sorte que l'on ait pris suffisamment de colonnes. On fixe :

$$k_{\max} = \begin{cases} \lfloor \alpha(t-n_1)\rfloor & \text{si} \quad MN_2 \geqslant \left(\frac{N_1}{T}\right)^d,\\ \lfloor -(m+n_2)\frac{\alpha}{d}\rfloor & \text{si} \quad MN_2 \leqslant \left(\frac{N_1}{T}\right)^d.\end{cases}$$

Remarquons que si la borne de Taylor est finement satisfaite, c'est-à-dire si $MN_2T^{d+1} \approx N_1^{d+1}$, alors nous avons $k_{\max} = \lfloor \alpha(t-n_1)\rfloor$. Les figures 1.4, 1.5 et 1.6 permettent de mieux visualiser les liens entre les indices i, j et k. Dans les trois cas, la partie hachurée a déjà été prise en compte dans le calcul de $(\det B_1)$, et on sépare la surface des indices i, j possibles (le grand triangle privé de la partie hachurée) en trois zones Z_1, Z_2 et Z_3 qui dépendent de la valeur de k. La première figure correspond au cas où $MN_2 \in \left[\left(\frac{N_1}{T}\right)^d, \left(\frac{N_1}{T}\right)^{d+1}\right]$, la deuxième figure correspond au cas où $MN_2 \in \left[\frac{N_1}{T}, \left(\frac{N_1}{T}\right)^d\right]$, et la dernière figure correspond au cas où $MN_2 \leqslant \frac{N_1}{T}$. Il convient de noter que si dans notre contexte on a la liberté de fixer le paramètre M comme on le souhaite, du point de vue de la complexité de l'algorithme, il est préférable de prendre T aussi grand que possible, et donc d'atteindre la borne de Taylor. Dans ce cas, on est dans la situation de la figure 1. Pour les trois situations

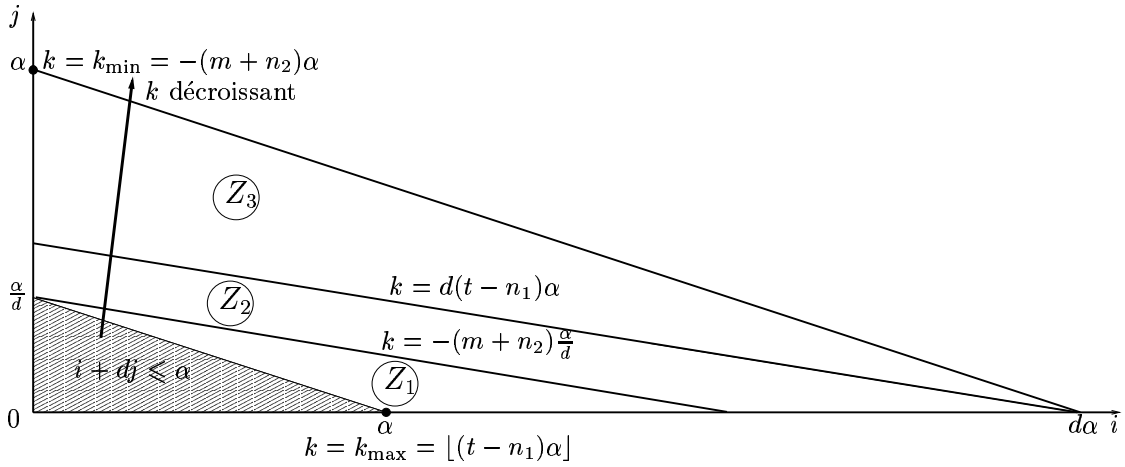


FIG. 1.4 – Lien entre les indices i, j et k lorsque $\left(\frac{N_1}{T}\right)^d \leqslant MN_2 \leqslant \left(\frac{N_1}{T}\right)^{d+1}$.

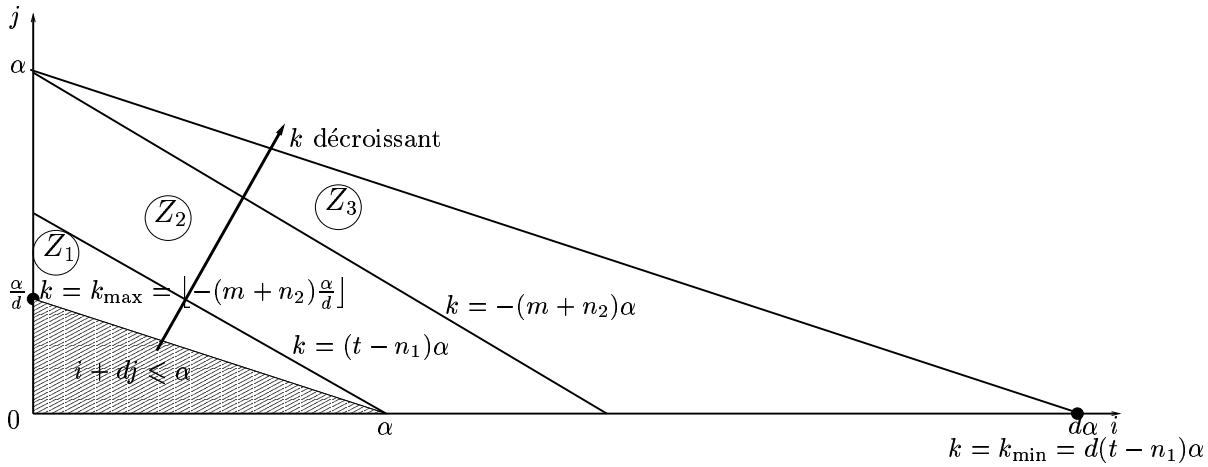


FIG. 1.5 – Lien entre les indices i, j et k lorsque $\frac{N_1}{T} \leqslant MN_2 \leqslant \left(\frac{N_1}{T}\right)^d$.

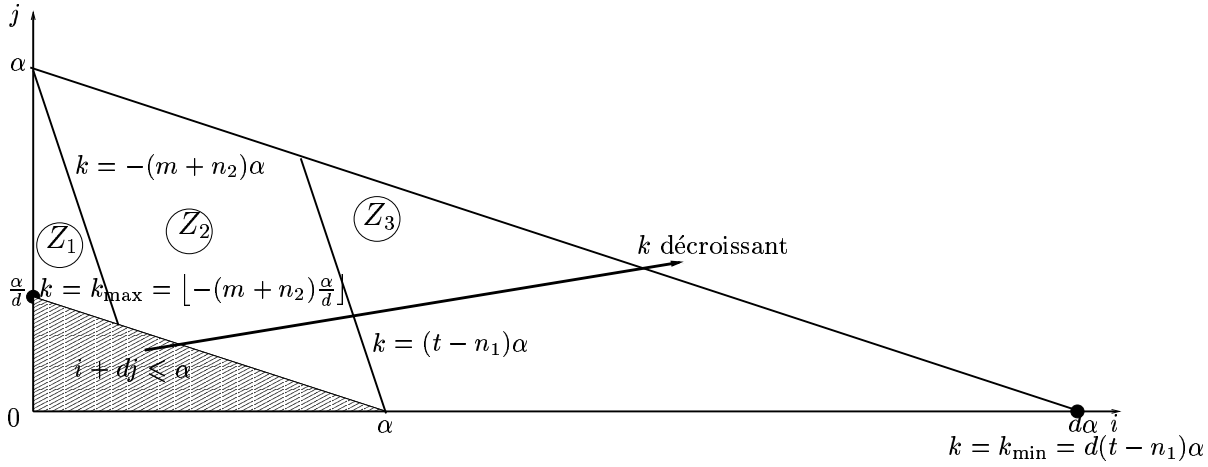


FIG. 1.6 – Lien entre les indices i, j et k lorsque $MN_2 \leqslant \frac{N_1}{T}$.

possibles, nous n'étudions à chaque fois que les zones Z_1 et Z_2, car nous allons montrer que $\tau\alpha$ correspond toujours à un indice k qui se situe dans Z_2.

Pour borner $\mathbf{C}$, on procède en plusieurs étapes : tout d'abord, on estime c_k en fonction de k, puis on détermine τ à partir du fait que l'on connaît la quantité $\dim B_3$, ce qui nous permet de finalement majorer $\mathbf{C}$.

Lemme 44. *Avec les notations qui précèdent,*

1. *Si* $\left(\frac{N_1}{T}\right)^d \leqslant MN_2 \leqslant \left(\frac{N_1}{T}\right)^{d+1}$ *et* $\alpha \to \infty$ *:*

$$c_k = \begin{cases} \frac{-k+(t-n_1)\alpha}{(n_1-t)((t-n_1)d+m+n_2)}\left(1+O\left(\frac{1}{\alpha}\right)\right) & si \quad k \in \left[\frac{-(m+n_2)\alpha}{d}, (t-n_1)\alpha\right] \quad (Z_1) \\ \frac{k}{(t-n_1)(m+n_2)}\left(1+O\left(\frac{1}{\alpha}\right)\right) & si \quad k \in \left[d(t-n_1)\alpha, \frac{-(m+n_2)\alpha}{d}\right] \quad (Z_2) \end{cases}$$

2. *Si* $\frac{N_1}{T} \leqslant MN_2 \leqslant \left(\frac{N_1}{T}\right)^d$ *et* $\alpha \to \infty$ *:*

$$c_k = \begin{cases} \frac{kd+\alpha(m+n_2)}{(m+n_2)((t-n_1)d+m+n_2)}\left(1+O\left(\frac{1}{\alpha}\right)\right) & si \quad k \in \left[(t-n_1)\alpha, \frac{-(m+n_2)\alpha}{d}\right] \quad (Z_1) \\ \frac{k}{(t-n_1)(m+n_2)}\left(1+O\left(\frac{1}{\alpha}\right)\right) & si \quad k \in [-(m+n_2)\alpha, (t-n_1)\alpha] \quad (Z_2) \end{cases}$$

3. *Si* $MN_2 \leqslant \frac{N_1}{T}$ *et* $\alpha \to \infty$ *:*

$$c_k = \begin{cases} \frac{kd+\alpha(m+n_2)}{(m+n_2)((t-n_1)d+m+n_2)}\left(1+O\left(\frac{1}{\alpha}\right)\right) & si \quad k \in \left[-(m+n_2)\alpha, \frac{-(m+n_2)\alpha}{d}\right] \quad (Z_1) \\ \frac{-(d-1)\alpha}{(t-n_1)d+m+n_2}\left(1+O\left(\frac{1}{\alpha}\right)\right) & si \quad k \in [(t-n_1)\alpha, -(m+n_2)\alpha] \quad (Z_2) \end{cases}$$

À partir du résultat précédent, on déduit directement que :

Lemme 45. *Avec les mêmes notations,*

1. *Si* $\left(\frac{N_1}{T}\right)^d \leqslant MN_2 \leqslant \left(\frac{N_1}{T}\right)^{d+1}$ *et* $\alpha \to \infty$ *:*

$$\sum_{l=k}^{(t-n_1)\alpha} c_l = \begin{cases} \frac{(-k+(t-n_1)\alpha)^2}{2(n_1-t)(d(t-n_1)+m+n_2)}\left(1+O\left(\frac{1}{\alpha}\right)\right) & si \quad k \in \left[\frac{-(m+n_2)\alpha}{d}, (t-n_1)\alpha\right] \\ \frac{-\alpha^2}{2d} + \frac{k^2}{2(n_1-t)(m+n_2)}\left(1+O\left(\frac{1}{\alpha}\right)\right) & si \quad k \in \left[d(t-n_1)\alpha, \frac{-(m+n_2)\alpha}{d}\right]. \end{cases}$$

2. *Si* $\frac{N_1}{T} \leqslant MN_2 \leqslant \left(\frac{N_1}{T}\right)^d$ *et* $\alpha \to \infty$ *:*

$$\sum_{l=k}^{\frac{-(m+n_2)\alpha}{d}} c_l = \begin{cases} \frac{-d\left(k+\frac{m+n_2}{d}\alpha\right)^2}{2(m+n_2)((t-n_1)d+m+n_2)}\left(1+O\left(\frac{1}{\alpha}\right)\right) & si \quad k \in \left[(t-n_1)\alpha, \frac{-(m+n_2)\alpha}{d}\right] \\ -\frac{\alpha^2}{2d} + \frac{k^2}{2(n_1-t)(m+n_2)}\left(1+O\left(\frac{1}{\alpha}\right)\right) & si\ k \in [-(m+n_2)\alpha, (t-n_1)\alpha]. \end{cases}$$

3. *Si* $MN_2 \leqslant \frac{N_1}{T}$ *et* $\alpha \to \infty$ *:*

$$\sum_{l=k}^{\frac{-(m+n_2)\alpha}{d}} c_l = \begin{cases} \frac{-d\left(k+\frac{m+n_2}{d}\alpha\right)^2}{2(m+n_2)((t-n_1)d+m+n_2)}\left(1+O\left(\frac{1}{\alpha}\right)\right) si \quad k \in \left[-(m+n_2)\alpha, \frac{-(m+n_2)\alpha}{d}\right] \\ \frac{(d^2-1)(m+n_2)\alpha^2+2d(d-1)\alpha k}{2d((t-n_1)d+m+n_2)}\left(1+O\left(\frac{1}{\alpha}\right)\right) si\ k \in [(t-n_1)\alpha, -(m+n_2)\alpha]. \end{cases}$$

Cela nous permet d'établir la valeur du paramètre τ, dans le lemme 46. De manière amusante, les cas Z_1 et Z_2 se regroupent :

Lemme 46. *Avec les mêmes notations :*

$$\tau = \begin{cases} -\sqrt{(n_1-t)(m+n_2)} & si \quad MN_2 \in \left[\frac{N_1}{T}, \left(\frac{N_1}{T}\right)^{d+1}\right] \\ -\frac{n_1-t+m+n_2}{2} & si \qquad MN_2 \leqslant \frac{N_1}{T}. \end{cases}$$

Démonstration. Nous ne considérons ici que le cas où $MN_2 \in \left[\left(\frac{N_1}{T}\right)^d, \left(\frac{N_1}{T}\right)^{d+1}\right]$, les cas où $MN_2 \in \left[\frac{N_1}{T}, \left(\frac{N_1}{T}\right)^d\right]$ et où $MN_2 \leqslant \frac{N_1}{T}$ pouvant être traités de manière similaire. Rappelons que l'on cherche à trouver τ tel que :

$$\sum_{k=\tau\alpha}^{(t-1)\alpha} c_k = \dim B_3 + O(\alpha) = \frac{-\alpha^2}{2d} + \frac{\alpha^2}{2} + O(\alpha).$$

Grâce au lemme 45, on voit que la somme des c_l pour $l \in \left[-(m+n_2)\frac{\alpha}{d}, (t-n_1)\alpha\right]$ vaut $\left(\frac{-1}{2d} + \frac{m+n_2}{2d^2(n_1-t)}\right)(\alpha^2 + O(\alpha))$. Comme on a $m+n_2 \leqslant (d+1)(n_1-t)$ (il s'agit de la condition de Taylor), on a $m+n_2 \leqslant d^2(n_1-t)$, ce qui signifie qu'il n'y a pas assez de termes dans la zone Z_1. Toujours à l'aide du lemme 45, on voit que la somme des c_l pour $l \in [d(t-n_1)\alpha, (t-n_1)\alpha]$ vaut $\left(\frac{-1}{2d} + \frac{d^2(n_1-t)}{2(m+n_2)}\right)(\alpha^2 + O(\alpha))$. La condition de Taylor nous indique ici qu'il y a trop de termes dans l'union des zones Z_1 et Z_2. Par conséquent, on a $\tau \in \left[d(t-n_1), \frac{-(m+n_2)}{d}\right]$. À l'aide du lemme 45, on obtient que τ est la solution négative de l'équation :

$$\frac{-1}{2d} + \frac{1}{2} = \frac{-1}{2d} + \frac{\tau^2}{2(n_1-t)(m+n_2)},$$

ce qui donne le résultat. □

Dans le lemme suivant, nous estimons les sommes $\sum_{l=\tau\alpha}^{k_{\max}} l c_l$.

Lemme 47. *Avec les mêmes notations,*

1. *Si $\frac{N_1}{T} \leqslant MN_2 \leqslant \left(\frac{N_1}{T}\right)^{d+1}$ et $\alpha \to \infty$, on a :*

$$\sum_{l=\tau\alpha}^{k_{\max}} l c_l = \frac{(n_1-t)d + (m+n_2) - 2d^2\sqrt{(n_1-t)(m+n_2)}}{6d^2}(\alpha^3 + O(\alpha^2)).$$

2. *Si $MN_2 \leqslant \frac{N_1}{T}$ et $\alpha \to \infty$, on a :*

$$\sum_{l=\tau\alpha}^{k_{\max}} l c_l = (d-1)\frac{3d^2(t-n_1)^2 - 6d^2(t-n_1)(m+n_2) - (d+2)^2(m+n_2)^2}{24d^2((t-n_1)d + m + n_2)}(\alpha^3 + O(\alpha^2)).$$

Démonstration. Nous ne démontrons ce lemme que pour la situation où l'on a $MN_2 \in \left[\left(\frac{N_1}{T}\right)^d, \left(\frac{N_1}{T}\right)^{d+1}\right]$, les deux autres assertions pouvant être montrées de manière similaire. À l'aide du lemme 44, on obtient :

$$\sum_{l=\frac{-(m+n_2)\alpha}{d}}^{(t-n_1)\alpha} lc_l = \frac{((t-n_1)d-2(m+n_2))((t-n_1)d+m+n_2)}{6(n_1-t)d^3}(\alpha^3+O(\alpha^2)).$$

De même, on a :

$$\sum_{l=\tau\alpha}^{\frac{-(m+n_2)\alpha}{d}} lc_l = \frac{(d^2(t-n_1)-(m+n_2)-d\tau)(m+n_2-d\tau)}{3(t-n_1)d^3}(\alpha^3+O(\alpha^2)).$$

En additionnant ces deux termes, on trouve le résultat annoncé. □

À partir de maintenant, nous ne considérons plus ici le cas où $MN_2 \leqslant \frac{N_1}{T}$, pour deux raisons. Tout d'abord, les formules qu'il fait intervenir ne semblent pas se simplifier, et d'autre part, ce cas n'est pas intéressant vis-à-vis de nos motivations, que ce soit pour le calcul des pires cas, ou pour inverser une fonction quand les premiers bits ont été perdus. En effet, dans ce cas-là, on a $MN_2 \leqslant N_1$, alors que pour la recherche des pires cas, on a $N_1 = N_2$, et dans le cas de l'inversion d'une fonction quand les premiers bits sont inconnus, on fait tendre M vers l'infini beaucoup plus vite que N_1. Le lemme 47 donne une borne sur $\mathbf{C}$ et donc sur le déterminant de la matrice B_3. Cela nous permet de borner le déterminant de la matrice B.

Théorème 29. *Avec les notations précédentes, si* $\frac{N_1}{T} \leqslant MN_2 \leqslant \left(\frac{N_1}{T}\right)^{d+1}$ *et si* $\alpha \to \infty$, *alors on a :*

$$\det B \leqslant 2^{O(\alpha^4)} \cdot (d+1)^{O(\alpha^3)} \cdot K^{O(\alpha^3)} \cdot (N_1N_2)^{\frac{\alpha^3}{6}+O(\alpha^2)} \cdot 2^{-\frac{1}{3}\sqrt{(n_1-t)(m+n_2)}(\alpha^3+O(\alpha^2))} \cdot (MT)^{O(\alpha^2)}.$$

Démonstration. Nous utilisons la relation :

$$\det B = \det B_1 \cdot \det B_3 \leqslant 2^{\frac{\alpha^4}{8}+O(\alpha^3)} \cdot \det B_1 \cdot \mathbf{L} \cdot \mathbf{C},$$

ainsi que les bornes que nous avons calculées pour chacun des termes. □

1.5.4 Borne de Coppersmith et conclusion.

Nous pouvons désormais conclure l'analyse dans le cas où $d \geqslant 3$. Nous rappelons que la dimension du réseau L considéré est $\frac{\alpha^2}{2}+O(\alpha)$. Puisque le plus court vecteur de L est de longueur au moins $(M')^{-\alpha}$ (cela provient de la relation $\lambda_1(L) \geqslant \min_i \|\mathbf{b}_i^*\|$, où les vecteurs $\mathbf{b}_i$ sont une base de L, comme expliqué au chapitre I-1), l'algorithme LLL renvoie deux vecteurs de normes ℓ_1 plus petites que 1 si $\sqrt{\dim B} \cdot 2^{\dim B}((M')^{-\alpha}\det B)^{\frac{1}{\dim B}} < 1$. Ceci nous donne la condition de Coppersmith suivante :

$$2^{O(\alpha^2)} \cdot (d+1)^{O(\alpha)} \cdot K^{O(\alpha)} \cdot (N_1N_2)^{\frac{\alpha}{3}+O(1)} \cdot 2^{-\frac{2}{3}\sqrt{(n_1-t)(m+n_2)}(\alpha+O(1))} \cdot (MT)^{O(1)} \leqslant 1.$$

Nous supposons maintenant que K est constant, que $\alpha \geqslant \log d$, et que $MN_2 \geqslant N_1 \geqslant T$. Alors, sous l'hypothèse que les deux premiers vecteurs renvoyés par l'algorithme LLL ont des polynômes associés qui sont algébriquement indépendants, on peut trouver toutes les solutions de l'équation (1.1) en temps $\mathcal{P}\mathrm{ol}(n_1, n_2, m, d, \alpha)\frac{N_1}{T}$, tant que :

$$t(1+\varepsilon_1) \leqslant \min\left(n_1 - \frac{m+n_2+O(1)}{d+1}, n_1 - \frac{(n_1+n_2)^2}{4(m+n_2)} + \varepsilon_2\right),$$

avec :

$$\begin{aligned}\varepsilon_1 &= \frac{1}{m+n_2}O(1) + O\left(\frac{1}{\alpha}\right)\\ \varepsilon_2 &= \frac{1}{m+n_2}O(\alpha^2) + \frac{n_1}{m+n_2}O(\alpha) + (n_1+n_2)O\left(\frac{1}{\alpha}\right) + (m+n_2)O\left(\frac{1}{\alpha^2}\right).\end{aligned}$$

Nous avons prouvé le théorème 25.

Négligeons maintenant les termes $O(\cdot), \varepsilon_1$ et ε_2, qui sont voués à être négligeables (l'exploitation rigoureuse des résultats est effectuée au paragraphe 1.2.4). Supposons aussi que $N_1 = N_2 = N$. Si on peut choisir M, alors en prenant $M \approx N^{\sqrt{d+1}-1}$ et $T \approx N^{1-\frac{1}{\sqrt{d+1}}}$, on aboutit à un algorithme de complexité $\approx N^{\frac{1}{\sqrt{d+1}}}$. On voit ainsi que pour que cette complexité devienne polynomiale en $n = \lg N$, il faut prendre $d = \Omega(n^2)$.

1.6 Le cas de fonctions de plusieurs variables.

Nous cherchons désormais à généraliser l'étude effectuée aux sections 1.4 et 1.5 à une fonction f de plusieurs variables. Supposons que f soit une fonction de ν variables. On pose quelques hypothèses simplificatrices pour faciliter l'analyse :

- la fonction $f([1/2,1[^\nu) \subset [1/2,1[$ est $\mathcal{C}^\infty$ sur $[1/2,1]^\nu$,
- si M_d est le maximum en valeur absolue de toutes les dérivées partielles d'ordre d de la fonction f sur $[1/2,1[^\nu$, alors $\frac{\nu^d}{d!}M_d \leqslant 1$,
- on prend le même entier N pour les « précisions d'entrée-sortie » (toutes les entrées et la sortie ont la même précision).

Pour simplifier l'analyse, nous supprimons aussi tous les termes ne dépendant ni de T, ni de N, ni de M : la notation $a \ll b$ signifie que $a \leqslant b$, à des facteurs multiplicatifs ne dépendant pas de T, N et M.

On cherche à trouver tous les ν-uplets $(x_1, \ldots, x_\nu) \in [|N/2, N-1|]^\nu$ tels que :

$$\left| Nf\left(\frac{x_1}{N}, \ldots, \frac{x_\nu}{N}\right) + c \bmod 1 \right| \leqslant \frac{1}{M}, \tag{1.4}$$

où N, M et c sont trois réels avec un nombre fini de bits après la virgule, et sont donnés en entrée.

Le principe de la méthode est exactement le même que dans le cas d'une fonction d'une seule variable. Nous subdivisons l'hypercube $[1/2,1]^\nu$ en $\left(\frac{N}{4T}\right)^\nu$ sous-hypercubes de côté $\frac{2T}{N}$. Pour chaque sous-hypercube, nous approchons la fonction f par un polynôme de ν

variables de degré total d, qui est en fait l'ensemble des termes de degré total inférieur ou égal à d dans le développement de Taylor de la fonction f pris au centre du sous-hypercube. De même que dans le cas d'une seule variable, on veut que l'erreur commise lors de cette approximation soit au plus de l'ordre de $\frac{1}{M}$, ce qui nous donne, en utilisant la formule de Taylor-Lagrange à l'ordre d :

$$MT^d \leqslant N^{d-1}. \tag{1.5}$$

Pour chacune des approximations polynomiales P, on cherche les ν-uplets $(x_1, \ldots, x_\nu) \in [|-T, T|]^\nu$ qui sont solutions de l'équation :

$$\left| NP\left(\frac{x_1}{N}, \ldots, \frac{x_\nu}{N}\right) \mod 1 \right| \leqslant \frac{2}{M}.$$

Pour faire cela, nous utilisons la méthode de Coppersmith à un certain ordre α : on réduit un réseau correspondant à une famille de polynômes à $\nu+1$ variables construits à partir du polynôme $Q(x_1, \ldots, x_\nu, y) = NP\left(\frac{x_1}{N}, \ldots, \frac{x_\nu}{N}\right) + y$, en utilisant ses puissances jusqu'à la α-ième. Après avoir réduit le réseau, on obtient au moins $\nu+1$ vecteurs de normes ℓ_1 plus petites que 1 strictement (si la condition de Coppersmith sur les paramètres T, M et N est satisfaite), et nous cherchons les $(\nu+1)$-uplets qui sont des racines réelles communes à ces polynômes. Cette dernière étape peut échouer, en particulier si les $\nu+1$ polynômes trouvés sont algébriquement dépendants. Dans ce cas, on subdivise l'hypercube en 2^ν sous-hypercubes et on réessaye la méthode sur chacun de ceux-là. Si le côté du sous-hypercube considéré devient petit, on effectue une recherche exhaustive. Dans l'analyse de l'algorithme, on supposera que l'étape de résolution du système algébrique de $\nu+1$ variables et $\nu+1$ polynômes n'échoue pas. Ainsi, l'analyse consiste uniquement à borner le déterminant du réseau que l'on réduit, pour obtenir la borne de Coppersmith correspondante.

1.6.1 L'analyse naïve.

Nous commençons par une étude naïve du réseau que l'on réduit, comme celle du paragraphe 1.3, que l'on avait effectuée dans le contexte d'une fonction d'une seule variable.

Nous considérons la famille de polynômes $Q_{\mathbf{i},j} = \mathbf{x}^{\mathbf{i}}(P(\mathbf{x}) + y)^j$ pour $0 \leqslant (\sum \mathbf{i}) + dj \leqslant d\alpha$, où $\mathbf{i} = (i_1, \ldots, i_\nu)$, $\mathbf{x}^{\mathbf{i}} = x_1^{i_1} \ldots x_\nu^{i_\nu}$ et $\sum \mathbf{i} = \sum_{k=1}^{\nu} i_k$. Nous ferons tendre le paramètre α vers l'infini ultérieurement. Les monômes qui apparaissent dans cette famille de polynômes sont les $\mathbf{x}^{\mathbf{i}} y^j$ pour $0 \leqslant (\sum \mathbf{i}) + dj \leqslant d\alpha$. À partir de maintenant, nous ordonnons ces polynômes et ces monômes par valeur de $(\sum \mathbf{i}) + dj$ croissante, puis par valeur de j croissante en cas d'égalité, puis par ordre lexicographique sur les x_i (en prenant $x_1 < x_2 < \ldots < x_k$ comme ordre sur les variables). Le réseau L que nous considérons est celui engendré par les lignes de la matrice constituée par les coefficients des polynômes $Q\left(T\mathbf{x}, \frac{1}{M'}y\right)$ vis-à-vis des monômes $\mathbf{x}^{\mathbf{i}} y^j$. Ce réseau a pour dimension

$$\dim L = \sum_{0 \leqslant (\sum \mathbf{i}) + dj \leqslant d\alpha} 1 = \frac{d^\nu}{(\nu+1)!}(\alpha^{\nu+1} + O(\alpha^\nu)).$$

De plus, son déterminant est facile à calculer, puisque la base est triangulaire.

$$\det L = T^{\kappa_1(\alpha^{\nu+2}+O(\alpha^{\nu+1}))}(M')^{-\kappa_2(\alpha^{\nu+2}+O(\alpha^{\nu+1}))},$$

avec :

$$\begin{array}{rcccl} \kappa_1 & = & \sum_{0\leqslant(\sum \mathbf{i})+dj\leqslant d\alpha}(\sum \mathbf{i}) & = & \frac{\nu d^{\nu+1}}{(\nu+2)!}, \\ \kappa_2 & = & \sum_{0\leqslant(\sum \mathbf{i})+dj\leqslant d\alpha} j & = & \frac{d^{\nu}}{(\nu+2)!}. \end{array}$$

Nous obtenons par conséquent la borne de Coppersmith $T \ll (M')^{\frac{1}{d\nu}}$. Avec la borne de Taylor (l'équation (1.5)), nous obtenons :

$$T \ll \min\left(\frac{N^{\frac{d}{d+1}}}{M^{\frac{1}{d+1}}}, M^{\frac{1}{d\nu}}\right).$$

Si on peut faire varier M, alors le choix optimal est $M \approx N^{\frac{d^2\nu}{d(\nu+1)+1}}$, avec $T \approx N^{\frac{d}{d(\nu+1)+1}}$.

Dans les paragraphes qui suivent, nous allons améliorer la borne de Coppersmith en généralisant les analyses des sections 1.4 et 1.5. Nous considérons désormais la famille restreinte de polynômes constituée des $Q_{\mathbf{i},j}$ avec $\mathbf{i}$ et j qui satisfont $0 \leqslant (\sum \mathbf{i})+j \leqslant \alpha$. Nous appelons B la matrice de la base associée à ces polynômes, et L' le réseau engendré par les lignes de B. La famille de monômes reste la même, de telle sorte que B a $\frac{d^{\nu}}{(\nu+1)!}(\alpha^{\nu+1} + O(\alpha^{\nu}))$ colonnes et son nombre de lignes est :

$$\dim L' = \frac{1}{(\nu+1)!}(\alpha^{\nu+1} + O(\alpha^{\nu})).$$

Le but des paragraphes suivants est de borner le plus finement possible la quantité $(\det L')$.

1.6.2 Analyse du cas $d = 2$.

Nous bornons ici le déterminant du réseau L' dans le cas où $d = 2$. Nous considérons la famille de polynômes $\bar{Q}_j$ définis par $\bar{Q}_0 = 1$ et $\bar{Q}_j(\mathbf{x}, y) = Q(\mathbf{x}, y)^j - \sum_{k=0}^{j-1}\binom{j}{k}(a_0 + \mathbf{a}_1 \cdot \mathbf{x})^{j-k}\bar{Q}_k(\mathbf{x}, y)$ si $j > 0$, où $Q(\mathbf{x}, y) = a_0 + \mathbf{a}_1 \cdot \mathbf{x} + \mathbf{a}_2 \cdot \mathbf{x}^2$, avec $\mathbf{a}_1 \cdot \mathbf{x} = \sum_{k=1}^{\nu} a_{1,k}x_k$ et $\mathbf{a}_2 \cdot \mathbf{x}^2 = \sum_{k_1,k_2\in[|1,\nu|]} a_{2,k_1,k_2}x_{k_1}x_{k_2}$. Il est facile de vérifier que l'on a l'égalité suivante, pour tout $j \geqslant 0$:

$$\bar{Q}_j(\mathbf{x}, y) = (\mathbf{a}_2 \cdot \mathbf{x}^2 + y)^j.$$

Dans la matrice B, nous remplaçons chaque ligne correspondant à un polynôme $\mathbf{x}^{\mathbf{i}}Q^j$ par la ligne correspondant au polynôme $\mathbf{x}^{\mathbf{i}}\bar{Q}^j$. Le déterminant reste le même car la transformation effectuée peut s'écrire comme une suite de transformations élémentaires chacune de déterminant 1. Soit $\bar{B}$ la nouvelle matrice. Celle-ci est de la forme suivante :

$$\bar{B} = \begin{bmatrix} \bar{B}_1 & 0 \\ 0 & \bar{B}_2 \end{bmatrix},$$

où $\bar{B}_1$ est carrée et correspond aux lignes et aux colonnes pour lesquelles on a $0 \leqslant (\sum \mathbf{i}) + 2j \leqslant \alpha$. On a $\det L' = \det \bar{B}_1 \cdot \det \bar{B}_2$. Comme la matrice $\bar{B}_1$ est carrée, son déterminant est simple à évaluer :

$$\det \bar{B}_1 = T^{\kappa_3(\alpha^{\nu+2}+O(\alpha^{\nu+1}))}(M')^{-\kappa_4(\alpha^{\nu+2}+O(\alpha^{\nu+1}))},$$

avec :

$$\begin{array}{rcccl} \kappa_3 & = & \sum_{0\leqslant(\sum \mathbf{i})+2j\leqslant\alpha}(\sum \mathbf{i}) & = & \frac{\nu}{2(\nu+2)!}, \\ \kappa_4 & = & \sum_{0\leqslant \mathbf{i}+2j\leqslant\alpha} j & = & \frac{1}{4(\nu+2)!}. \end{array}$$

La matrice $\bar{B}_2$ s'écrit :

$$\bar{B}_2 = \begin{bmatrix} T^e\cdot A_1^{([\alpha/2])} & & & & & & & & \\ & \ddots & & & & & & & \\ & & A_{\alpha-2i}^{(\alpha-i)} & & & & & & \\ & & & T\cdot A_{\alpha-2i+1}^{(\alpha-i)} & & & & & \\ & & & & \ddots & & & & \\ & & & & & A_{\alpha-2}^{(\alpha-1)} & & & \\ & & & & & & T\cdot A_{\alpha-1}^{(\alpha-1)} & & \\ & & & & & & & A_\alpha^{(\alpha)} \end{bmatrix},$$

où $A_k^{(l)}$ est la matrice $\left(\binom{\sum \mathbf{i}}{j} T^{2(l-j)}\mathbf{a}_2^{(\sum \mathbf{i})-j}(M')^{-j}\right)_{0\leqslant j\leqslant(\sum \mathbf{i})\leqslant l}$, dont les lignes pour lesquelles $\sum \mathbf{i} < k$ ont été effacées. On a $e = 1$ si α est pair, et $e = 0$ sinon. Les indices j des colonnes augmentent de telle sorte que la matrice $A_0^{(l)}$ soit triangulaire inférieure. La notation $\mathbf{a}_2^{(\sum \mathbf{i})-j}$ signifie que l'on prend l'ensemble des monômes apparaissant dans $(\sum_{l_1,l_2\in[|1,\nu|]} a_{2,l_1,l_2})^{i_1+\ldots+i_\nu-j}$, et que chacun de ces monômes donne un coefficient dans la matrice. Par exemple, pour $\nu = 2$, la matrice $A_1^{(2)}$ est :

$$\left(\begin{array}{ccccc|ccc|c} \multicolumn{5}{c|}{j=0} & \multicolumn{3}{c|}{j=1} & j=2 \\ \hline T^4a_{2,1,1} & T^4a_{2,1,2} & T^4a_{2,2,2} & & & \frac{T^2}{M'} & & & \\ & T^4a_{2,1,1} & T^4a_{2,1,2} & T^4a_{2,2,2} & & & \frac{T^2}{M'} & & \\ & & T^4a_{2,1,1} & T^4a_{2,1,2} & T^4a_{2,2,2} & & & \frac{T^2}{M'} & \\ T^4a_{2,1,1}^2 & 2T^4a_{2,1,1}a_{2,1,2} & 2T^4a_{2,1,1}a_{2,2,2}+T^4a_{2,1,2}^2 & 2T^4a_{2,1,2}a_{2,2,2} & T^4a_{2,2,2}^2 & 2a_{2,1,1}\frac{T^2}{M'} & 2a_{2,1,2}\frac{T^2}{M'} & 2a_{2,2,2}\frac{T^2}{M'} & \left(\frac{1}{M'}\right)^2 \end{array}\right),$$

Lemme 48. *Soit $a_2 = \max_{l_1,l_2\in[|0,\nu|]}|a_{2,l_1,l_2}|$. Si on fait tendre α, k et l vers $+\infty$, on a :*

$$\det A_k^{(l)} \leqslant 2^{O(\alpha^{\nu+1})} a_2^{\kappa_5} T^{\kappa_6} M^{\kappa_7},$$

avec :

$$\begin{array}{rcl} \kappa_5 & = & \dfrac{k(l^\nu - k^\nu) + O((l+k)^\nu)}{\nu!}, \\ \kappa_6 & = & \dfrac{2[(l^{\nu+1}-k^{\nu+1}) + (\nu+1)(kl^\nu - lk^\nu)] + O((l+k)^\nu)}{(\nu+1)!}, \\ \kappa_7 & = & -\dfrac{k^{\nu+1} - (\nu+1)kl^\nu + \nu l^{\nu+1} + O((l+k)^\nu)}{(\nu+1)!}. \end{array}$$

Démonstration. On généralise la preuve du lemme 41. En effet, on peut voir que :

$$\det A_k^{(l)} \leqslant \prod_{k\leqslant\sum \mathbf{i}\leqslant l}\left[\left(\frac{1}{M'T^2}\right)^{(\sum \mathbf{i})-k}\left((l+1)^{\frac{\nu}{2}}2^l T^{2l}a_2^k\right)\right],$$

ce qui donne immédiatement le résultat. □

Nous bornons maintenant le déterminant de la matrice $\bar{B}_2$ en utilisant ce dernier lemme :

$$\begin{aligned}\det \bar{B}_2 &= T^{\lfloor \alpha/2 \rfloor} \prod_{i=1}^{\alpha} \det A_i^{\left(\lfloor \frac{\alpha+i}{2} \rfloor\right)} \\ &\leqslant (2^\alpha TNM)^{O(\alpha^{\nu+1})} a_2^{\frac{2^\nu(\nu-1)+1}{2^\nu(\nu+2)!}\alpha^{\nu+2}} T^{\frac{2^\nu(2\nu-1)+1}{2^\nu(\nu+2)!}\alpha^{\nu+2}} M^{-\frac{2^{\nu+1}-\nu-2}{2^{\nu+1}(\nu+2)!}\alpha^{\nu+2}}.\end{aligned}$$

En utilisant la borne sur $\det \bar{B}_1$, on obtient ainsi :

$$\det L' \leqslant (2^\alpha TNM)^{O(\alpha^{\nu+1})} a_2^{\frac{2^\nu(\nu-1)+1}{2^\nu(\nu+2)!}\alpha^{\nu+2}} T^{\frac{2^{\nu-1}(5\nu-2)+1}{2^\nu(\nu+2)!}\alpha^{\nu+2}} M^{-\frac{5\cdot 2^{\nu-1}-\nu-2}{2^{\nu+1}(\nu+2)!}\alpha^{\nu+2}}.$$

Puisque l'on a $a_2 \ll \frac{1}{N}$, l'équation de Coppersmith est de la forme :

$$T^{2^\nu(5\nu-2)+2} \ll M^{5\cdot 2^{\nu-1}-\nu-2} N^{2^{\nu+1}(\nu-1)+2}.$$

On a supposé que l'équation de Taylor (1.5) est satisfaite finement, on obtient donc pour $d = 2$ les paramètres optimaux :

$$\begin{aligned}M &\approx N^{\frac{2^{\nu+1}(2\nu+1)-2}{2^{\nu-1}(10\nu+11)-3\nu-4}}, \\ T &\approx N^{\frac{2^\nu(2\nu+3)-2\nu-2}{2^{\nu-1}(10\nu+11)-3\nu-4}},\end{aligned}$$

ce qui donne un algorithme de complexité $\approx N^{\frac{2^{\nu-1}(6\nu+5)-\nu-2}{2^{\nu-1}(10\nu+11)-3\nu-4}}$. Pour $\nu = 1$, on retrouve bien la complexité $\approx N^{4/7}$ obtenue à la section 1.4. Pour $\nu = 2$, on obtient une complexité de l'ordre de $\approx N^{15/13}$.

1.6.3 Analyse du cas $d \geqslant 3$.

La matrice B à réduire est de la forme :

$$B = \begin{bmatrix} B_1 & 0 \\ B_2 & B_3 \end{bmatrix},$$

où la matrice B_1 est carrée et correspond aux lignes et aux colonnes pour lesquelles on a $0 \leqslant (\sum \mathbf{i}) + dj \leqslant \alpha$. On a $\det L' = \det B_1 \cdot \det B_3$. Le déterminant de la matrice B_1 est simple à calculer :

$$\det B_1 = T^{\kappa_8(\alpha^{\nu+2}+O(\alpha^{\nu+1}))} (M')^{-\kappa_9(\alpha^{\nu+2}+O(\alpha^{\nu+1}))},$$

avec :

$$\begin{aligned}\kappa_8 &= \textstyle\sum_{0\leqslant(\sum \mathbf{i})+dj\leqslant\alpha} (\sum \mathbf{i}) &&= \frac{\nu}{d(\nu+2)!}, \\ \kappa_9 &= \textstyle\sum_{0\leqslant(\sum \mathbf{i})+dj\leqslant\alpha} j &&= \frac{1}{d^2(\nu+2)!}.\end{aligned}$$

Comme dans la section 1.5, on peut utiliser le théorème 28 pour borner le déterminant de la matrice B_3.

Lemme 49. *La matrice B_3 est bornée par le produit des quantités :*

$$\begin{aligned} L_{\mathbf{i},j} &= 2^j (d+1)^{\nu j} N^{(\sum \mathbf{i})+j}, \\ C_{\mathbf{i}',j'} &= \frac{2^{j'} T^{i'}}{M^{j'} N^{(\sum \mathbf{i}')+j'}}. \end{aligned}$$

On utilise donc le théorème 28, ce qui permet d'obtenir :

$$\det B_3 \ll \mathbf{L} \cdot \mathbf{C},$$

où $\mathbf{L}$ est le produit des $L_{\mathbf{i},j}$ pour $(\sum \mathbf{i}) + j \in [|0, \alpha|]$ et $(\sum \mathbf{i}) + dj \in [|\alpha+1, d\alpha|]$, et $\mathbf{C}$ est le produit des $(\dim B_3)$ plus grands $C_{\mathbf{i},j}$ pour $(\sum \mathbf{i}) + dj \in [|\alpha+1, d\alpha|]$. Dans la formule ci-dessus, certains termes sont faciles à évaluer.

Lemme 50. *Avec les notations ci-dessus, on a :*

$$\begin{aligned} \dim B_3 &= \frac{d-1}{d} \frac{1}{(\nu+1)!} (\alpha^{\nu+1} + O(\alpha^\nu)), \\ \mathbf{L} &\ll N^{\left(\nu \frac{d-1}{d} + \frac{d^2-1}{d^2}\right) \frac{1}{(\nu+2)!} (\alpha^{\nu+2} + O(\alpha^{\nu+1}))}. \end{aligned}$$

Démonstration. On a la suite d'égalités suivante :

$$\begin{aligned} \prod_{(\sum \mathbf{i})+j \in [|0,\alpha|], (\sum \mathbf{i})+dj \in [|\alpha+1, d\alpha|]} N^{(\sum \mathbf{i})+j} &= \left(\prod_{(\sum \mathbf{i})+j \leqslant \alpha} N^{(\sum \mathbf{i})+j} \right) \cdot \left(\prod_{(\sum \mathbf{i})+dj \leqslant \alpha} N^{-(\sum \mathbf{i})-j} \right), \\ \prod_{(\sum \mathbf{i})+j \leqslant \alpha} N^{(\sum \mathbf{i})+j} &= N^{\frac{\nu+1}{(\nu+2)!} (\alpha^{\nu+2} + O(\alpha^{\nu+1}))}, \\ \prod_{(\sum \mathbf{i})+dj \leqslant \alpha} N^{-(\sum \mathbf{i})-j} &= N^{-\frac{\nu}{d(\nu+2)!} (\alpha^{\nu+2} + O(\alpha^{\nu+1}))} N^{-\frac{1}{d^2(\nu+2)!} (\alpha^{\nu+2} + O(\alpha^{\nu+1}))}. \end{aligned}$$

□

Il nous reste à évaluer la quantité $\mathbf{C}$. Nous faisons cela exactement comme à la section 1.5. Nous écrivons $\mathbf{C} \ll \prod_{k=\tau\alpha}^{\lfloor k_{\max} \rfloor} 2^{k c_k}$, où c_k est le nombre de $(\nu+1)$-uplets $(\mathbf{i}, j)$ tels que $(\sum \mathbf{i}) + dj \in [|\alpha+1, d\alpha|]$ et :

$$(t-n)\left(\sum \mathbf{i}\right) - (m+n)j \in [k, k+1[,$$

avec $t = \lg T$, $n = \lg N$ et $m = \lg M$. Enfin, le paramètre de sommation τ est choisi minimal tel que $\sum_{k=\tau\alpha}^{\lfloor k_{\max} \rfloor} c_k \geqslant \dim B_3$, c'est-à-dire de telle sorte que l'on ait pris suffisamment de colonnes. Comme on a supposé que la condition de Taylor était finement satisfaite, cela simplifie l'étude par rapport à la section 1.5, car nous n'avons plus que l'équivalent de la première situation à considérer (c'est-à-dire l'équivalent de la figure 1.4). En particulier, on a :

$$k_{\max} = (t-n)(\alpha + O(1)).$$

Pour borner $\mathbf{C}$, on procède en plusieurs étapes : tout d'abord, on estime c_k en fonction de k, puis on détermine τ à partir du fait que l'on connaît la quantité $\dim B_3$, ce qui nous permet finalement de majorer $\mathbf{C}$.

Lemme 51. *Si $k, \alpha \to \infty$, alors on a :*

$$\begin{aligned}
\sum_{l=-\frac{(m+n)\alpha}{d}}^{(t-n)\alpha} c_l &= \frac{-(d(t-n))^\nu + (-m-n)^\nu}{(\nu+1)!d^{\nu+1}(t-n)^\nu}(\alpha^{\nu+1} + O(\alpha^\nu)),\\
c_k &= \frac{k^\nu + O(k^{\nu-1})}{\nu!(t-n)^\nu(m+n)} \quad si\ k \in \left[d(t-n)\alpha, \frac{-(m+n)\alpha}{d}\right],\\
\sum_{l=\lambda\alpha}^{-\frac{(m+n)\alpha}{d}} c_l &= \left(-\frac{(-m-n)^\nu}{(\nu+1)!(t-n)^\nu d^{\nu+1}} - \frac{\lambda^{\nu+1}}{(\nu+1)!(t-n)^\nu(m+n)}\right)(\alpha^{\nu+1} + O(\alpha^\nu)).
\end{aligned}$$

Ce dernier lemme nous permet de déduire facilement les résultats suivants :

Lemme 52. *Si $\alpha \to \infty$, alors on a :*

$$\begin{aligned}
\tau &= -\left((n-t)^\nu(m+n)\right)^{\frac{1}{\nu+1}}\\
\sum_{k=-\frac{(m+n)\alpha}{d}}^{(t-n)\alpha} kc_k &\ll \frac{\nu d(n-t) + (m+n) - (\nu+1)d^2((n-t)^\nu(m+n))^{\frac{1}{\nu+1}}}{(\nu+2)!d^2}.
\end{aligned}$$

À partir de ce dernier lemme, il devient facile de borner le déterminant du réseau L' :

$$\det L' \ll N^{\frac{\nu+1}{(\nu+2)!}(\alpha^{\nu+2}+O(\alpha^{\nu+1}))} T^{\frac{1-\nu}{d(\nu+2)!}(\alpha^{\nu+2}+O(\alpha^{\nu+1}))} 2^{-\frac{\nu+1}{(\nu+2)!}((n-t)^\nu(m+n))^{\frac{1}{\nu+1}}(\alpha^{\nu+2}+O(\alpha^{\nu+1}))}.$$

On obtient ainsi la condition de Coppersmith suivante :

$$n \ll (n-t)^\nu (m+n)^{\frac{1}{\nu+1}}.$$

Si l'on tient compte du fait que la condition de Taylor $m + (d+1)t \leqslant dn$ est finement satisfaite, on obtient les valeurs optimales :

$$\begin{aligned}
T &\approx N^{1-\frac{1}{(d+1)^{\frac{1}{\nu+1}}}},\\
M &\approx N^{(d+1)^{\frac{\nu}{\nu+1}}-1}.
\end{aligned}$$

Cela donne finalement un algorithme de complexité $\approx N^{\frac{\nu}{(d+1)^{\frac{1}{\nu+1}}}}$. On remarque que pour le choix de paramètre « $\nu = 1$ », on retrouve bien les résultats de la section 1.5. Pour $\nu = 2$ et $d = 3$, on obtient une complexité de l'ordre de $\approx N^{2^{1/3}} \approx N^{1.26}$.

1.7 Données expérimentales.

L'algorithme décrit dans ce chapitre a été programmé pour les fonctions d'une seule variable, et est disponible à l'URL `http://www.loria.fr/~stehle/these`. Il a été écrit en C, et utilise les bibliothèques GNU MP [73] et MPFR [147]. La réduction de réseaux est réalisée à l'aide du code décrit au chapitre II-4, qui implante l'algorithme du chapitre II-3.

En ce qui concerne l'implantation de l'algorithme de la figure 1.3 (au paragraphe 1.2.3), les réels sont multipliés par de grandes puissances de 2 puis arrondis, de telle sorte que l'on ne manipule que des entiers. Cela a été expliqué à la section 1.2. Pour la résolution du système de deux équations polynomiales sur les entiers, nous essayons de résoudre le système modulo un petit nombre premier p, et de « relever » les solutions à l'aide du relèvement de Hensel. Si cela échoue, c'est-à-dire si la matrice qui intervient dans le relèvement n'est pas inversible modulo p, on prend un autre nombre premier. On procède comme cela en essayant un petit nombre de nombres premiers, et si le relèvement échoue, on prend une autre paire de vecteurs suffisamment courts dans la base réduite. Une fois que l'on a épuisé toutes les paires de vecteurs suffisamment courts, l'algorithme échoue, et on recommence en divisant la taille du sous-intervalle par 2, comme cela est expliqué à la figure 1.2.

Il convient de souligner que les estimations de temps d'exécution données aux figures 1.7 et 1.8 sont extrêmement approximatives. En effet, de nombreux paramètres interviennent dans l'efficacité pratique de la méthode : précision des calculs, type d'algorithme LLL utilisé, paramètres de réduction pour l'algorithme LLL, ... Un autre facteur important est la taille des dérivées successives dans la partie de l'intervalle $[1/2, 1[$ considéré : si elles sont petites, on peut choisir des sous-intervalles élémentaires plus grands. Le code correspondant à l'algorithme décrit dans le présent chapitre n'est en aucun cas optimisé : il s'agit plutôt d'une preuve de faisabilité, et sert à estimer les ordres de grandeur des temps d'exécution. Cette implantation est avant tout expérimentale.

Un autre code pour l'algorithme avait été écrit auparavant par Paul Zimmermann et est disponible à l'URL `http://www.loria.fr/~zimmerma/free/wclr-1.6.1.tar.gz`. Cette implantation est restreinte au choix de paramètres $d = \alpha = 2$, utilise une réduction de réseaux avec une arithmétique pseudo-rationnelle (les rationnels sont multipliés par des multiples des dénominateurs pour obtenir des entiers), et effectue un calcul de résultant à la place du relèvement de Hensel. Ce code précédent a servi à Paul Zimmermann et à Vincent Lefèvre pour trouver les pires cas pour l'arrondi de la fonction 2^x entre $1/2$ et 1 en double précision étendue[31]. On a par exemple le pire cas suivant pour l'arrondi au plus proche :

$$2^{\frac{1574166561444031150 1}{2^{64}}} = \underbrace{1.110\ldots110}_{64}1\underbrace{0\ldots0}_{63}11\ldots.$$

1.7.1 Recherche des pires cas d'une fonction d'une variable.

Pour la recherche de pires cas pour l'arrondi des fonctions mathématiques, la limite de faisabilité se situe autour de la double précision étendue (64 bits de mantisse), et il apparaît qu'à ce moment, la méthode (avec $d = \alpha = 2$) devienne comparable en efficacité avec l'algorithme de Lefèvre [108]. Pour $n = 64$, il semble que le meilleur choix de paramètres soit $d = 3$ et $\alpha = 2$. À la figure 1.7, nous donnons les temps estimés pour trouver le pire cas de la fonction $\exp x$ pour $x \in [1/2, 1[$, en double précision, en double précision étendue et en quadruple précision, avec à chaque fois les paramètres qui semblent optimaux en

[31] voir l'URL `http://www.loria.fr/equipes/spaces/slz.fr.html`

n	d	α	M	T	temps estimé
53	2	2	2^{53}	$2^{19.85}$	9.1 jours
53	3	2	2^{53}	$2^{20.45}$	7.3 jours
53	3	2	$\mathbf{2^{64}}$	$2^{21.5}$	3.8 jours
64	2	2	2^{64}	$2^{23.95}$	3.2 ans
64	3	2	2^{64}	$2^{24.60}$	2.6 ans
64	3	2	2^{80}	$2^{26.30}$	0.8 an
113	2	2	2^{113}	$2^{42.95}$	$6.9 \cdot 10^9$ ans
113	3	2	2^{113}	$2^{44.45}$	$3 \cdot 10^9$ ans
113	3	3	$\mathbf{2^{128}}$	$2^{48.2}$	$0.95 \cdot 10^9$ ans

FIG. 1.7 – Temps de calcul estimé pour la recherche du pire cas de $\exp x$ sur $[1/2, 1[$, sur un Opteron 2.4GhZ.

pratique. Les estimations sont données pour un Opteron 2.4GhZ. Une grande part du temps d'exécution est consacrée aux réductions de réseaux (environ 45%), le reste étant partagé entre la recherche des racines sur $\mathbb{R}$ (environ 45%) et la construction de la matrice à réduire. Lorsque l'on augmente les paramètres d et α (comme au paragraphe 1.7.3), la réduction de réseaux voit sa part augmenter très rapidement, les deux autres contributions mentionnées ci-dessus devenant négligeables.

Dans les tableaux ci-dessous, nous donnons les bornes de complexité théoriques pour un certain nombre de petits paramètres d et α. Pour un choix de paramètres pour d et α, nous donnons les valeurs $m = \lg M$ et $t = \lg T$ qui optimisent la complexité de la méthode, qui est $\approx 2^{cn}$. Pour $d = 3$, la valeur optimale de M satisfait toujours l'inégalité $M \geqslant N$. Puisque l'on s'attend à avoir une solution à l'équation (1.1) uniquement si $M \lesssim N$, dans les cas où $M > N$, on utilise l'algorithme avec tous les choix possibles des $m - n$ bits superflus ce qui multiplie la complexité par 2^{m-n}. Cela est pris en compte dans les valeurs données pour c qui sont en gras. Pour $d = 3$, cela ne donne pas forcément la meilleure stratégie : il semblerait qu'en ne satisfaisant qu'une des deux conditions de manière fine (et l'autre largement) parmi les conditions de Taylor et Coppersmith, on n'obtient pas la valeur optimale pour M, mais si on a la contrainte $M \leqslant N$ à satisfaire, cela peut donner de meilleures valeurs pour c.

Quand la valeur exacte de c est connue (cela signifie que nous avons étudié le réseau qui intervient et calculé précisément son déterminant), nous la donnons, sinon, nous en donnons une valeur obtenue expérimentalement.
$d = 2$:

α	1	2	3	4	5	10	$\alpha \to \infty$
t	$1/3^{32}$	5/13	13/33	27/67	24/59	71/167	3/7
$\approx t$	0.333	0.385	0.394	0.403	0.407	0.425	0.429
m	1	11/13	9/11	53/67	46/59	121/167	5/7
$\approx m$	1	0.846	0.818	0.791	0.780	0.725	0.714
c	2/3	8/13	20/33	40/67	35/59	96/167	4/7
$\approx c$	0.667	0.615	0.606	0.597	0.593	0.575	0.571
dim	3	6	10	15	21	66	$\approx \alpha^2/2$

$d = 3$:

α	2	3	4	5	6	7	10	$\alpha \to \infty$
t	3/7	13/29	17/37	22/47	–	–	–	1/2
$\approx t$	0.429	0.448	0.459	0.468	0.474	0.479	0.485	0.5
m	9/7	35/29	43/37	53/47	–	–	–	1
$\approx m$	1.284	1.207	1.162	1.128	1.104	1.084	1.060	1
c	**6/7**	**22/29**	**26/37**	**31/47**	–	–	–	1/2
$\approx c$	**0.857**	**0.759**	**0.703**	**0.660**	**0.630**	**0.605**	**0.575**	0.5
dim	6	10	15	21	34	36	66	$\approx \alpha^2/2$

Le choix du degré 3 se justifie d'autant plus si l'on cherche le pire cas d'une fonction pour l'ensemble des exposants possibles (après avoir « ramené » tous les intervalles vers $[1/2, 1[$ par des changements de variable). Cela a en effet l'avantage d'affaiblir la condition $M \lesssim N$: par exemple en double précision, il y a 2^{11} exposants possibles, de telle sorte qu'au lieu d'avoir $M \lesssim 2^{53}$, cette condition devient 2^{64} (il y a 2^{64} entrées possibles). Cela est à nuancer car le temps d'exécution de la méthode dépend fortement de l'exposant choisi. Pour les fonctions classiques, plus l'exposant est petit, mieux les fonctions sont approchées par des polynômes de petits degrés et moins il y a de sous-intervalles à considérer. Pour d'autres fonctions, de nombreux grands exposants sont sans intérêt, comme par exemple les fonctions exponentielles et trigonométriques hyperboliques pour lesquelles on obtient rapidement un dépassement de capacité (l'heuristique liée au « 2^{64} » ci-dessus est alors erronée), ou encore les fonctions trigonométriques qui s'approchent très mal par des polynômes quand l'exposant augmente. La situation peut varier fortement d'une fonction à une autre et d'un exposant à l'autre.

1.7.2 Recherche de pires cas de fonctions de deux variables.

L'étude effectuée à la section 1.6 pourrait être utile pour trouver des mauvais cas de fonctions de deux variables, comme par exemple x^y et $\tan \frac{x}{y}$. Dans cette situation, on s'attend à ce que pour un exposant donné, il y ait une solution à l'équation (1.4) si et seulement si $M \lesssim N^2$, où $N = 2^n$ et n est la précision. Quand nous obtenons une borne $M \geqslant N^2$, il faut alors « deviner » $m - 2n$ bits, de manière similaire au cas d'une seule variable. Cela est pris en compte dans les estimations de complexité en gras. Les tableaux ci-dessous généralisent ceux de la section précédente.

$d = 2$:

α	2	3	4	5	$\alpha \to \infty$
t	2/7	–	–	–	11/26
$\approx t$	0.286	0.295	0.305	0.310	0.423
m	8/7	–	–	–	19/26
$\approx m$	1.143	1.115	1.085	1.040	0.731
c	10/7	–	–	–	4/3
$\approx c$	1.429	1.410	1.390	1.380	1.154
dim	10	20	35	56	$\approx \alpha^3/6$

$d = 3$:

α	2	3	4	5	6	$\alpha \to \infty$
t	2/7	4/13	–	–	–	$1 - 4^{-1/3}$
$\approx t$	0.286	0.308	0.325	0.335	0.345	0.370
m	13/7	23/13	–	–	–	$4^{2/3} - 1$
$\approx m$	1.857	1.769	1.700	1.660	1.620	1.520
c	10/7	18/13	–	–	–	$2^{1/3}$
$\approx c$	1.429	1.385	1.350	1.330	1.310	1.260
dim	10	20	35	56	84	$\approx \alpha^3/6$

$d = 4$:

α	2	3	4	5	6	$\alpha \to \infty$
t	2/7	–	–	–	–	$1 - 5^{-1/3}$
$\approx t$	0.286	0.320	0.340	0.355	0.365	0.415
m	18/7	–	–	–	–	$5^{2/3} - 1$
$\approx m$	2.571	2.400	2.300	2.225	2.175	1.924
c	**2**	–	–	–	–	$2 \cdot 5^{-1/3}$
$\approx c$	**2**	**1.760**	**1.620**	**1.515**	**1.445**	1.170
dim	10	20	35	56	84	$\approx \alpha^3/6$

Pour un réseau de dimension 10, le meilleur choix de paramètres semble être $d = \alpha = 2$, mais le gain de complexité par rapport à la recherche exhaustive n'est peut-être pas très net en simple précision (24 bits de mantisse), d'autant plus que le coût de la réduction de réseaux n'est pas pris en compte dans la complexité annoncée (il est négligeable quand n est grand). Pour $n = 53$, le coût de la méthode est trop élevé. Ainsi, l'intérêt pratique de la méthode décrite n'est pas très net pour la recherche de pires cas de fonctions de plusieurs variables, pour les formats de la norme IEEE-754.

1.7.3 Calculs de bornes supérieures pour l'arrondi correct, et inversion d'une suite de bits.

En quadruple précision (113 bits de mantisse), la recherche des pires cas d'une fonction d'une seule variable sur un intervalle donné est hors de portée, même avec de nombreuses machines qui travailleraient en parallèle. Puisque la complexité de l'algorithme décroît quand la quantité m augmente (jusqu'à devenir polynomiale en n pour $m = \Theta(n^2)$), il devrait être possible de calculer des bornes supérieures sur la qualité des pires cas. Moralement, on devrait pouvoir déterminer un nombre de bits $m > n$ suffisant pour décider l'arrondi correct, mais a priori la borne m sera très grande par rapport à la borne optimale, qui, dans le modèle aléatoire décrit au paragraphe 1.2.1, devrait être de l'ordre de n. Malheureusement, bien que tendant vers une complexité polynomiale, le temps d'exécution requis en pratique reste trop élevé quelque soit la valeur choisie pour m.

Dans le tableau de la figure 1.8, nous nous intéressons à inverser une suite de m bits suivants les 64 premiers, pour un nombre de départ de 64 bits aussi. La tâche qui consiste à inverser une telle suite de bits est exactement la même que celle qui consiste à calculer

n	d	α	M	T	dimensions des bases	temps estimé
64	2	2	2^{64}	$2^{23.95}$	6×9	3.2 ans
64	3	2	2^{64}	$2^{24.60}$	6×12	2.6 ans
64	3	2	2^{100}	$2^{26.95}$	6×12	0.5 an
64	4	2	2^{120}	$2^{29.55}$	6×15	51 jours
64	5	3	2^{150}	$2^{32.80}$	10×34	25 jours
64	8	3	2^{210}	$2^{35.95}$	10×52	6 jours
64	10	4	2^{280}	$2^{39.30}$	15×105	5.5 jours
64	16	5	2^{400}	$2^{43.55}$	21×261	2.1 jours

FIG. 1.8 – Temps de calcul estimé pour l'inversion d'une suite de bits générée par la fonction $\exp x$ sur $[1/2, 1[$ pour différents paramètres, sur un Opteron 2.4GhZ.

une borne supérieure pour l'arrondi. Nous estimons le temps de calcul qu'il faudrait sur un Opteron 2.4GhZ pour différents types de bornes et de paramètres.

1.8 Problèmes ouverts.

Plusieurs questions naturelles se posent vis-à-vis de l'algorithme que nous venons d'exposer. Tout d'abord, comme plus généralement dans le cas de la méthode de Coppersmith modulaire multivariée, il serait très intéressant de savoir quand l'heuristique sur la non-nullité du résultant va être vérifiée (ou ne pas l'être), autrement dit de prouver la méthode, ou de donner des conditions sous lesquelles la méthode fonctionne toujours. En effet, notre algorithme échoue dans certaines situations. En particulier, si l'on considère la fonction $f(x) = \frac{N}{2^n x}$ où $n = \lfloor \lg N \rfloor$ et N est un module RSA, en prenant $N_1 \approx N_2 \approx 2^{n/2}$, et M arbitrairement grand, on obtiendrait un algorithme qui factorise N en temps polynomial. Sur cette instance particulière, notre méthode fonctionne pour de petites valeurs de d et α, puis le résultant calculé devient toujours nul, ce qui rend incorrecte l'analyse de complexité. On observe pour cette fonction que l'algorithme LLL renvoie des vecteurs significativement plus courts que $(\det L)^{\frac{1}{\dim L}}$, où L est le réseau réduit. Les réseaux qui apparaissent ici ne sont pas du tout « aléatoires ». Dans le cas de la fonction racine carrée, la méthode échoue aussi. Cela est incontournable à cause du nombre trop important de solutions. Par exemple, pour tout nombre pair n_1, tout $n_2 \geqslant n_1/2$ et tout nombre x_0 de la forme $\frac{x^2}{2^{n_1}}$ avec $x \in [|2^{\frac{n_1-1}{2}}, 2^{\frac{n_1}{2}}|]$, on a $2^{n_2}\sqrt{x_0} = 0 \mod 1$: il y a environ $2^{\frac{n_1}{2}}$ telles solutions, qui ne peuvent pas être calculées en temps polynomial. Plus généralement, notre méthode semble plutôt mal fonctionner avec les fonctions algébriques.

Du point de vue de la méthode elle-même, une question centrale est de savoir si les bornes de Coppersmith que nous avons calculées sont les plus fines possibles. Elles semblent être fines pour de petits paramètres d et α car nous avons vérifié en pratique que l'on n'obtenait pas de vecteur nettement plus court que nous le prédisent les bornes, mais il est possible qu'elles puissent être améliorées pour de plus grands paramètres. En particulier, il est surprenant que dans le cas où $d = 2$ on puisse obtenir de meilleures

bornes avec la méthode que nous avons exposée dans la section 1.4 qu'avec la méthode que nous avons utilisée pour $d \geqslant 3$. Cela nous amène à penser que la technique développée pour $d \geqslant 3$ ne donne peut-être pas les meilleures bornes possibles. Deux arguments vont à l'encontre de la possible non-optimalité de la borne asymptotique $m = O(n_1 + n_2)^2$. D'une part, cette borne est du même ordre de grandeur que celle obtenue par Nesterenko et Waldschmidt [135], comme nous l'avons annoncé en introduction de chapitre. D'autre part, si les réseaux réduits possédaient des vecteurs plus courts que ceux prédits en théorie, l'algorithme LLL le détecterait (il calcule une approximation du premier minimum du réseau donné en entrée), et nous n'avons rien observé de tel en pratique. Ce deuxième argument est faible car le temps d'exécution est très vite trop élevé pour étudier le comportement asymptotique de l'algorithme.

Il est aussi légitime de s'intéresser à généraliser les méthodes développées dans ce chapitre pour les mauvais cas simultanés de deux fonctions f_1 et f_2. Nous abordons cette question au chapitre suivant, mais dans la limite où les résultats obtenus peuvent être exploités pour l'application que l'on considère alors, c'est-à-dire la technique des tables de Gal.

D'un point de vue plus pratique, il serait intéressant de voir comment coder efficacement la méthode, par exemple en utilisant un code de réduction de réseaux conçu précisément pour les réseaux issus de la méthode de Coppersmith, ou en utilisant d'autres approximations polynomiales que celles données par les développements de Taylor. On pourrait aussi essayer de réutiliser une réduction de réseaux faite pour un intervalle sur l'intervalle suivant, en stockant la matrice de transformation. Une autre idée serait d'optimiser le passage d'un intervalle au suivant lorsqu'on trouve un vecteur plus court que prévu, en ajustant le paramètre T après l'étape de réduction. De nombreuses améliorations pratiques heuristiques semblent possibles. Un code qui serait significativement plus rapide que le code existant permettrait de faire plus de tests pour mieux comprendre le comportement de l'algorithme : bien que celui-ci soit de complexité polynomiale pour certains choix de paramètres, le temps d'exécution reste trop élevé pour qu'une étude expérimentale intensive puisse être entamée.

Du point de vue de la recherche des pires cas pour l'arrondi des fonctions mathématiques, notre algorithme apporte des améliorations par rapport à la méthode de Lefèvre, mais pas suffisamment importantes pour pouvoir s'intéresser à la quadruple précision. Est-il possible de construire un autre algorithme ou de modifier celui-ci pour que cela devienne réalisable ? Une autre question importante que notre algorithme ne résout pas est celle des pires cas pour l'arrondi de fonctions dont les dérivées peuvent être particulièrement grandes, comme c'est le cas pour $\sin x$ et $\cos x$ pour de grands exposants, une fois effectué le changement de variables transformant l'intervalle de départ en $[1/2, 1[$.

Chapitre III-2

Amélioration de la méthode de Gal.

Le travail de recherche correspondant à ce chapitre a fait l'objet d'une publication dans les actes de la conférence ARITH'17 (17th Symposium on Computer Arithmetic) [170]. L'idée expliquée au début du paragraphe 2.2.2, qui consiste à prendre une précision d'entrée différente de la précision de sortie pour les éléments de la table, a été suggérée par David Defour. Elle nous permis de diminuer la taille des tables par rapport à la version publiée [170]. Le présent chapitre est ainsi une version améliorée de [170].

Sommaire

2.1 La méthode classique de Gal. 207
2.1.1 Le modèle aléatoire. 207
2.1.2 La méthode classique de Gal. 207
2.2 Amélioration de la méthode de Gal. 209
2.2.1 Le cas d'une seule fonction associée. 209
2.2.2 Le cas de deux fonctions associées. 210
2.3 Le calcul des tables. 214
2.3.1 Recherche des mauvais cas simultanés de deux fonctions. . 215
2.3.2 Correction de l'algorithme. 217
2.3.3 Analyse de complexité de l'algorithme. 218
2.4 Résultats expérimentaux. 219
2.4.1 La première table pour 2^x. 219
2.4.2 La table pour $\sin x$. 220
2.4.3 La table pour 2^x et $\ln 2 \cdot 2^x$. 220
2.5 Généralisation possible. 221

La méthode des tables de Gal, décrite initialement dans [62] puis détaillée dans [63], est une technique répandue [55, 121, 134] pour implanter efficacement les fonctions mathématiques en précision fixe, en particulier en double précision. Elle permet d'obtenir des valeurs très souvent correctement arrondies, sans utiliser, dans les calculs intermédiaires, une précision supérieure à la précision d'entrée-sortie de la fonction. La mise en place de cette méthode nécessite un gros précalcul d'une table de valeurs prises en des points distingués par la fonction considérée — ou par plusieurs fonctions associées. Nous

améliorons la méthode de Gal de deux manières. D'une part, nous décrivons ce qui est vraisemblablement le meilleur ensemble de points distingués et expliquons en quoi cet ensemble permet d'obtenir une meilleure implantation de la fonction, des points de vue de l'efficacité et de la qualité de ses sorties. D'autre part, nous donnons un algorithme considérablement plus efficace que celui de Gal pour construire les tables en question. Ces améliorations sont liées au dilemme du fabricant de tables (voir le chapitre III-1) : les tables vont être constituées de mauvais cas pour l'arrondi des fonctions mathématiques et les algorithmes pour les construire sont liés à l'algorithme décrit au chapitre III-1. Nous prouvons expérimentalement la faisabilité des méthodes décrites ici, en nous intéressant particulièrement aux fonctions 2^x, $\sin x$ et $\cos x$, pour $x \in [1/2, 1[$, en double précision (53 bits de mantisse). Nous ne nous attardons pas sur l'implantation des fonctions à partir de ces tables, le but du chapitre étant plutôt de montrer comment construire de bonnes tables efficacement.

Lorsque l'on veut implanter une fonction mathématique sur tout son domaine pour une précision d'entrée-sortie donnée, on partage souvent le calcul en deux phases : par exemple, c'est la stratégie adoptée dans [51] pour la fonction exponentielle, ou plus généralement dans la bibliothèque Crlibm [177]. La figure 2.1 illustre cette stratégie. La première phase, appelée *phase rapide*, renvoie le résultat correctement arrondi pour une très grande majorité des entrées en un laps de temps aussi court que possible. La seconde phase est la *phase précise*. Elle n'est utilisée que lorsque la première phase ne s'est pas avérée suffisante pour garantir l'arrondi correct de la sortie, ce qui advient peu souvent. Elle est plus coûteuse car elle fait intervenir des calculs en précision plus élevée et souvent plus élaborés, mais elle permet d'obtenir un résultat correctement arrondi, quel que soit le nombre donné en entrée. Elle repose souvent sur la stratégie en peaux d'oignon de Ziv [191], qui consiste à étendre la précision des calculs jusqu'à ce que l'on puisse garantir l'arrondi correct. Quand elles sont connues, comme par exemple celles obtenues dans [107, 109, 110], les bornes sur les pires cas permettent d'être certain que la stratégie de Ziv fonctionnera[33], et souvent qu'une seule extension de précision suffira.

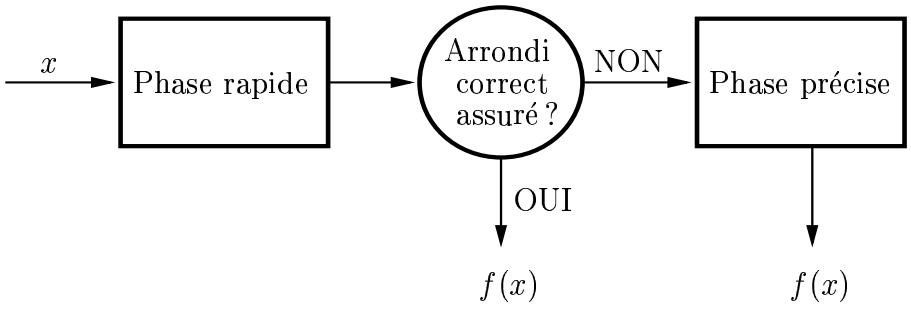


FIG. 2.1 – Les deux phases de l'évaluation de $f(x)$.

En général, la phase rapide utilise comme précision de calcul la précision d'entrée-sortie, ou éventuellement une précision étendue dans un nombre très restreint d'étapes.

[33]Elle aurait pu ne pas finir si $f(x)$ était un nombre représentable pour une certaine entrée x valide.

Elle est elle-même fréquemment divisée en quatre phases, comme illustré à la figure 2.2 :

- **Première réduction d'argument.** On restreint l'implantation de la fonction à un petit intervalle, en utilisant les propriétés mathématiques de la fonction considérée. Par exemple, la formule $\exp(x+k\ln 2) = 2^k \cdot \exp(x)$ permet de se restreindre à $[0, \ln 2[$ pour $\exp x$, la formule $2^{x+k} = 2^k \cdot 2^x$ permet de se restreindre à $[0, 1[$ pour 2^x, et la formule $\sin(x + k\frac{\pi}{2}) = f_k(x)$ avec $f_k = \pm\sin x$ ou $f_k = \pm\cos x$ suivant la valeur de k modulo 4, permet de se restreindre à l'intervalle $[-\frac{\pi}{4}, \frac{\pi}{4}[$ pour $\sin x$.
- **Deuxième réduction d'argument.** À l'aide d'une table, l'intervalle considéré est de nouveau réduit. Par exemple, nous écrivons $2^x = 2^{x_0} \cdot 2^h$ (respectivement $\sin x = \sin x_0 \cdot \cos h + \cos x_0 \cdot \sin h$), où $(x_0, 2^{x_0})$ (respectivement $(x_0, \sin x_0, \cos x_0)$) appartient à une table précalculée, et $h = x - x_0$ est petit. Intuitivement, la borne sur h est la longueur de l'intervalle de départ divisée par le nombre d'éléments de la table. Nous appelons *fonctions associées* les fonctions utilisées dans cette deuxième réduction d'argument. En reprenant les exemples ci-dessus, les fonctions associées sont 2^x pour 2^x, et $\sin x$ et $\cos x$ pour $\sin x$.
- **Évaluation polynomiale.** Les termes restants — c'est-à-dire 2^h, $\cos h$ et $\sin h$ dans nos exemples — sont calculés de façon approchée en évaluant des polynômes qui approchent les fonctions associées sur l'intervalle restreint.
- **Reconstruction.** Enfin, on collecte et regroupe les termes calculés aux trois étapes précédentes pour renvoyer une approximation de la valeur à calculer.

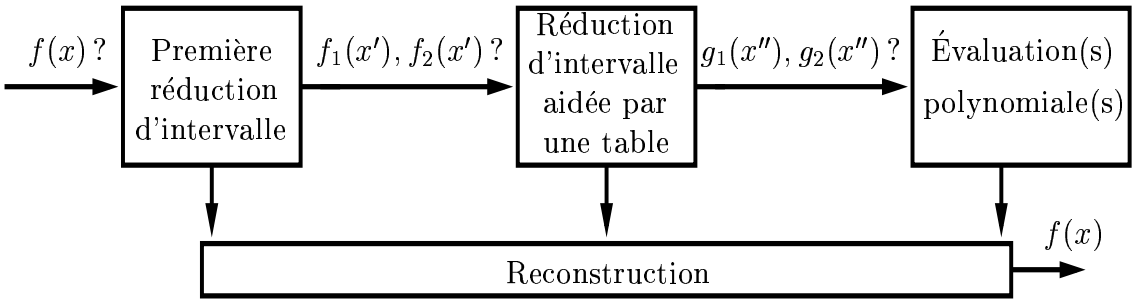


FIG. 2.2 – La phase rapide.

La première étape a été étudiée par exemple dans [36, 145], et des réponses très précises sont apportées pour le choix du polynôme de la troisième étape dans [37]. La méthode des tables de Gal concerne à la deuxième étape. L'idée est de construire une table de points presque régulièrement espacés dont les images par la ou les fonctions associées sont extrêmement proches de nombres représentables. Si les points sont régulièrement espacés, la borne supérieure sur h est faible. Si on s'autorise à ce qu'ils soient moins régulièrement espacés, on peut augmenter la qualité de la table. Il y a donc un compromis à réaliser. Dans l'article de Gal, la table fait quelques kilo-octets (pour des raisons de taille de cache [50]) et est calculée à l'aide d'une recherche exhaustive naïve.

Nous ne nous intéressons dans ce chapitre qu'à la deuxième réduction d'argument.

En particulier, nous n'expliquons pas comment obtenir une implantation correcte pour l'ensemble des exposants possibles à partir d'une implantation correcte pour l'intervalle obtenu après la première réduction d'argument. Cela ne semble pas être une tâche facile (sauf pour les exposants positifs de la fonction 2^x car la formule $2^{x+k} = 2^k \cdot 2^x$ est exacte), et se situe hors de notre propos. De la même manière, nous ne cherchons pas à optimiser l'étape d'approximation polynomiale : dans tous nos exemples, les polynômes que nous choisissons sont les débuts des développements de Taylor des fonctions considérées. Ce n'est vraisemblablement pas le meilleur choix possible, mais a l'avantage de ne pas ajouter de difficultés à l'exposé de notre méthode. Le but de ce chapitre est de montrer que l'on peut améliorer la qualité des tables utilisées lors de la deuxième réduction d'argument. Les choix des fonctions étudiées, des précisions choisies et des tailles des tables peuvent sembler arbitraires : ce sont des illustrations pratiques de la faisabilité de la méthode.

Nous améliorons la méthode de Gal de deux façons. Tout d'abord, nous décrivons le lien qui existe entre les tables de Gal et le dilemme du fabricant de tables. En effet, la table est constituée de mauvais cas pour les fonctions associées pour les arrondis dirigés (vers 0 ou $\pm\infty$). Ce lien nous permet de définir ce que sont les « meilleures » tables de Gal. La deuxième amélioration est la façon dont sont calculées les tables. La recherche naïve de Gal aurait un coût prohibitif pour obtenir les tables que nous désirons. Dans le cas d'une seule fonction associée, on peut utiliser directement l'algorithme de recherche de pires cas décrit au chapitre précédent, ou l'algorithme de Lefèvre [107]. Dans le cas de plusieurs fonctions associées, nous proposons un algorithme inspiré de celui du chapitre précédent, qui permet de trouver des pires cas simultanés de plusieurs fonctions.

Bien que la méthode soit générale, nous nous concentrerons sur les fonctions 2^x et $\sin x$ sur $[1/2, 1[$, en double précision (53 bits de mantisse). Pour la fonction 2^x, nous proposons deux alternatives. Pour ces choix, nos améliorations par rapport à la méthode de Gal sont les suivantes :

1. Les tables pour $\sin x$ et la deuxième alternative pour 2^x sont construites nettement plus efficacement. Avec la méthode naïve, le calcul d'une table de cette qualité aurait coûté plus de 2^{62} appels à $\sin x$ et $\cos x$ en précision étendue pour $\sin x$ et 2^{62} appels à 2^x en précision étendue pour 2^x. En comparaison, nous avons construit des tables de cette qualité en moins d'une journée sur un Opteron 2.4 GHz.
2. Pour $\sin x$, la proportion d'entrées pour lesquelles la phase rapide échoue chute de $\approx 2^{-10}$ à $\approx 2^{-20}$. Pour la première alternative de 2^x, nous pouvons faire reposer la phase précise aussi sur la méthode de Gal, en n'utilisant que la quadruple précision. Pour la deuxième alternative de 2^x, nous faisons décroître de $\approx 2^{-10}$ à $\approx 2^{-20}$ la proportion d'entrées pour lesquelles la phase rapide échoue.

Dans la section 2.1 nous décrivons la méthode classique des tables de Gal, dont nous proposons les améliorations mentionnées ci-dessus à la section 2.2. Nous décrivons nos algorithmes pour construire les tables à la section 2.3, et donnons des résultats expérimentaux à la section 2.4. Enfin, à la section 2.5 nous discutons le cas de plus de deux fonctions associées.

2.1 La méthode classique de Gal.

Avant de donner notre amélioration de la méthode de Gal, nous expliquons dans quel modèle aléatoire nous nous plaçons et rappelons cette méthode.

2.1.1 Le modèle aléatoire.

Nous nous plaçons dans le même modèle aléatoire qu'au chapitre III-1 : nous supposons que d'un point de vue statistique la fonction étudiée se comporte comme une boîte noire aléatoire uniforme, à partir du moment où les bits les plus significatifs ont été écartés.

Nous généralisons ce modèle au cas de deux fonctions $f_1, f_2 : [1/2, 1[\to [a, b[$ pour des réels $a < b$, par exemple $\sin x$ et $\cos x$, ou 2^x et $\ln 2 \cdot 2^x$. Ces deux fonctions sont modélisées par des boîtes noires aléatoires indépendantes : elles sont toutes les deux des boîtes noires, et pour tous n et $k_1 \neq k_2$ suffisamment grands, si x est choisi aléatoirement et uniformément dans $[1/2, 1[_n$ (où n est la précision d'entrée-sortie), alors le k_1-ième bit de l'écriture binaire de $f_1(x)$ et le k_2-ième bit de l'écriture binaire de $f_2(x)$ sont des variables aléatoires uniformes indépendantes. Dans ce modèle, on a les propriétés suivantes :

- La probabilité d'obtenir des suites de p bits identiques — à partir d'un rang donné — dans les écritures binaires de $f_1(x)$ et $f_2(x)$ est 2^{2-2p} (il y a quatre possibilités : deux suites de zéros, deux suites de uns, une suite de zéros et une suite de uns, et réciproquement).
- Si nous considérons 2^{2p} entrées x (par exemple des entrées consécutives), alors il y en a $O(1)$ parmi elles pour lesquelles les écritures binaires de $f_1(x)$ et $f_2(x)$ ont toutes les deux p bits consécutifs identiques à partir d'un rang donné.
- L'ensemble des $x \in [1/2, 1[_n$ tels que les écritures binaires de $f_1(x)$ et $f_2(x)$ ont p bits identiques à partir d'un certain rang est de cardinalité proche de 2^{n-2p}. L'espérance de la distance maximale entre deux éléments consécutifs dans cet ensemble est inférieure à $\approx n2^{n-2p}$.

Les preuves de ces faits sont soit immédiates, soit identiques à celles du cas d'une seule fonction (voir le chapitre III-1). Il est facile de généraliser ce modèle à plus de deux fonctions.

Ces hypothèses peuvent paraître suspectes quand on considère la paire de fonctions 2^x et $\ln 2 \cdot 2^x$, qui sont linéairement liées, et aussi pour la paire de fonctions $\sin x$ et $\cos x$, qui vérifient la relation $\sin^2 x + \cos^2 x = 1$. D'une part, remarquons que nous ne demandons qu'une indépendance statistique bit à bit et non une indépendance calculatoire des chaînes de bits (on ne pourrait pas déterminer efficacement qu'il n'y a pas indépendance, ce qui serait une hypothèse beaucoup plus forte). D'autre part, les expériences effectuées et décrites à la section 2.4 tendent à valider cette hypothèse.

2.1.2 La méthode classique de Gal.

La méthode de Gal [62, 63] concerne la seconde réduction d'argument de la phase rapide du calcul de $f(x)$. La méthode est générale, mais pour simplifier nous nous concentrons sur les exemples $f(x) = 2^x$ et $f(x) = \sin x$. La seconde réduction d'argument repose

sur les identités suivantes :

$$\begin{array}{lrcl} (a) & 2^x & = & 2^{x_0} \cdot 2^h, \\ (b) & \sin x & = & \sin x_0 \cdot \cos h + \cos x_0 \cdot \sin h, \\ (c) & 2^x & = & 2^{x_0} + (\ln 2 \cdot 2^{x_0}) \cdot h + 2^{x_0} \cdot \left(2^h - 1 - \ln 2 \cdot h\right), \end{array}$$

où : x_0 est dans la table et est stocké avec des approximations de 2^{x_0}, respectivement $\sin x_0$ et $\cos x_0$, et respectivement 2^{x_0} et $\ln 2 \cdot 2^{x_0}$; $h = x - x_0$ est petit, c'est-à-dire de l'ordre de la largeur de l'intervalle de départ, divisée par le nombre de points distingués. Plus généralement, on peut utiliser la formule suivante pour la réduction d'argument :

$$f(x) = f(x_0) + f'(x_0) \cdot h + h^2 \cdot g(h, x_0),$$

où g est une fonction auxiliaire et la table contient les points distingués x_0 et des valeurs approchées de $f(x_0)$ et $f'(x_0)$.

Gal ne traite pas explicitement la réduction (c). Nous verrons que par rapport à la réduction (a), elle permet d'obtenir des résultats corrects plus souvent. Remarquons aussi que dans ce cas, on peut se dispenser de stocker l'approximation de $\ln 2 \cdot 2^{x_0}$ car on peut la retrouver à partir du nombre flottant en double précision approchant le mieux $\ln 2$ et de l'approximation stockée pour 2^{x_0} (voir l'annexe du chapitre).

Après la réduction d'argument effectuée grâce à la table, les quantités restantes sont évaluées à l'aide de polynômes de petits degrés qui les approchent suffisamment : il s'agit des quantités 2^h, respectivement $\sin h$ et $\cos h$, et respectivement $2^h - 1 - \ln 2 \cdot h$, pour h proche de 0. Pour simplifier les détails, nous supposerons que ces polynômes sont les débuts des développements de Taylor des quantités qu'ils approchent, mais il est possible de choisir de meilleurs polynômes [37]. Puisque nous supposons que la précision d'entrée-sortie de la fonction à implanter est la double précision, les calculs doivent être effectués le plus possible en double précision. Pour des raisons de taille de cache [50], la taille des tables ne doit pas dépasser quelques kilo-octets.

L'approche naïve pour construire la table est de prendre des points distingués régulièrement espacés, ce qui a l'avantage de minimiser la borne sur $|h|$ (pour une taille de table fixée). La méthode de Gal consiste à relâcher légèrement cette condition d'optimalité pour se permettre de prendre de meilleurs points distingués : Gal choisit des points distingués x_0 presque régulièrement espacés, qui sont tels que les approximations tabulées de 2^{x_0}, respectivement $\sin x_0$ et $\cos x_0$, respectivement 2^{x_0} et $\ln 2 \cdot 2^{x_0}$, sont extrêmement proches des valeurs qu'elles approchent. Les valeurs tabulées en double précision « simulent » ainsi une précision plus grande. D'un point de vue pratique, pour tout $0 \leqslant i < 2^{10}$, il choisit — ou choisirait puisqu'il ne considère pas la deuxième possibilité pour la fonction 2^x — $x_0^{(i)}$ comme le nombre flottant en double précision le plus proche de $\frac{1}{2} + \frac{i}{2^{11}}$ qui satisfait :

$$|2^{52} \cdot 2^{x_0^{(i)}} \bmod 1| \leqslant 2^{-10} \text{ pour la première variante de } 2^x,$$

$$|2^{53+e} \cdot \sin x_0^{(i)} \bmod 1|, |2^{53} \cdot \cos x_0^{(i)} \bmod 1| \leqslant 2^{-10} \text{ pour } \sin x,$$

$$|2^{52} \cdot 2^{x_0^{(i)}} \bmod 1|, |2^{52+e} \cdot \ln 2 \cdot 2^{x_0^{(i)}} \bmod 1| \leqslant 2^{-10} \text{ pour la deuxième variante de } 2^x,$$

où, dans la deuxième équation, on a $e = 1$ si $|x_0^{(i)}| \leqslant \frac{\pi}{6}$ et $e = 0$ sinon ; et dans la troisième équation, on a $e = 1$ si $|x_0^{(i)}| \leqslant -\lg \ln 2$ et $e = 0$ sinon. Si l'on se place dans le modèle aléatoire décrit au paragraphe précédent, le point distingué $x_0^{(i)}$ devrait être extrêmement proche de $\frac{1}{2} + \frac{i}{2^{11}}$, ce qui implique une augmentation négligeable de la borne sur $|h|$. Par ailleurs, ces points distingués donnent une meilleure implantation de la fonction : puisque $|x - x_0|$ est borné par $\approx 2^{-12}$, la valeur renvoyée à la fin pour $f(x)$ peut être calculée avec environ 63 bits de précision simulée tout en ne faisant les calculs qu'avec des nombres flottants en double précision : par exemple pour la première variante de 2^x, on calcule $2^h - 1$ en double précision, ce qui donne une valeur correcte à $\approx 2^{-63}$ près (en précision absolue) car h est petit ; on multiplie cette quantité par 2^{x_0}, ce qui donne une valeur approchée à $\approx 2^{-63}$ près de $2^{x_0} \cdot (2^h - 1)$; finalement, on rajoute à cela 2^{x_0}, pour obtenir un résultat correctement arrondi, sauf dans le cas où les ≈ 10 derniers bits de $2^{x_0} \cdot (2^h - 1)$ sont tous identiques. Toujours dans le modèle aléatoire, le résultat renvoyé pour $f(x)$ est correctement arrondi avec probabilité proche de $1 - 2^{-10}$, ce qui est supérieur à 99.9%.

Les tables sont construites par recherche exhaustive, de telle sorte qu'il y a environ 2^9 essais pour obtenir chaque élément distingué dans la première situation, et environ 2^{18} essais pour chaque élément distingué dans les deux autres situations. La recherche exhaustive est donc possible pour de telles tables.

2.2 Amélioration de la méthode de Gal.

Nos améliorations de la méthode de Gal reposent essentiellement sur le fait que l'on peut exiger mieux que 10 bits consécutifs identiques : dans le cas d'une fonction associée unique, on peut exiger une quarantaine de bits identiques[34] et dans le cas de deux fonctions associées, il est possible d'exiger 25 bits identiques. Nous décrivons dans un premier temps ce que l'on peut faire avec une meilleure table dans le cadre d'une seule fonction associée, puis nous nous intéresserons au cas de deux fonctions associées.

2.2.1 Le cas d'une seule fonction associée.

Dans le cas d'une seule fonction associée, comme par exemple pour la première alternative pour 2^x, la méthode de Gal peut aussi être utilisée pour la phase précise de l'évaluation de $f(x)$. Nous décrivons ici ce que pourrait être une implantation de la fonction $2^x : [1/2, 1[\to [1, 2[$ en double précision avec arrondi correct pour les arrondis dirigés. Comme d'habitude, nous avons une phase rapide et une phase précise. Cette dernière est utilisée lorsque la phase rapide ne s'avère pas suffisante pour déterminer l'arrondi correct de $f(x)$.

La phase rapide. Nous utilisons la méthode de Gal avec une table constituée des 42-mauvais cas pour 2^x sur $[1/2, 1[$, c'est-à-dire les nombres flottants en double précision x_0

[34] En utilisant l'idée consistant à fixer les 10 premiers bits comme dans le cas de deux fonctions associées, on pourrait exiger jusqu'à une cinquantaine de bits identiques. Cependant, cette amélioration aurait son effet annulé par l'erreur numérique provenant de l'évaluation polynomiale, pour la phase rapide, et ne présenterait aucun avantage supplémentaire lors de la phase précise.

qui satisfont (voir le chapitre III-1) :

$$\left|2^{52} \cdot 2^{x_0} \bmod 1\right| \leqslant 2^{-42}.$$

La table contient les couples $(x_0, \diamond(2^{x_0}))$, où l'on a $|2^{x_0} - \diamond(2^{x_0})| \leqslant 2^{-94}$. Il y a environ 2^{12} éléments distingués dans la table et le calcul est réduit à l'évaluation approchée de 2^h où $|h| \leqslant 2^{-10.914}$ (cette valeur est justifiée à la section 2.4). Pour cette dernière évaluation, nous approchons 2^h par le polynôme $P(h) = 1 + a_1 h + \ldots + a_4 h^4$, avec a_i un nombre flottant en double précision le plus proche de $\frac{(\ln 2)^i}{i!}$. Il est possible de vérifier que $P(h)$ peut être évalué, avec des opérations uniquement en double précision, avec une erreur bornée par $\approx 2^{-63}$ (en utilisant par exemple le schéma de Horner). Cela implique que si x n'est pas un 10-mauvais cas pour 2^x, la valeur calculée pour 2^x dans la phase rapide est correcte. De façon amusante, si x est un 42-mauvais cas, le résultat peut aussi être correctement arrondi : pour chaque entrée de la table $(x_0, \diamond(2^{x_0}))$, on ajoute un bit d'information qui dit si la valeur stockée pour 2^{x_0} est légèrement plus grande ou légèrement plus petite que 2^{x_0}.

La phase précise. Supposons désormais que la phase rapide n'ait pas suffit pour calculer l'arrondi dirigé de 2^x. Nous gardons la même table, avec les mêmes valeurs distinguées, et ne changeons que la phase d'évaluation polynomiale. Nous prenons le polynôme $Q(h) = 1 + b_1 h + \ldots + b_7 h^7$ où b_i est un nombre flottant en quadruple précision le plus proche de $\frac{(\ln 2)^i}{i!}$. Il est possible de vérifier que $Q(h)$ peut être évalué, avec des opérations en quadruple précision, pour donner une approximation de 2^h à $2^{-106.821}$ près quand $|h| \leqslant 2^{-10.914}$. Soit $\bar{Q}$ la valeur calculée pour $Q(h)$. On approche finalement 2^x par le produit $\diamond(2^{x_0}) \cdot \bar{Q}$, effectué en quadruple précision. L'erreur commise est bornée par :

$$2^{-106.821+1} + 2^{-94} \cdot 1.00036 + 2^{-114+1} \leqslant 2^{-93.999}.$$

Cela implique que si x n'est pas un 41.999-mauvais cas, alors le résultat est arrondi correctement. Au bilan, nous avons une implantation de 2^x qui renvoie le résultat correct pour toutes les entrées x qui sont des 41.999-mauvais cas. Nous réglons ce petit inconvénient de la manière suivante. Nous considérons une table faite des 42.00072-mauvais cas au lieu des 42-mauvais cas, ce qui signifie que l'on enlève $173f930a6f9c23/2^{53}$ de la table et le considérons comme un cas particulier. Ceci ne change pas la borne sur $|h|$. De plus, il est facile d'observer que l'analyse d'erreur ci-dessus donne maintenant une borne de 2^{-94}. Ainsi, le résultat final est correctement arrondi car nous sommes dans l'une des trois situations suivantes :

- l'entrée x n'est pas un 42-mauvais cas, et la borne d'erreur implique que la valeur renvoyée pour l'arrondi dirigé de 2^x est correcte,
- l'entrée x est un 42.000072-mauvais cas, et la sortie est stockée dans la table (avec un bit supplémentaire pour savoir si l'arrondi a été fait vers le haut ou le bas),
- l'entrée x est exactement $173f930a6f9c23/2^{53}$ (avec ici aussi un bit supplémentaire pour savoir si l'arrondi a été fait vers le haut ou le bas).

2.2.2 Le cas de deux fonctions associées.

Sous l'hypothèse du modèle aléatoire, il est aussi possible de renforcer les exigences de la méthode de Gal sur les longueurs des suites de bits identiques, pour le cas de deux

fonctions associées. Avant tout, nous discutons la façon dont x_0 est choisi à partir de x. Du point de vue de la précision, le mieux est de prendre x_0 le plus proche possible de x parmi les points distingués (car cela diminue la borne sur $|h|$). Il est cependant plus efficace de prendre l'élément de la table qui a les mêmes bits les plus significatifs que x : à partir de x, on prend par exemple les 10 bits les plus significatifs sauf le premier ce qui donne un entier de 9 bits, qui est le numéro de l'entrée de la table qui va donner x_0. Cela est plus efficace que de chercher l'élément x_0 le plus proche de x, car il faudrait alors effectuer plusieurs comparaisons. Puisque les 10 premiers bits de l'entrée de la table sont déjà déterminés, il n'y a pas besoin de les représenter, et on peut donc prendre des éléments distingués $x_0 \in [1/2, 1[_{63}$. On stockera en fait $x_0 - \frac{2i+1}{2^{11}}$, où i est l'entier donné par les 10 premiers bits de x_0. La quantité stockée peut en effet être représentée exactement par un nombre flottant en double précision.

Nous considérons ici les cas de $\sin x$ — avec pour fonctions associées $\sin x$ et $\cos x$ — et la deuxième variante de 2^x — avec pour fonctions associées 2^x et $\ln 2 \cdot 2^x$. Si l'on se place dans le cadre du modèle aléatoire, on peut obtenir une table de bonne taille si les x_0 sont des 25-mauvais cas simultanés pour les deux fonctions (car $25 < (63-10)/2$), c'est-à-dire les $x_0 \in [1/2, 1[_{63}$ tels que :

$$\begin{aligned} |2^{53+e} \cdot \sin x_0 \bmod 1|, |2^{53} \cdot \cos x_0 \bmod 1| &\leqslant 2^{-25}, \\ |2^{52} \cdot 2^{x_0} \bmod 1|, |2^{52+e} \cdot \ln 2 \cdot 2^{x_0} \bmod 1| &\leqslant 2^{-25} \end{aligned}$$

où, dans la première équation, on a $e = 1$ si $|x_0| \leqslant \frac{\pi}{6}$ et $e = 0$ sinon ; et dans la deuxième équation, on a $e = 1$ si $|x_0| \leqslant -\lg \ln 2$ et $e = 0$ sinon. Ceci donne trop de solutions, on garde donc, pour chaque plage où les dix premiers bits sont fixés, l'élément le plus proche du milieu de la plage, ce qui a pour effet de diminuer la valeur maximale prise par $|h|$. Dans la section 2.4, nous décrivons le calcul de ces tables. Quel que soit x, on a $|h| \leqslant 2^{-10.26}$ pour la table de $\sin x$, et $|h| \leqslant 2^{-10.17}$ pour 2^x. Par choix des x_0, les erreurs faites sur les termes $\sin x_0, \cos x_0, 2^{x_0}$, et $\ln 2 \cdot 2^{x_0}$ diminuent de $\approx 2^{-63}$ (pour la méthode classique) à $\approx 2^{-78}$:

$$\begin{aligned} |\sin x_0 - \diamond(\sin x_0)| \leqslant 2^{-79-e} \quad , \quad |\cos x_0 - \diamond(\cos x_0)| \leqslant 2^{-79}, \\ |2^{x_0} - \diamond(2^{x_0})| \leqslant 2^{-78} \quad , \quad |\ln 2 \cdot 2^{x_0} - \diamond(\ln 2 \cdot 2^{x_0})| \leqslant 2^{-78-e}. \end{aligned}$$

Nous expliquons maintenant comment faire en sorte que les évaluations des polynômes approchant $\sin h, \cos h$ et 2^h soient aussi précises, et donc comment avoir des approximations de $\sin x$ et 2^x de qualités équivalentes.

Le cas de $\sin x$**.** Soient $S(h) = -a_3 + a_5 h^2$ et $C(h) = -1/2 + a_4 h^2$, où a_i est le nombre flottant en double précision le plus proche de $\frac{1}{i!}$ pour $i \in [|3, 5|]$. On calcule une approximation de $\sin x$ de la façon suivante :

$$\begin{aligned} P &:= h^2 \cdot (C(h) \cdot \sin x_0 + h \cdot (S(h) \cdot \cos x_0)), \\ Q &:= \sin x_0 + h \cdot \cos x_0, \\ R &:= P + Q, \end{aligned}$$

où le calcul de Q est fait avec des opérations fma (*fused multiply and add*) qui simulent des double-doubles, avec des double-doubles directement, ou en précision étendue, et toutes les autres opérations sont effectuées avec des doubles. Pour l'analyse d'erreur, nous supposerons que le calcul de Q est effectué avec des opérations fma.

En utilisant le fait que $|h| \leqslant 2^{-10.26}$, et la fonction **maximize**[35] de Maple [120], on obtient :

$$|\sin h - h - h^3 S(h)| \leqslant 2^{-83.97}, \ |\cos h - 1 - h^2 C(h)| \leqslant 2^{-71.05}.$$

Nous donnons maintenant l'erreur[36] effectuée lors du calcul de P.

Opération	Quantité approchée	Erreur	Borne
$k{:=}\diamond(h^2)$	h^2	$\leqslant 2^{-(20+53+1)} = 2^{-74}$	$2^{-20.51}$
$S{:=}\diamond(a_5 \cdot k)$	$a_5 h^2$	$a_5 \cdot 2^{-74} + 2^{-(27+53+1)} \leqslant 2^{-79.95}$	$2^{-27.42}$
$S{:=}\diamond(-a_3 + S)$	$S(h)$	$2^{-79.95} + 2^{-56} \leqslant 2^{-55.99}$	$2^{-2.58}$
$S{:=}\diamond(S \cdot \diamond(\cos x_0))$	$S(h)\cos x_0$	$2^{-0.88-55.99} + 2^{-2.58-79} + 2^{-57} \leqslant 2^{-55.93}$	$2^{-3.46}$
$S{:=}\diamond(S \cdot h)$	$hS(h)\cos x_0$	$2^{-55.93-10.26} + 2^{-67} \leqslant 2^{-65.53}$	$2^{-13.71}$
$C{:=}\diamond(a_4 \cdot k)$	$a_4 h^2$	$a_4 \cdot 2^{-74} + 2^{-81} \leqslant 2^{-78.33}$	$2^{-25.10}$
$C{:=}\diamond(-1/2 + C)$	$C(h)$	$2^{-78.33} + 2^{-55} \leqslant 2^{-54.99}$	$2^{-1.00}$
$C{:=}\diamond(C \cdot \diamond(\sin x_0))$	$C(h)\sin x_0$	$2^{-0.24-54.99} + 2^{-1.00-79} + 2^{-55} \leqslant 2^{-54.11}$	$2^{-1.24}$
$S{:=}\diamond(S + C)$	P/h^2	$2^{-54.11} + 2^{-65.53} + 2^{-55} \leqslant 2^{-53.48}$	$2^{-1.23}$
$\bar{P}{:=}\diamond(S \cdot k)$	P	$2^{-20.51-53.48} + 2^{-1.23-74} + 2^{-75} \leqslant 2^{-73.04}$	$2^{-21.73}$

Pour le calcul de Q nous utilisons deux opérations fma :

$$\begin{aligned} Q_h &:= \diamond(h \cdot \diamond(\cos x_0) + \diamond(\sin x_0)) \\ Q_l &:= \diamond(Q_h - \diamond(\sin x_0)) \\ Q_l &:= \diamond(Q_l - h \cdot \diamond(\cos x_0)) \end{aligned}$$

Puisque $|h \cdot \diamond(\cos x_0)| \leqslant \diamond(\sin x_0)/2$, on a $Q_h \in [\diamond(\sin x_0)/2, 2 \diamond (\sin x_0)]$, et donc la deuxième opération est exacte, d'après le lemme de Sterbenz [172]. Par conséquent, on a l'égalité $Q_h + Q_l = h \cdot \diamond(\cos x_0) + \diamond(\sin x_0)$, et :

$$|(Q_h + Q_l) - (h \cdot \cos x_0 + \sin x_0)| \leqslant 2^{-10.26} \cdot 2^{-79} + 2^{-79} \leqslant 2^{-78.99}.$$

Le calcul de R est effectué de la manière suivante :

$$\begin{aligned} R_1 &:= \diamond(\bar{P} + Q_l) \\ R_2 &:= \diamond(R_1 + Q_h), \end{aligned}$$

[35] Cette procédure détermine le maximum global d'une fonction sur un intervalle donné.

[36] Dans le tableau, les calculs d'erreur sont naïfs. Il s'agit d'erreurs absolues entre des flottants et les valeurs exactes qu'ils approchent. L'erreur sur $\diamond(a+b)$, notée $\mathrm{err}(\diamond(a+b))$ est inférieure à la somme de l'erreur sur a, de celle sur b et de l'erreur commise lors de l'arrondi de la somme des valeurs calculées a et b, c'est-à-dire $\mathrm{err}(a)+\mathrm{err}(b)+2^{-54}(\mathrm{bnd}(a)+\mathrm{bnd}(b))$, où $\mathrm{bnd}(x)$ est une borne sur la quantité calculée x. De même, on utilise la majoration $\mathrm{err}(\diamond(a \cdot b)) \leqslant \mathrm{err}(a)\mathrm{bnd}(b) + \mathrm{err}(b)\mathrm{bnd}(a) + 2^{-54} \cdot \mathrm{bnd}(a) \cdot \mathrm{bnd}(b)$.

où R_2 est le résultat final renvoyé pour $\sin x$. En utilisant les résultats précédents, on obtient :

$$\begin{aligned} |R_1 - (\sin x - Q_h)| &\leqslant |R_1 - (P + Q_l)| + |P - (\sin x - h \cdot \cos x_0 - \sin x_0)| \\ &\leqslant 2^{-73.04} + 2^{-75} + 2^{-71.05} \cdot 2^{-0.24} + 2^{-83.97} \cdot 2^{-0.18} \\ &\leqslant 2^{-70.83}. \end{aligned}$$

Par conséquent, on obtient le bon résultat pour $\diamond(\sin x)$ sauf si la distance de $\sin x$ au milieu de deux nombres flottants en double précision le plus proche est inférieure à $2^{-70.83}$. Dans l'hypothèse du modèle aléatoire, cela arrive avec probabilité de l'ordre de $2^{-16.83}$. Pour un arrondi dirigé au lieu de l'arrondi au plus proche, on a le même résultat.

Le cas de 2^x. Soit $P(h) = a_2 + a_3 h + a_4 h^2 + a_5 h^3$, où a_i est le nombre flottant en double précision le plus proche de $\frac{(\ln 2)^i}{i!}$ pour $i \in [|2, 5|]$. On calcule une approximation de 2^x de la façon suivante :

$$\begin{aligned} P &:= 2^{x_0} \cdot h^2 \cdot P(h), \\ Q &:= 2^{x_0} + h \cdot (\ln 2 \cdot 2^{x_0}), \\ R &:= P + Q, \end{aligned}$$

où le calcul de Q est fait avec des opérations fma, avec des double-doubles directement, ou en précision étendue, et toutes les autres opérations sont effectuées avec des doubles. Pour l'analyse d'erreur, nous supposerons que le calcul de Q est effectué avec des opérations fma.

En utilisant le fait que $|h| \leqslant 2^{-10.17}$, et la fonction **maximize** de Maple [120], on obtient :

$$|2^h - 1 - \ln 2 \cdot h - h^2 P(h)| \leqslant 2^{-73.85}.$$

Nous donnons maintenant l'erreur effectuée lors du calcul de P.

Opération	Quantité approchée	Erreur	Borne
$\bar{P}{:=}\diamond(h \cdot a_5)$	$a_5 h$	$\leqslant 2^{-73}$	$2^{-19.72}$
$\bar{P}{:=}\diamond(a_4 + \bar{P})$	$a_4 + a_5 h^2$	$2^{-73} + 2^{-60} \leqslant 2^{-59.99}$	$2^{-6.69}$
$\bar{P}{:=}\diamond(h \cdot \bar{P})$	$a_4 h + a_5 h^2$	$2^{-10.17-59.99} + 2^{-70} \leqslant 2^{-69.07}$	$2^{-16.86}$
$\bar{P}{:=}\diamond(a_3 + \bar{P})$	$a_3 + a_4 h + a_5 h^2$	$2^{-69.07} + 2^{-58} \leqslant 2^{-57.99}$	$2^{-4.17}$
$\bar{P}{:=}\diamond(h \cdot \bar{P})$	$a_3 h + a_4 h^2 + a_5 h^3$	$2^{-10.17-57.99} + 2^{-68} \leqslant 2^{-67.07}$	$2^{-14.34}$
$\bar{P}{:=}\diamond(a_2 + \bar{P})$	$P(h)$	$2^{-67.07} + 2{-}56 \leqslant 2^{-55.99}$	$2^{-2.05}$
$\bar{P}{:=}\diamond(h \cdot \bar{P})$	$hP(h)$	$2^{-10.17-55.99} + 2^{-66} \leqslant 2^{-65.07}$	$2^{-12.22}$
$\bar{P}{:=}\diamond(h \cdot \bar{P})$	$h^2 P(h)$	$2^{-10.17-65.07} + 2^{-76} \leqslant 2^{-74.57}$	$2^{-22.39}$
$\bar{P}{:=}\diamond(\diamond(2^{x_0}) \cdot \bar{P})$	P	$2^{-78-22.39} + 2^{-74.57+1} + 2^{-76} \leqslant 2^{-73.32}$	$2^{-21.38}$

Pour le calcul de Q nous utilisons deux opérations fma :

$$\begin{aligned} Q_h &:= \diamond(h \cdot \diamond(\ln 2 \cdot 2^{x_0}) + \diamond(2^{x_0})) \\ Q_l &:= \diamond(Q_h - \diamond(2^{x_0})) \\ Q_l &:= \diamond(Q_l - h \cdot \diamond(\ln 2 \cdot 2^{x_0})) \end{aligned}$$

où $\diamond(\ln 2 \cdot 2^{x_0})$ est dans la table précalculée, ou est recalculé à partir de $\diamond(2^{x_0})$, comme expliqué dans l'annexe du chapitre. Puisque $|h \cdot \diamond(\ln 2 \cdot 2^{x_0})| \leqslant \diamond(2^{x_0})/2$, on a $Q_h \in [\diamond(2^{x_0})/2, 2 \diamond (2^{x_0})]$, et donc la deuxième opération est exacte, d'après le lemme de Sterbenz [172]. Par conséquent, on a l'égalité $Q_h + Q_l = h \cdot \diamond(\ln 2 \cdot 2^{x_0}) + \diamond(2^{x_0})$, et :

$$|(Q_h + Q_l) - (h \cdot \ln 2 \cdot 2^{x_0} + 2^{x_0})| \leqslant 2^{-10.17} \cdot 2^{-78} + 2^{-78} \leqslant 2^{-77.99}.$$

Le calcul de R est effectué de la manière suivante :

$$\begin{aligned} R_1 &:= \diamond(\bar{P} + Q_l) \\ R_2 &:= \diamond(R_1 + Q_h), \end{aligned}$$

où R_2 est le résultat final renvoyé pour 2^x. En utilisant les résultats précédents, on obtient :

$$\begin{aligned} |R_1 - (2^x - Q_h)| &\leqslant |R_1 - (P + Q_l)| + |P - (2^x - h \cdot \ln 2 \cdot 2^{x_0} - 2^{x_0})| \\ &\leqslant 2^{-73.32} + 2^{-75} + 2^{-73.85} \cdot 2^{1.00} \\ &\leqslant 2^{-71.88}. \end{aligned}$$

Par conséquent, on obtient le bon résultat pour $\diamond(2^x)$ sauf si la distance de 2^x au milieu de deux nombres flottants en double précision le plus proche est inférieure à $2^{-71.88}$. Dans l'hypothèse du modèle aléatoire, cela arrive avec probabilité de l'ordre de $2^{-17.88}$. Pour un arrondi dirigé au lieu de l'arrondi au plus proche, on a le même résultat.

2.3 Le calcul des tables.

Jusqu'à maintenant, nous avons décrit des schémas d'évaluation qui reposent sur des tables précalculées, sans avoir expliqué comment ces tables ont été calculées. Ce précalcul n'est pas un problème trivial ; notre exigence plus forte sur la qualité de la table rend la recherche exhaustive de Gal hors de portée : il faudrait 2^{52} appels de fonction en précision étendue pour la première table pour 2^x, et 2^{62} appels pour les deux autres tables. Ceci est hors de portée de nos moyens actuels. Pour la première table pour 2^x, il s'agit d'une recherche de mauvais cas, comme celles décrites au chapitre III-1, et on peut donc utiliser l'algorithme de Lefèvre [107] ou l'algorithme qui repose sur la méthode de Coppersmith. On utiliserait plus vraisemblablement l'algorithme de Lefèvre qui reste le plus efficace en double précision. Dans cette section, nous décrivons et analysons une modification de l'algorithme proposé au chapitre III-1, qui permet de trouver les mauvais cas simultanés de deux fonctions. On pourrait utiliser l'algorithme de Lefèvre sur la première fonction et tester les valeurs trouvées sur la seconde pour voir s'il s'agit bien d'un mauvais cas simultané. Cette approche coûte asymptotiquement $2^{2n/3+o(n)}$ si n est la précision d'entrée-sortie. L'algorithme que nous décrivons ci-dessous a une complexité asymptotique de l'ordre de $2^{n/2+o(n)}$ quand les précisions d'entrée et de sortie sont les mêmes, ce qui est meilleur. Comme le montrent les résultats de la section 2.4, il en est de même en pratique.

2.3.1 Recherche des mauvais cas simultanés de deux fonctions.

On se donne deux fonctions f_1 et f_2 définies sur $[1/2, 1[$, et trois entiers N, N' et M. On souhaite trouver tous les entiers $x \in [\,|\,\lceil N/2\rceil\,, N-1\,|]$ tels que :

$$\forall i \in \{1,2\}, \left| N' \cdot f_i\left(\frac{x}{N}\right) \mod 1 \right| \leqslant \frac{1}{M}. \tag{2.1}$$

Cette équation est une généralisation immédiate du problème d'inversion d'une fonction quand les premiers bits sont inconnus (définition 29, page 157), à la résolution duquel nous avons consacré le chapitre III-1. Si l'intervalle d'arrivée des fonctions est $[1/2, 1[$, alors le lien avec la recherche de mauvais cas simultanés est le suivant : on prend $N = 2^n$ où n est la précision d'entrée ($n = 63$ dans notre cas), $N' = 2^{n'}$ où n' est la précision de sortie ($n' = 53$ dans notre cas), et $M = 2^m$ si l'on cherche des m-mauvais cas simultanés. Si l'intervalle d'arrivée des fonctions n'est pas $[1/2, 1[$, alors il faut changer n' en $n' + j_i$ où j_i est choisi en fonction de l'intervalle d'arrivée de f_i. Éventuellement, on peut résoudre plusieurs équations du même type que l'équation (2.1) si les intervalles d'arrivée des deux fonctions chevauchent plusieurs plages d'exposants.

La méthode décrite ici est très similaire à celle du chapitre III-1. L'intervalle $[1/2, 1[$ est subdivisé en plusieurs sous-intervalles. Pour chacun de ces sous-intervalles, on approche les fonctions étudiées par des polynômes. On résout alors l'équation (2.1) pour ces polynômes, à l'aide de la méthode de Coppersmith pour trouver les petites racines de polynômes modulaires multivariés (voir la section 1.1 du chapitre III-1). On utilise en fait un cas dégénéré de la méthode car on se restreint au choix de paramètre $\alpha = 1$ (α est la puissance du module utilisée dans la méthode)[37].

La recherche sur l'intervalle de départ $[1/2, 1[$ est divisée en $N/2T$ recherches sur des intervalles de longueur T/N. Chacune de ces petites recherches est effectuée extrêmement rapidement — asymptotiquement, en temps quadratique en $\log N, \log N'$ et $\log M$ — et on cherche donc naturellement à augmenter T car la complexité de la recherche complète sera de l'ordre de N/T. La raison pour laquelle on ne peut augmenter T arbitrairement est que la recherche rapide sur les petits intervalles est possible seulement si les intervalles en question sont suffisamment petits. Le but est donc de maximiser T sous la condition que la recherche sur les petits intervalles soit réalisable efficacement.

Pour un petit intervalle donné, l'algorithme commence par approcher les fonctions f_1 et f_2 par des polynômes P_1 et P_2 de degré 2 sur l'intervalle. On peut par exemple prendre les développements de Taylor d'ordre 2 des fonctions au milieu de l'intervalle. Ce n'est vraisemblablement pas la meilleure approximation, mais cela est suffisant ici. Une fois les deux polynômes calculés, on essaie de résoudre l'équation (2.1) pour les polynômes P_i à la place des fonctions f_i. Bien entendu, ces polynômes doivent être suffisamment proches des fonctions et M doit être légèrement diminué pour ne pas écarter de solution. Plus précisément, on veut que $N' \cdot |f_i(x) - P_i(x)| \leqslant 1/M$ pour tout x dans l'intervalle considéré. Si f est $\mathcal{C}^3$, on sait qu'il existe un réel t dans l'intervalle tel que $|f_i(x) - P_i(x)| = f^{(3)}(t)\frac{T^3}{N^3}$.

[37] Cette valeur de α est celle qui donne les meilleures bornes dans notre situation : augmenter α ne semble pas améliorer les bornes.

Si l'on suppose que la dérivée troisième de f est uniformément bornée par une constante, on obtient alors la condition nécessaire :

$$\frac{N'T^3}{N^3} \leqslant \frac{1}{M},$$

que l'on appelle *condition de Taylor*. On est alors ramené à trouver les solutions de l'équation : $|N' \cdot P_i\left(\frac{x}{N}\right) \bmod 1| \leqslant \frac{2}{M}$, pour $x \in [|-T, T|]$ et $i \in \{1, 2\}$.

Il reste ensuite à trouver les solutions entières de ces deux dernières équations simultanées. Cela est fait à l'aide de la méthode de Coppersmith pour trouver des petites racines de polynômes multivariés modulaires, décrite de manière plus générale au chapitre III-1 (section 1.1). En effet, il s'agit de résoudre le système d'équations polynomiales à trois inconnues suivant :

$$\forall i \in \{1, 2\}, Q_i(x, y_i) \bmod 1 = 0,$$

où $Q_i(x, y_i) = N' \cdot P_i(x/N) + y_i$, et les racines x, y_1 et y_2 sont petites, c'est-à-dire $|x| \leqslant X, |y_1| \leqslant Y$ et $|y_2| \leqslant Y$ avec $X = T$ et $Y = 2/M$. Comme nous l'avons vu, la méthode de Coppersmith modulaire multivariée est seulement heuristique[38], ce qui rend notre algorithme lui aussi heuristique. Quand la méthode de Coppersmith échoue, nous divisons T par 2 et remplaçons l'intervalle considéré par deux sous-intervalles. Quand T devient trop petit, alors on effectue une recherche exhaustive. La conséquence de notre stratégie est que l'on trouvera toutes les solutions aux équations de départ, et que l'aspect heuristique est déplacé vers l'analyse de complexité.

L'algorithme est décrit à la figure 2.3, qui est l'équivalent des étapes 8 à 10 de l'algorithme de la figure 1.2 du chapitre III-1, page 166 (le traitement des cas d'échecs effectué alors peut être réutilisé ici sans modification). À l'étape 7 de l'algorithme, la fonction **Réduire** peut consister en l'algorithme LLL, par exemple sa variante décrite au chapitre II-3, ou en l'algorithme glouton de réduction en petite dimension décrit au chapitre II-2. Remarquons aussi que dans certaines étapes de l'algorithme décrit ci-dessous, on travaille avec des nombres réels en précision infinie. En particulier, les coefficients des polynômes construits à l'étape 1 sont des réels. Il est en fait suffisant de les calculer avec une précision de $O(\log N' + \log M)$ bits pour que cela n'influe pas sur l'étape 2. À l'étape 4, on arrondit les coefficients des polynômes $\tilde{P}_i$: cela permet de manipuler des entrées entières lors de la réduction de réseau[39]. À cause de ces erreurs ajoutées au calcul et à cause de l'erreur

[38] Le problème des résultants nuls ne se pose pas ici car les trois petits polynômes obtenus après réduction sont chacun de la forme $A(x) + \alpha y_1 + \beta y_2$: il suffit donc d'éliminer les variables y_1 et y_2 avec de l'algèbre linéaire, car ils sont linéairement indépendants (les vecteurs associés font partie de la base d'un réseau). Par contre, puisque la dimension du réseau ne tend pas vers $+\infty$ (α est constant), on ne peut se prémunir de la même manière qu'au chapitre III-1 contre l'éventualité qu'il n'y ait qu'un seul vecteur court. On suppose donc ici que les vecteurs obtenus après réduction ont des longueurs similaires, ce qui est une hypothèse heuristique mais que nous avons vérifiée en pratique sur les tests que nous avons effectués.

[39] C'est pour cette raison que l'algorithme diffère un peu de l'algorithme de la figure 1.2 du chapitre III-1, page 166. En particulier, plutôt que d'utiliser la méthode de Coppersmith avec des nombres réels modulo 1, on l'utilise avec des entiers modulo C. Cela rend nécessaire l'introduction des polynômes $\tilde{P}_i$, mais rapproche l'algorithme de son implantation.

des approximations polynomiales de l'étape 1, on peut trouver des solutions parasites, c'est-à-dire qui ne sont pas des solutions de l'équation (2.1). C'est pourquoi on effectue une vérification à l'étape 12.

Par ailleurs, à l'étape 9, plutôt que de prendre les trois vecteurs les plus courts, on peut prendre n'importe quel triplet de vecteurs courts. À l'étape 1, l'indépendance linéaire des vecteurs implique leur indépendance algébrique (ce qui n'est pas le cas en général pour la méthode de Coppersmith), car les polynômes sont de degré 1 en ϕ et ν. Par la forme du réseau, on sait aussi que Q sera de degré au plus 1, ce qui rend la recherche des racines de l'étape 11 particulièrement simple.

Algorithme: **Recherche de mauvais cas simultanés.**
Entrée : Deux fonctions f_1, f_2 et quatre entiers N, N', M, T.
Sortie : Tous les $x \in [|-T, T|]$ qui satisfont l'équation (2.1).

1. Soient P_1, P_2 les développements de Taylor de f_1, f_2 en 0.
2. Calculer ε tel que $N' \cdot |P_i(t) - f_i(t)| < \varepsilon$ pour tout $t \in \left[-\frac{T}{N}, \frac{T}{N}\right]$ et tout $i \in \{1, 2\}$.
3. $M' := \left\lfloor \frac{1/2}{1/M + \varepsilon} \right\rfloor$, $C := 3M'$, et $\tilde{P}_i(\tau) = CN' \cdot P_i(T\tau/N)$ pour $i \in \{1, 2\}$.
4. $e_1 := 1, e_2 := \tau, e_3 := \tau^2, e_4 := \nu, e_5 := \phi$.
5. Arrondir en entiers les coefficients des $\tilde{P}_i$.
6. $g_1 := C, g_2 := C \cdot \tau, g_3 := \tilde{P}_1(\tau) + 3\nu, g_4 := \tilde{P}_2(\tau) + 3\phi$.
7. Créer la matrice L avec 4 lignes et 5 colonnes pour laquelle $L_{k,l}$ est le coefficient du monôme e_l dans le polynôme g_k.
8. $V :=$ **Réduire**(L).
9. Soient $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$ trois plus courts vecteurs de V. Soient $Q_1(\tau), Q_2(\tau), Q_3(\tau)$ les trois polynômes correspondants.
10. S'il existe $i \in \{1, 2, 3\}$ tel que $\|\mathbf{v}_i\|_1 \geqslant C$, renvoyer ÉCHEC.
11. Calculer $Q(\tau)$ une combinaison linéaire des Q_i qui ne dépend ni de ν ni de ϕ.
12. Soit $q(x) = Q(\frac{x}{T})$. Pour tout $x_0 \in [|-T, T|]$ racine de q, tester si x_0 satisfait l'équation (2.1). Si c'est le cas, renvoyer x_0.

FIG. 2.3 – L'algorithme de recherche de mauvais cas simultanés.

2.3.2 Correction de l'algorithme.

Le théorème suivant donne la correction de l'algorithme décrit à la figure 2.3. La preuve est une généralisation directe de celle du théorème 24, page 168.

Théorème 30. *Dans le cas où l'algorithme de la figure 2.3 ne renvoie pas* ÉCHEC*, il se comporte correctement, c'est-à-dire il renvoie tous les $x_0 \in [|-T, T|]$ qui satisfont l'équation (2.1).*

Démonstration. Grâce à l'étape 12, il suffit de montrer que l'on n'écarte pas de solution à l'équation (2.1). Supposons que l'on ait un $x_0 \in [|-T, T|]$ qui satisfasse l'équation (2.1). À

partir de la définition des P_i à l'étape 1 et la définition de ε à l'étape 2, on a pour $i \in \{1,2\}$,

$$\left| N' \cdot P_i\left(\frac{x_0}{N}\right) \mod 1 \right| \leqslant \frac{1}{M} + \varepsilon \leqslant \frac{1}{2M'}.$$

On en déduit que $|\tilde{P}_i(x_0/T) \bmod C| \leqslant 3/2$ pour $i \in \{1,2\}$. Si l'on arrondit au plus proche chaque coefficient de $\tilde{P}_i$, on obtient $|\tilde{P}_i(x_0/T) \bmod C| \leqslant 3$ car il s'agit d'un polynôme de degré 2 et il a donc au plus trois coefficients, et $|x_0| \leqslant T$.

On en déduit que les polynômes g_1, g_2, g_3 et g_4 définis à l'étape 6 ont une racine commune $(x_0/T, \nu_0, \phi_0) \in [-1,1]^3$ modulo C. Comme les Q_i sont des combinaisons linéaires des g_i, le triplet $(x_0/T, \nu_0, \phi_0)$ est aussi une racine commune aux Q_i modulo C. Si l'on passe le test de l'étape 10, les coefficients des Q_i sont assez petits pour que $Q_i(x_0/T, \nu_0, \phi_0) < C$ et donc le triplet $(x_0/T, \nu_0, \phi_0)$ est une racine commune aux Q_i sur les réels (sans le modulo). Par conséquent, le réel x_0/T est racine de Q et donc x_0 est une racine entière de q et sera trouvé à l'étape 12. □

2.3.3 Analyse de complexité de l'algorithme.

Maintenant que nous sommes assurés que l'algorithme est correct, nous estimons sa complexité. La borne de complexité est seulement heuristique, à cause de la possibilité d'échec de l'algorithme à l'étape 10. Soient $a_0^{(i)}, a_1^{(i)}, \ldots$ les coefficients des approximations de Taylor des f_i pour $i \in \{1,2\}$. Nous supposons ici que pour tout $k \leqslant 3$ et tout $i \in \{1,2\}$, on ait $|a_k^{(i)}| = O(1)$.

Dans l'algorithme de recherche des mauvais cas simultanés, les paramètres d'entrée sont N et N' (pour notre situation, on a $N = 2^{63}$ et $N' = 2^{53}$). À ceux-ci se rajoutent les paramètres T et M (qui sont respectivement la taille des sous-intervalles et la qualité des mauvais cas simultanés cherchés), qui sont fixés à la fois pour rendre l'algorithme efficace et pour obtenir suffisamment de mauvais cas simultanés. Comme nous l'avons vu, nous avons déjà une borne liée à l'approximation des fonctions de départ par des polynômes de degré 2, il s'agit de la borne de Taylor :

$$\frac{N'T^3}{N^3} \leqslant \frac{1}{M}.$$

Taille des coefficients des polynômes $\tilde{P}_i$. Les polynômes $\tilde{P}_i(\tau) = \tilde{p}_0^{(i)} + \tilde{p}_1^{(i)}\tau + \tilde{p}_2^{(i)}\tau^2$ calculés à l'étape 3 de l'algorithme de la figure 2.3 satisfont $\tilde{p}_2^{(i)} = O(MN'T^2N^{-2})$. En effet, on a $\tilde{P}_i(\tau) = CN' \cdot P_i(T\tau/N)$ pour $i \in \{1,2\}$, et les coefficients des P_i sont $O(1)$. Rappelons aussi pour compléter l'argument que $C = O(M)$.

La matrice calculée à l'étape 7. À l'étape 8, il faut réduire le réseau engendré par les lignes de la matrice suivante :

$$L = \begin{bmatrix} C & & & & \\ & CT & & & \\ \tilde{p}_0^{(1)} & \tilde{p}_1^{(1)} & \tilde{p}_2^{(1)} & 3 & \\ \tilde{p}_0^{(2)} & \tilde{p}_1^{(2)} & \tilde{p}_2^{(2)} & & 3 \end{bmatrix}.$$

Le déterminant de cette matrice est facile à calculer :

$$|\det L| = 3C^2T\sqrt{\left(\tilde{p}_2^{(1)}\right)^2 + \left(\tilde{p}_2^{(2)}\right)^2 + 9} = O(M^3T^3N'N^{-2}).$$

Borne de Coppersmith. Pour que l'algorithme de la figure 2.3 ne renvoie pas ÉCHEC à l'étape 10, il faut au moins que l'un des vecteurs renvoyés soit de norme ℓ_1 plus petite que C. Cela ne suffit pas, mais en général, les vecteurs renvoyés par LLL sont de longueurs similaires, donc si le premier est suffisamment court, les autres devraient l'être aussi. Cette heuristique semble tout à fait raisonnable en pratique. Le théorème de Minkowski (voir le chapitre I-1) — dont une version constructive et efficace est donnée au chapitre II-3, et aussi au chapitre II-2 car on est en dimension 4 — nous donne ainsi une condition que l'on appelle condition de Minkowski : $|\det L|^{1/4} < C$, ce qui se récrit de la façon suivante, à une constante multiplicative près :

$$T^3N'N^{-2} < M.$$

Conclusion. Les bornes de Coppersmith et de Taylor indiquent que le M optimal est de l'ordre de $N^{1/2}$ (on ne peut pas prendre M plus grand, sinon nous n'aurions pas assez de solutions, et prendre M plus petit ralentit l'algorithme) et T de l'ordre de $(N^{5/2}/N')^{1/3}$. La complexité de l'algorithme est donc de l'ordre de :

$$\mathcal{P}\mathrm{ol}(\log N) \cdot \frac{N}{T} \approx \mathcal{P}\mathrm{ol}(\log N) \cdot N^{1/6}N'^{1/3}.$$

Il est amusant de voir que la précision d'arrivée, c'est-à-dire N', contribue deux fois plus au coût que la précision de départ, c'est-à-dire N. Quand $N = N'$, cette complexité est de l'ordre de $N^{1/2}$.

Remarque. La technique développée ici ressemble beaucoup à la méthode présentée au chapitre III-1 pour trouver des mauvais cas pour l'arrondi d'une fonction élémentaire. Cependant, les choses sont un peu plus simples qu'alors. Par exemple, augmenter le degré des approximations ne semble rien apporter de plus et considérer des puissances et des produits des P_i ne semble pas améliorer l'algorithme non plus. Par ailleurs, à la place d'un calcul de résultants, on effectue l'élimination de variables finale à l'aide d'un pivot de Gauss.

2.4 Résultats expérimentaux.

Nous décrivons ici les trois tables mentionnées dans ce chapitre, celles-ci étant disponibles à l'URL `http://www.loria.fr/~stehle/these.html`.

2.4.1 La première table pour 2^x.

Nous donnons à l'URL ci-dessus les solutions $x_0 \in [|2^{52}, 2^{53}[|$ de l'équation :

$$\left|2^{52+\frac{x_0}{2^{53}}} \bmod 1\right| \leqslant 2^{-m},$$

avec $m \geqslant 41$. La table nous a été donnée par Lefèvre, et a été calculée à l'aide de son algorithme pour trouver les mauvais cas d'une fonction.

Il y a 4001 éléments dans la table complète, 1985 si on se restreint à $m \geqslant 42$, 973 avec $m \geqslant 43$, 491 avec $m \geqslant 44$ et 265 si on choisit $m \geqslant 45$. La distance maximale entre deux éléments consécutifs est $\frac{9331088135396}{2^{53}} < 2^{-9.914}$, et le pire cas est $x_0 =$ `13e34fa6ab969e` avec $m = 52$.

2.4.2 La table pour $\sin x$.

Nous donnons à l'URL ci-dessus 512 solutions[40] simultanées $x_0 \in [|2^{62}, 2^{63}[|$ des deux équations :

$$|2^{53+e} \cdot \sin \frac{x_0}{2^{63}} \bmod 1|, |2^{53} \cdot \cos \frac{x_0}{2^{63}} \bmod 1| \leqslant 2^{-25},$$

avec $e = 1$ si $\left|\frac{x_0}{2^{63}}\right| \leqslant \frac{\pi}{6}$ et $e = 0$ sinon. Ces solutions sont choisies de la manière suivante : toutes les solutions de l'équation ont été calculées, et on a gardé les solutions les plus proches des $\frac{2^{10}+2j+1}{2^{11}}$ pour $j \in [|0, 2^9[|$. La distance maximale entre une solution et le $\frac{2^{10}+2j+1}{2^{11}}$ le plus proche est inférieure à $2^{-11.58}$ et on aura donc toujours $|h| \leqslant 2^{-11}+2^{-11.58} \leqslant 2^{-10.26}$. L'ensemble total des solutions des deux équations contient environ 2^{14} éléments. Remarquons que ces valeurs sont cohérentes avec le modèle aléatoire adopté.

Le calcul a été effectué avec l'algorithme de la figure 2.3, sur un Opteron 2.4 GHz. Il a pris environ une journée. Pour comparer, l'algorithme de Lefèvre effectué sur la première fonction aurait pris au moins une centaine de jours (en ne comptant que les appels aux fonctions sin et cos en précision de 63 bits en entrée et 110 bits en sortie, avec MPFR [147]). La recherche exhaustive aurait nécessité 2^{62} appels des fonctions sin et cos. Ici, le calcul a fait intervenir un peu moins de 2 milliards de réductions de réseaux en dimension 4, celles-ci étant effectuées à l'aide de l'algorithme décrit au chapitre II-3, plus précisément à l'aide de l'implantation de cet algorithme décrite au chapitre II-4 (*heuristic.c* avec comme arithmétique flottante la double précision). Le code utilisé pour ce calcul est disponible lui aussi à l'URL ci-dessus.

2.4.3 La table pour 2^x et $\ln 2 \cdot 2^x$.

Nous donnons aussi à l'URL ci-dessus 512 solutions simultanées $x_0 \in [|2^{62}, 2^{63}[|$ des deux équations :

$$|2^{52} \cdot 2^{\frac{x_0}{2^{63}}} \bmod 1|, |2^{52+e} \cdot \ln 2 \cdot 2^{\frac{x_0}{2^{63}}} \bmod 1| \leqslant 2^{-25},$$

où $e = 1$ si $\left|\frac{x_0}{2^{63}}\right| \leqslant -\lg \ln 2$ et $e = 0$ sinon. Ces solutions sont choisies de la manière suivante : toutes les solutions de l'équation ont été calculées, et on a gardé les solutions les plus proches des $\frac{2^{10}+2j+1}{2^{11}}$ pour $j \in [|0, 2^9[|$. La distance maximale entre une solution et son $\frac{2^{10}+2j+1}{2^{11}}$ correspondant est inférieure à $2^{-11.37}$ et on aura donc toujours $|h| \leqslant 2^{-11} + 2^{-11.37} \leqslant 2^{-10.17}$. L'ensemble complet des solutions des deux équations contient environ 2^{14} éléments. Remarquons que ces valeurs sont cohérentes avec le modèle aléatoire adopté. Les calculs ont été effectués de la même manière que pour la fonction $\sin x$.

[40] Pour obtenir une table de quelques kilo-octets.

2.5 Généralisation possible.

Une question naturelle serait de s'intéresser aux mauvais cas simultanés de trois fonctions. En effet, considérons la formule :

$$f(x_0+h) = f(x_0) + f'(x_0)\cdot h + \frac{f''(x_0)}{2}\cdot h^2 + O(h^3).$$

Prenons une table de la même taille (2^9 entrées), avec un $|h|$ maximal du même ordre de grandeur, par exemple $|h| \leqslant 2^{-10}$. Si l'on veut plus de 80 bits de précision simulée (avec comme précision réelle la double précision) sur le résultat de $f(x_0+h)$, on peut procéder comme suit :

1. Évaluer le terme $O(h^3)$ en double précision (on doit un peu augmenter le degré de l'approximation polynomiale pour baisser l'erreur d'approximation elle aussi),
2. Exiger que x_0 soit un 30-mauvais cas de f,
3. Exiger que x_0 soit un 20-mauvais cas de f' (car $|h| \leqslant 2^{-10}$) et évaluer $f'(x_0)\cdot h$ en précision étendue,
4. Exiger que x_0 soit un 10-mauvais cas de $f''/2$ (car $|h|^2 \leqslant 2^{-10}$) et évaluer $f''(x_0)\cdot h^2$ en précision étendue.

Dans le modèle aléatoire, cela semble possible car $30+20+10 = 60 < 63$. Il convient cependant d'ajuster les constantes pour avoir suffisamment d'éléments dans la table. Un autre point est d'adapter l'algorithme de la figure 2.3 au cas de mauvais cas simultanés déséquilibrés (k_1-mauvais pour la fonction f_1, k_2-mauvais pour la fonction f_2, ...), et voir comment ajouter une troisième fonction (on pourrait éventuellement s'intéresser aux deux premières fonctions, et garder parmi les solutions ainsi obtenues celles qui sont aussi des mauvais cas pour la troisième fonction). Nous aurions finalement trois niveaux de méthode de Gal pour la fonction f qui coûtent de plus en plus cher : la méthode de Gal classique qui échoue avec une probabilité $\approx 2^{-10}$, la méthode de Gal avec deux fonctions associées qui échoue avec une probabilité $\approx 2^{-20}$, et la méthode avec trois fonctions associées qui échoue avec une probabilité $\approx 2^{-30}$. Il serait intéressant de voir comment les faire s'enchaîner efficacement en pratique, en particulier comment réutiliser certains calculs intermédiaires.

Pour la fonction $\sin x$, cette méthode ne nécessite peut-être pas de calcul supplémentaire car $f''(x_0) = -f(x_0)$, mais il faut faire attention au fait que l'on s'intéresse à $f''/2$ et non à f'' (cela décale l'exposant des nombres renvoyés). Pour la fonction 2^x, on s'intéresserait à la fonction $\frac{(\ln 2)^2}{2}\cdot 2^x$. À l'annexe du chapitre, on montre que dans ce cas, il n'y a pas besoin de stocker les approximations des $\frac{(\ln 2)^2}{2}\cdot 2^{x_0}$ car on peut les retrouver rapidement à l'aide des approximations des 2^{x_0}.

Annexe.

Nous nous intéressons ici à la table de la deuxième variante de 2^x, dans laquelle sont stockées des entrées $(x_0, \diamond(2^{x_0}), \diamond(\ln 2\cdot 2^{x_0}))$, où $\diamond(2^{x_0})$ et $\diamond(\ln 2\cdot 2^{x_0})$ sont extrêmement

proches des valeurs correspondantes. Nous montrons comment ne pas stocker la troisième colonne, c'est-à-dire les valeurs $\diamond(\ln 2 \cdot 2^{x_0})$. Nous faisons de même pour le cas où l'on voudrait aussi une quatrième colonne correspondant à $\diamond\left(\frac{(\ln 2)^2}{2} \cdot 2^{x_0}\right)$.

Théorème 31. *On considère la double précision. Soient* $a = \diamond(\ln 2)$ *et* $b = \diamond\left(\frac{(\ln 2)^2}{2}\right)$. *Soit* $x_0 \in [1/2, 1[$ *un réel tel que :*

$$\begin{aligned} |2^{x_0} - \diamond(2^{x_0})| &\leqslant 2^{-m}, \\ |\ln 2 \cdot 2^{x_0} - \diamond(\ln 2 \cdot 2^{x_0})| &\leqslant 2^{-m'}, \\ \left|\frac{(\ln 2)^2}{2} \cdot 2^{x_0} - \diamond\left(\frac{(\ln 2)^2}{2} \cdot 2^{x_0}\right)\right| &\leqslant 2^{-m'}, \end{aligned}$$

avec $2^{-m} + 2^{-m'} \leqslant 2^{-57}$. *Alors on a :*

$$\diamond(a \cdot \diamond(2^{x_0})) = \diamond(\ln 2 \cdot 2^{x_0}) \text{ et } \diamond(b \cdot \diamond(2^{x_0})) = \diamond\left(\frac{(\ln 2)^2}{2} \cdot 2^{x_0}\right).$$

Pour la preuve, on se sert du fait que l'on sait à l'avance que le résultat de l'opération $\diamond(a \cdot \diamond(2^{x_0}))$ est extrêmement proche d'un nombre représentable, et du fait que $\diamond(2^{x_0})$ est lui aussi très proche d'un nombre exactement représentable.

Démonstration. Tout d'abord, remarquons que $a \in [1/2, 1[$, $b \in [1/8, 1/4[$ et :

$$|a - \ln 2| \leqslant 2^{-55.25}, \left|b - \frac{(\ln 2)^2}{2}\right| \leqslant 2^{-56.54}.$$

Nous avons donc :

$$\begin{aligned} |a \cdot \diamond(2^{x_0}) - \diamond(\ln 2 \cdot 2^{x_0})| &\leqslant |a - \ln 2| \cdot \diamond(2^{x_0}) + |2^{x_0} - \diamond(2^{x_0})| \cdot \ln 2 + 2^{-m'} \\ &\leqslant 2^{-55.25} \cdot 2 + 2^{-m} + 2^{-m'} \\ &< 2^{-54}. \end{aligned}$$

Par ailleurs, on sait que $a \cdot \diamond(2^{x_0}) \geqslant 1/2$, donc $\diamond(\ln 2 \cdot 2^{x_0})$ est le nombre flottant en double précision le plus proche de $a \cdot \diamond(2^{x_0})$, ce qui donne la première partie du résultat. Pour la deuxième partie, nous procédons de même :

$$\begin{aligned} \left|b \cdot \diamond(2^{x_0}) - \diamond\left(\frac{(\ln 2)^2}{2} \cdot 2^{x_0}\right)\right| &\leqslant \left|b - \frac{(\ln 2)^2}{2}\right| \cdot \diamond(2^{x_0}) + |2^{x_0} - \diamond(2^{x_0})| \cdot \frac{(\ln 2)^2}{2} + 2^{-m'} \\ &\leqslant 2^{-55.54} + 2^{-m} + 2^{-m'} \\ &< 2^{-55}. \end{aligned}$$

Par ailleurs, on sait que $b \cdot \diamond(2^{x_0}) \geqslant 1/4$, donc $\diamond\left(\frac{(\ln 2)^2}{2} \cdot 2^{x_0}\right)$ est le nombre flottant en double précision le plus proche de $b \cdot \diamond(2^{x_0})$. □

Chapitre III-3

Cryptanalyse du schéma de chiffrement de J. E. Littlewood.

Le travail de recherche correspondant à ce chapitre a été initié à l'université de Stanford sous la direction et avec les conseils de Dan Boneh. Il a fait l'objet d'une publication dans le journal Cryptologia [166].

J. E. Littlewood, 1953 — *The legend that every cipher is breakable is of course absurd,*

though still widespread among people who should know better.

Sommaire

3.1 Présentation et cryptanalyse du schéma original de Littlewood. 224

3.2 Une modification du schéma de Littlewood. 227

3.2.1 Randomisation du générateur de Littlewood. 228

3.3 Une attaque reposant sur l'approximation polynomiale. . 229

3.3.1 Approcher f par un polynôme. 229

3.3.2 D'un problème sur les réels à un problème sur les entiers. . 230

3.3.3 Comment retrouver a avec de la réduction de réseau. . . . 231

3.4 Conclusion et question ouverte. 232

L'objet de ce chapitre est d'utiliser la méthode de recherche des pires cas de fonctions mathématiques décrite dans le chapitre III-1 pour casser un schéma de chiffrement proposé par le théoricien des nombres John Edensor Littlewood dans les années 1950. Plus précisément, nous donnons une attaque très simple contre ce schéma qui fonctionne pour une grande plage de paramètres raisonnables ; nous expliquons ensuite la façon naturelle de contourner cette attaque élémentaire ; et enfin nous décrivons comment casser cette procédure réparée en utilisant une approche qui repose sur la méthode de Coppersmith pour trouver les petites racines de polynômes modulaires multivariés, et qui s'inspire de l'algorithme décrit au chapitre III-1 pour recherche des pires cas de fonctions mathématiques.

3.1 Présentation et cryptanalyse du schéma original de Littlewood.

Un des buts principaux de la cryptographie est de chercher des problèmes mathématiques sur lesquels faire reposer des primitives de sécurité. Un certain nombre d'objets plus ou moins élaborés ont été étudiés sous cet angle, avec des succès variés, comme par exemple le problème de la factorisation, les problèmes de logarithme discret dans des corps finis ou sur des courbes algébriques, le problème du plus court vecteur dans un réseau Euclidien, le problème du mot de poids minimal dans un code linéaire, les problèmes de conjugaison dans les groupes de tresses, etc. De façon surprenante, l'analyse, et en particulier les fonctions mathématiques élémentaires, a été peu étudiée sous l'angle cryptologique. Cela est d'autant plus surprenant qu'il est très facile d'observer - à l'aide d'une calculatrice par exemple - l'aléa des chiffres non significatifs obtenus à l'aide des fonctions élémentaires, par exemple sin ou log. Le schéma de Littlewood provient directement de cette observation naïve, faite alors sur des tables de calcul plutôt que sur des calculatrices. Cette idée d'utiliser les fonctions élémentaires pour construire des primitives cryptographiques peut être motivée par le fait qu'il existe des implantations très efficaces de certaines d'entre elles en logiciel et en matériel, et qu'elles sont universellement répandues.

En 1953, dans le livre *A Mathematician's Miscellany* [117], Littlewood propose d'utiliser une table de logarithmes pour faire du chiffrement. Voici ses propos :

> *The legend that every cipher is breakable is of course absurd, though still widespread among people who should know better. I give a sufficient example, without troubling about its precise degree of practicability. Suppose we have a 5-figure number N. Starting at a place N in a 7-figure* log*-table take a succession of pairs of digits $d_1d'_1, d_2d'_2, \ldots$ from the last figures of the entries. Take the remainder of the 2-figure number $d_nd'_n$ after division by 26. This gives a 'shift' s_n, and the code is to shift the successive letters of the message by $s_1, s_2, \ldots$ respectively.*
>
> *It is sufficiently obvious that a single message cannot be unscrambled, and this even if all were known except the key number N (indeed the triply random character of s_n is needlessly elaborate). If the same code is used for a number of messages it could be broken, but all we need do is to vary N. It can be made to be dependent on a date, given it in clear; the key might e.g. be that N is the first 5 figures of the 'tangent' of the date (read as degrees, minutes, seconds : 28° 12' 52" for Dec. 28, 1952). This rule could be carried in the head, with nothing on paper to be stolen or betrayed. If any one thinks there is a possibility of the entire scheme being guessed he could modify 26 to 21 and use a date one week earlier than the one given in clear.*

Dans [187, 188], Wilson décrit très précisément comment mettre au point une cryptanalyse de ce schéma. En particulier, avec les paramètres exacts, les calculs peuvent même être menés à la main. Évidemment, il faut revoir à la hausse ces paramètres si l'on considère les moyens de calcul actuels.

Pour être précis, Littlewood définit un générateur pseudo-aléatoire de nombres et l'utilise comme *one-time pad* pour obtenir un chiffrement par flots. Avant de donner

la formalisation du générateur de Littlewood que nous prendrons, rappelons ce qu'est un générateur pseudo-aléatoire de nombres cryptographiquement sûr. Intuitivement, les bits doivent être facilement calculables et le k-ième bit doit être imprévisible même sous l'hypothèse que l'adversaire (qui a une capacité de calcul polynomiale) a déjà vu les $(k-1)$ bits précédents. De façon plus forte, la suite des bits ne doit pas être discernable d'une suite de bits vraiment aléatoire. Nous donnons ici la définition formelle d'un générateur pseudo-aléatoire de bits [67].

Définition 32 (Générateur pseudo-aléatoire). *Un algorithme déterministe $\mathcal{G}$ de complexité polynomiale en temps et renvoyant des bits est un générateur pseudo-aléatoire de bits s'il existe une fonction strictement croissante $l : \mathbb{N} \to \mathbb{N}$ telle que pour n'importe quel algorithme probabiliste $\mathcal{A}$ de complexité polynomiale en temps et prenant une suite de bits en entrée et renvoyant un bit, pour n'importe quel polynôme P à valeurs positives et pour n'importe quel entier k suffisamment grand,*

$$\left|\Pr[\mathcal{A}(\mathcal{G}(U_k)) = 1] - \Pr[\mathcal{A}(U_{l(k)}) = 1]\right| < \frac{1}{P(k)},$$

où U_n est la distribution uniforme sur l'ensemble $\{0,1\}^n$ et les probabilités sont prises relativement à U_k et $U_{l(k)}$ ainsi qu'à l'aléa lié à l'algorithme $\mathcal{A}$.

Nous formalisons maintenant le générateur pseudo-aléatoire de Littlewood.

Définition 33 (Générateur pseudo-aléatoire de Littlewood). *Soient n, n', m et t des paramètres entiers pouvant être ajustés pour atteindre un niveau de sécurité désiré.*

1. *Choisir un entier secret N de n bits.*
2. *Pour i de 0 à t, renvoyer les bits fractionnaires des rangs $n'+1$ à $n'+m$ de $\log(N+i)$, c'est-à-dire :*
$$\left\lfloor 2^{n'+m} \log(N+i) \right\rfloor \mod 2^m.$$

Intuitivement, nous avons une table de logarithmes dont les entrées sont écrites en lignes, et nous renvoyons ligne par ligne le bloc de bits compris entre les lignes N et $N+t$ et entre les colonnes $n'+1$ et $n'+m$. La figure 3.1 donne un exemple de ce procédé.

La faiblesse principale de ce générateur réside dans le fait que les logarithmes de deux grands nombres sont très proches, et en utilisant cette remarque, même avec un très petit nombre de bits de sortie connus, l'adversaire peut calculer les bits de sortie suivants. Notre attaque repose sur le lemme suivant :

Lemme 53. *Soit Δ l'opérateur sur les suites défini par : $\Delta(u)_k = u_{k+1} - u_k$, et soit $\Delta^{(i)}$ la composée i fois cet opérateur. Soient N un entier de n bits et $a_k = \log(N+k)$ pour $k \geqslant 0$. Alors pour tout $i \geqslant 1$ et tout $k \geqslant 0$, on a :*

$$\left|\Delta^{(i)}(a_k)\right| \leqslant \frac{(i-1)!}{2^{i(n-1)}}.$$

Démonstration. On commence par étendre Δ aux fonctions : $[\Delta(f)](x) = f(x+1) - f(x)$. Soit $f(x) = \log(N+x)$, de telle sorte que $a_k = f(k)$. On applique le théorème des valeurs intermédiaires plusieurs fois de suite :

N	$\log(N)$		
30	11.0110011	010110100	1101111
31	11.0110111	100011001	1100100
32	11.0111011	100111010	0111011
33	11.0111111	100011011	0001111
34	11.1000011	010111111	1001000
35	11.1000111	000101011	0100101

010110100 | 100011001 | 100111010 | 100011011 | 010111111 | 000101011

FIG. 3.1 – Un exemple de fonctionnement du générateur pseudo-aléatoire de bits de Littlewood, avec $N = 30, t = 5, n' = 7$ et $m = 9$.

$$\begin{aligned}
|\Delta^{(i)}[f(k)]| &= |\Delta^{(i-1)}[f(k+1)] - \Delta^{(i-1)}[f(k)]| \\
&= |[\Delta^{(i-1)}(f)](k+1) - [\Delta^{(i-1)}(f)](k)| \\
&\leqslant \max(|[\Delta^{(i-1)}(f)]'(x)|\ ,\ x \in [k, k+1]) \\
&\leqslant \max(|[\Delta^{(i-1)}(f')](x)|\ ,\ x \in [k, k+1]) \\
&\leqslant \max(|[\Delta^{(i-2)}(f'')](x)|\ ,\ x \in [k, k+2]) \\
&\vdots \\
&\leqslant \max(|f^{(i)}(x)|\ ,\ x \in [k, k+i]) \\
&\leqslant \frac{(i-1)!}{N^i},
\end{aligned}$$

car $f^{(i)}(x) = (-1)^{i+1}\frac{(i-1)!}{x^i}$. Le résultat provient du fait que $N \geqslant 2^{n-1}$. □

Du lemme précédent on déduit immédiatement que la $(i+1)$-ème sortie du générateur, c'est-à-dire $u_{i+1} = \lfloor 2^{n'+m} \log(N+i+1) \rfloor \mod 2^m$, peut être calculée à partir des sorties 0 à i, en utilisant la formule :

$$u_{i+1} = (u_{i+1} - \Delta^{(i)}(u_0)) + \Delta^{(i)}(u_0).$$

Ce calcul, illustré à la figure 3.2, nécessite $O(i^2)$ soustractions. La quantité $u_{i+1} - \Delta^{(i)}(u_0)$ dépend de $u_0, \ldots, u_i$ (qui sont connus) mais non de u_{i+1} (que l'on cherche), et on peut borner la valeur de $\Delta^{(i)}(u_0)$ grâce au lemme précédent. L'attaque consiste à calculer $u_{i+1} - \Delta^{(i)}(u_0)$ à l'aide de $u_0, \ldots, u_i$ et à considérer négligeable le terme $\Delta^{(i)}(u_0)$. Du fait de l'erreur cumulée provenant des différents arrondis, on obtient : $|\Delta^{(i)}(u_0)| \leqslant 2^{n'+m}|\Delta^{(i)}(a_0)| + 2^i$. Pour retrouver le premier bit de sortie suivant ceux correspondant à u_{i+1}, il suffit que $|\Delta^{(i)}(u_0)| \leqslant 2^{m-1}$, ce qui est impliqué par la condition suffisante :

$$|2^{n'+m}\Delta^{(i)}(\log N)| \leqslant 2^{m-1} - 2^i.$$

On en déduit que le générateur de Littlewood décrit à la définition 33 n'est pas cryptographiquement sûr à partir du moment où il existe i tel que

$$m - 2 \geqslant i \geqslant \frac{n' + 2 + \log((i-1)!)}{n-1}.$$

Si m est petit, il se pourrait qu'un tel i n'existe pas, et le phénomène de la propagation des erreurs numériques empêche l'attaque. Il en est de même si n' est très grand. Ces deux cas sont peu réalistes puisqu'il faudrait soit calculer beaucoup de $\log x$ différents, soit beaucoup de bits de chaque $\log x$ pour renvoyer finalement peu de bits. Cette attaque semble fonctionner pour n'importe quels paramètres réalistes. Par exemple, si $n = m = 100$, elle peut être effectuée tant que $n' \leqslant 9196$; et si $n = m = 1000$, cette borne sur n' devient alors 988500.

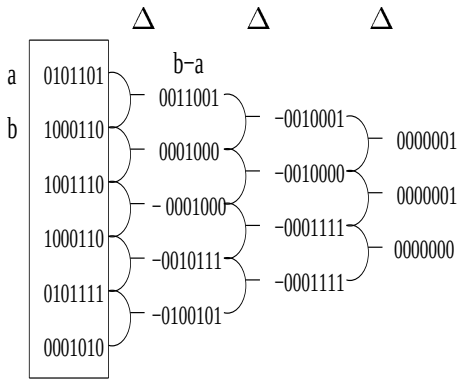


FIG. 3.2 – Applications successives de l'opérateur Δ à la séquence obtenue à la figure 3.1. Dans la dernière colonne, les bits dominants de chaque ligne sont nuls.

Remarque : Le même raisonnement est valable pour une très grande famille de fonctions f à la place de $f = \log$. Il suffit en effet que lorsque N est suffisamment grand, il existe deux constantes $a > 0$ et b telles que les dérivées successives de f satisfassent :

$$f^{(i)}(x) = O\left(\frac{1}{N^{ai+b}}\right), \forall x \geqslant N, \forall i \geqslant i_0.$$

En particulier, on peut tout à fait remplacer log par un polynôme, une fraction rationnelle, ou une racine α-ième, cela n'empêchera pas cette attaque.

3.2 Une modification du schéma de Littlewood.

Nous présentons ici une modification naturelle du générateur pseudo-aléatoire de bits de Littlewood, qui résiste à l'attaque très facile décrite ci-dessus. Le générateur obtenu ressemble alors beaucoup au générateur pseudo-aléatoire décrit dans [28]. Avant d'expliquer le nouveau générateur, nous avons besoin de quelques notations et résultats élémentaires sur le calcul des fonctions mathématiques.

Définition 34 (Fonction réelle calculable en temps polynomial). *Soient $a < b \in \mathbb{R}$. Une fonction $f : [a,b] \to \mathbb{R}$ est calculable en temps polynomial s'il existe un algorithme déterministe $\mathcal{A}$ et un polynôme P tels que pour n'importe quel $x \in [a,b]_n$, l'algorithme $\mathcal{A}$ calcule les n bits les plus significatifs de $f(x)$ en au plus $P(n)$ opérations élémentaires.*

Bien entendu, des fonctions telles que $\ln, \exp, \sin, \cos, \tan, \sin^{-1}, \cos^{-1}, \tan^{-1}$ sont calculables en temps polynomial pour n'importe quel intervalle. Brent montre dans [29] que calculer $\cos, \sin$, arctan, ln et exp peut être fait en temps quasi-linéaire en utilisant la multiplication rapide [158].

Théorème 32 (Brent, 1976). *Si π et $\ln 2$ sont pré-calculés, la fonction mathématique élémentaire $f(x)$ peut être évaluée avec une précision n en $(K+o(1))M(n)\log n$ opérations élémentaires, avec :*

$$K = \begin{cases} 13 & si \quad f(x) = \ln(x) \ ou \ \exp(x), \\ 34 & si \quad f(x) = \arctan(x) \ ou \ \sin(x) \ ou \ \cos(x). \end{cases}$$

3.2.1 Randomisation du générateur de Littlewood.

Le problème principal du générateur pseudo-aléatoire de bits de Littlewood réside dans le fait que l'on prend des lignes de la table de logarithmes qui sont consécutives. Celles-ci sont trop corrélées quand N est grand. Une façon naturelle de résoudre ce problème est d'utiliser aléatoirement les lignes. L'objet que nous définissons ci-dessous est un générateur pseudo-aléatoire conceptuellement très proche de celui décrit dans [28]. En particulier, les attaques contre ces deux générateurs se ressemblent.

Définition 35 (Problème du nombre réel secret). *Soit $f : [0,1] \to [0,1]$ une fonction réelle calculable en temps polynomial, comme par exemple* sin. *Soient n, n', m et t des paramètres de sécurité. Soit $\alpha \in [0, \frac{1}{2}]_n$ un nombre secret. Le problème du nombre réel secret consiste à retrouver α à partir de la connaissance de :*

$$\left(x_i, \mathrm{bits}_{(n'+1)..(n'+m)}[f(\alpha + x_i)] \right)_{i=0..t},$$

où les x_i sont choisis uniformément et indépendamment dans $[0, \frac{1}{2}]_n$.

Définition 36 (Générateur randomisé de Littlewood). *Avec les mêmes notations, le générateur pseudo-aléatoire généralisé de Littlewood est la fonction d'extension suivante :*

$$(x_0, \ldots, x_t) \in \left[0, \frac{1}{2}\right]_n^{t+1} \quad \to \quad < x_0, \ldots, x_t, b_0, \ldots, b_t >,$$

avec $b_i = \mathrm{bits}_{(n'+1)..(n'+m)}[f(\alpha + x_i)]$ pour i de 0 à t.

La sécurité de ce générateur pseudo-aléatoire de bits repose directement sur la difficulté du problème du nombre réel secret. De façon classique, il est possible de rebrancher le générateur sur les précédentes sorties, en utilisant la technique décrite dans [24]. Remarquons aussi que si les x_i ne sont pas choisis indépendamment, en particulier si $x_{i+1} = x_i + 2^{-n}$, alors l'attaque naïve précédemment décrite pourrait encore fonctionner.

Les paramètres de sécurité doivent satisfaire certaines bornes pour que le générateur échappe à la recherche exhaustive et à l'attaque des anniversaires. On peut faire une recherche exhaustive pour essayer de deviner les n' premiers bits fractionnaires de $y = f(\alpha + x_i)$, et alors en calculant $f^{-1}(y)$ (si f^{-1} est une fonction réelle calculable en temps polynomial) on peut retrouver α. Cette attaque a une complexité de l'ordre de $2^{n'}$, et donne donc la borne $n' \geqslant s$ si l'on veut une sécurité en au moins 2^s. L'attaque des anniversaires donne la borne $m \geqslant 2s$.

3.3 Une attaque reposant sur l'approximation polynomiale.

L'attaque proposée ici casse le générateur de la définition 36. Elle est cependant heuristique car elle suppose que les vecteurs renvoyés par l'algorithme LLL auquel on a donné en entrée un certain type de réseaux, aient tous à peu près la même longueur. Elle consiste en un algorithme qui résout le problème du nombre réel secret. La première étape est l'approximation de f par un polynôme à coefficients réels sur tout son domaine de définition $[0, 1]$. Puis on réduit le problème du nombre réel secret à la recherche de petites racines modulaires communes de plusieurs polynômes multivariés. On utilise finalement la méthode de Coppersmith pour résoudre ce dernier problème et retrouver le secret α. La technique développée dans cette attaque est très similaire à l'algorithme que nous avons proposé pour trouver les pires cas pour l'arrondi des fonctions mathématiques élémentaires, au chapitre III-1. La partie « réseaux » de cette attaque est très similaire à celle utilisée dans [28].

Nous allons montrer le résultat suivant :

Théorème 33. *Soit $f : [0, 1] \to [0, 1]$ une fonction $\mathcal{C}^\infty$ calculable en temps polynomial. On suppose que lorsque d tend vers l'infini, $\max_{x\in[0,1]} |f^{(d)}(x)| = O(d!2^d)$. Alors il existe un algorithme de complexité polynomiale en n, n' et m, tel que si t est plus grand qu'une borne croissant de façon polynomiale avec n, n' et m, alors, sous une heuristique qui est donnée plus bas, il résout le problème du nombre réel secret.*

3.3.1 Approcher f par un polynôme.

L'approximation des fonctions continues par des polynômes a été un centre d'intérêt pour les mathématiciens depuis longtemps, mais il a vu naître sa mise en pratique algorithmique seulement récemment. On se référera à l'ouvrage de Jean-Michel Muller [134] pour de plus amples détails sur ce sujet. Nous considérons ici des approximations pour la norme infinie, c'est-à-dire :

$$\|f - P\|_\infty = \sup(|f(x) - P(x)| \ , \ x \in [0, 1]).$$

Il est bien connu que d'un point de vue théorique il existe un polynôme qui approche une fonction donnée autant que voulu :

Théorème 34 (Weierstrass,1885). *Soit f une fonction continue. Pour tout $\varepsilon > 0$, il existe un polynôme P tel que $\|f - P\|_\infty \leqslant \varepsilon$.*

Malheureusement, ce théorème n'est pas constructif (il ne dit pas comment trouver ces polynômes), et il ne donne pas non plus la vitesse de convergence en fonction du degré du polynôme. Une méthode très efficace pour calculer une approximation d'une fonction $\mathcal{C}^\infty$ est d'utiliser les polynômes de Chebychev. Le polynôme de Chebychev de degré d approchant f peut être calculé avec des algorithmes classiques d'analyse numérique. Plus précisément, on peut calculer, en temps polynomial en d et k, des approximations relatives à 2^{-k} près des coefficients de l'approximant de Chebychev de degré d de la fonction f. De plus, quand d augmente, les approximants de Chebychev convergent très vite. Pour plus de détails, le lecteur est renvoyé à [17], où l'on peut trouver le théorème suivant :

Théorème 35 (Approximation de Chebychev). *Soit $f : [0,1] \to [0,1]$ une fonction $\mathcal{C}^\infty$, et P_d l'approximant de Chebychev de degré d de f. Alors :*

$$\|f - P_d\|_\infty \leqslant \frac{\|f^{(d+1)}\|_\infty}{(d+1)!2^{2d+1}}.$$

Pour notre problème, nous prendrons le plus petit degré d tel que $\|f - P_d\|_\infty \leqslant \frac{1}{2^{n'+m}}$. On déduit directement de ce qui précède que d est borné de façon polynomiale en n' et m, et que l'on peut construire des approximations des coefficients de P_d avec un nombre polynomial de bits en un temps polynomial en tous les différents paramètres.

3.3.2 D'un problème sur les réels à un problème sur les entiers.

Supposons maintenant qu'un adversaire connaisse les paires (x_i, b_i) pour $i \in [|1,t|]$, avec

$$b_i = \mathrm{bits}_{(n'+1)..(n'+m)}[f(\alpha + x_i)] \ \in \ [|0, 2^m - 1|].$$

Le lemme qui suit montre comment transformer chaque relation en une équation satisfaite par un polynôme multivarié à coefficients et inconnues entiers :

Lemme 54. *Soient $x_0 \in [0,1]_n$ et $b = \mathrm{bits}_{(n'+1)..(n'+m)}[f(x_0)]$. Soient P un polynôme de degré d qui satisfait $\|f - P\|_\infty \leqslant 2^{-(n'+m)}$, et*[41] *$P'(x) = \left\lfloor 2^{n'+m}P \right\rceil(x)$. Soit $Q(t) = 2^{dn}P'\left(\frac{t}{2^n}\right)$. Alors Q est un polynôme de degré d à coefficients entiers, et :*

$$Q(2^n x_0) - 2^{dn} b = 2^{dn+m} y + z,$$

avec y et z des inconnues entières qui satisfont $0 \leqslant y < 2^{n'}$ et $|z| \leqslant (d+5)2^{dn-1}$.

Démonstration. Avec la définition de P', on a $|P'(x) - 2^{n'+m}P(x)| \leqslant \frac{d+1}{2}$ pour tout x de $[0,1]$. L'inégalité triangulaire nous donne donc qu'il existe un entier $0 \leqslant y < 2^{n'}$ tel que $|P'(x_0) - 2^m y - b| \leqslant \frac{d+1}{2} + 2$. Finalement, $|Q(2^n x_0) - 2^{dn+m} y - 2^{dn} b| \leqslant (d+5)2^{dn-1}$. □

Nous appliquons maintenant le lemme 54 au polynôme P_d trouvé durant la phase d'approximation polynomiale de l'attaque. Soit Q le polynôme de degré d sur $\mathbb{Z}$ ainsi obtenu. Pour chaque paire (x_i, b_i), on obtient une équation :

$$Q(2^n(\alpha + x_i)) - 2^{dn} b_i - 2^{dn+m} z_i + y_i = 0 \quad (E_i),$$

[41] Cette notation signifie que l'on arrondit chaque coefficient de $2^{n'+m}P$ à l'entier le plus proche.

avec α, y_i et z_i des inconnues qui satisfont $0 \leqslant z_i < 2^{n'}$ et $|y_i| \leqslant (d+5)2^{dn-1}$. Nous transformons maintenant les équations (E_i) sur $\mathbb{Z}$ en des équations modulaires :

$$Q_i(a) + y_i = 0 \mod A \quad (E'_i),$$

avec $A = 2^{dn+m}$, $a = 2^n\alpha \in [|0, 2^{n-1}|]$, $Q_i(a) = Q(a + 2^n x_i) - 2^{dn} b_i$ et $|y_i| \leqslant (d+5)2^{dn-1}$. Il s'agit donc maintenant de trouver a, sachant que $Q_i(a)$ est petit modulo A, avec des polynômes Q_i connus. Nous nous retrouvons tout à fait dans le contexte général de la méthode de Coppersmith.

Remarque : Comme il avait été annoncé plus haut, un nombre polynomial de bits des coefficients de P_d suffit pour obtenir finalement le bon polynôme Q. En effet, à partir de la définition de P' dans le lemme 54, il suffit que $n' + m + 1$ bits fractionnaires de chaque coefficient de P_d aient été calculés, ou plus précisément que l'on ait une erreur sur chaque coefficient bornée par $2^{-n'-m}$.

3.3.3 Comment retrouver a avec de la réduction de réseau.

La dernière étape pour retrouver le secret α, ou de façon équivalente l'entier a, est d'utiliser une réduction d'un réseau lié aux équations (E'_i) trouvées à l'étape d'approximation polynomiale. On réduit le problème de trouver a à la recherche de vecteurs courts linéairement indépendants dans un réseau euclidien, problème que l'on résout à l'aide de l'algorithme LLL. Plus précisément, on construit un réseau pour lequel les vecteurs courts nous fournissent des équations polynomiales en a et les y_i, sur les entiers (sans modulo). Sous l'heuristique que l'on a suffisamment de ces équations sur les entiers, une élimination de variables par algèbre linéaire permet d'obtenir un polynôme non-nul à une seule variable et coefficients entiers. Les racines de ce polynôme sont alors calculées, et elles donnent directement a.

Supposons que l'on ait t équations $(E'_1), \ldots, (E'_t)$. Soit $q_i(x, \nu_1, \ldots, \nu_t) = Q_i(x) + \nu_i$ pour i dans $[|1, t|]$. On sait que :

$$\forall i \in [|1, t|], q_i(a, y_1, \ldots, y_t) = Q_i(a) + y_i = 0 \mod A,$$

avec $A = 2^{dn+m}$ et $|y_i| \leqslant Y$ pour $i \leqslant t$ avec $Y = (d+5)2^{dn-1}$. On construit alors le réseau qui leur correspond. Par exemple, en prenant $d = 3$ et $t = 4$, on obtient un réseau de la forme :

	1	x	x^2	x^3	ν_1	ν_2	ν_3	ν_4
(A)	A							
(AX)		AX						
(AX^2)			AX^2					
(AX^3)				AX^3				
(E'_1)	$-$	$-$	$-$	$-$	Y			
(E'_2)	$-$	$-$	$-$	$-$		Y		
(E'_3)	$-$	$-$	$-$	$-$			Y	
(E'_4)	$-$	$-$	$-$	$-$				Y

,

où $X = 2^n$ est la borne a priori sur a. À partir de la définition des q_i, nous savons que les entrées de la colonne correspondant à x^d sont identiques pour toutes les lignes (E'_i). Il est donc intéressant de regarder le sous-réseau engendré par les lignes (AX^i) pour $0 \leqslant i \leqslant d-1$ et les lignes $(E'_j) - (E'_1)$ pour $2 \leqslant j \leqslant t$. Cela fournit un réseau de dimension $d+t-1$ et de déterminant :

$$X^{\frac{d(d-1)}{2}} Y^{t-1} A^d.$$

En prenant $t+1$ petits vecteurs de ce réseau et en annulant les colonnes correspondant à $\nu_1, \ldots, \nu_t$ à l'aide d'une étape d'algèbre linéaire, on calcule ensuite les racines du polynôme en x de degré $d-1$ que l'on a ainsi obtenu, et a est l'une d'entre elles. Obtenir un vecteur suffisamment court est faisable tant que (cela nous est donné par le théorème 3) :

$$X^{\frac{d(d-1)}{2}} Y^{t-1} \ll A^{t-1}.$$

On peut maintenant arguer du fait qu'en général, les vecteurs courts linéairement indépendants ont des longueurs similaires. Plus précisément, pour un réseau « aléatoire »[42] L de dimension l, les l minima successifs de L ont des longueurs de l'ordre de $\det(L)^{1/l}$, à des facteurs polynomiaux en d près. Le théorème 3 donne une borne très pessimiste pour les longueurs des différents vecteurs renvoyés, par rapport à ce que l'on observe en pratique. Une version « probabiliste » pourrait être[43] :

Étant donnée une base aléatoire $[\mathbf{b}_1, \ldots, \mathbf{b}_l]$ *d'un réseau* $L \subset \mathbb{Z}$*, l'algorithme LLL renvoie avec grande probabilité (par rapport à l'aspect aléatoire du réseau) et en temps polynomial en la taille de l'entrée, une base* $(\mathbf{v}_1, \ldots, \mathbf{v}_l)$ *qui satisfait :*

$$\forall i \in [|1, l|], \|\mathbf{v}_i\| \approx \det(L)^{\frac{1}{l}}.$$

Sous cette dernière heuristique, on s'attend donc à pouvoir trouver a tant que :

$$X^{\frac{d(d-1)}{2}} Y^{t-1} \ll A^{t-1},$$

ce qui donne en remplaçant X, Y et A par leurs valeurs et en prenant le logarithme :

$$nd(d-1) \ll 2(t-1)m,$$

qui devient valide quand $t \gg \frac{nd(d-1)}{2m}$.

3.4 Conclusion et question ouverte.

Le générateur pseudo-aléatoire de bits proposé par Littlewood et sa généralisation ont montré leurs faiblesses cryptographiques. Une simple attaque reposant sur des différences

[42] Le mot « aléatoire » est entre guillemets parce qu'il est difficile de définir proprement ce qu'est un réseau aléatoire (voir [8] pour une bonne façon de faire cela), le fait énoncé n'est pas facilement prouvable pour des réseaux aléatoires, et il n'est pas très clair à quel point nos réseaux sont aléatoires.

[43] Nous avons déjà fait des hypothèses similaires au chapitre III-1 lorsque la dimension des réseaux ne croissait pas (au paragraphe 1.3.3), et au chapitre III-2 (à la section 2.3).

en cascade permet de casser le schéma original pour une large plage de paramètres, et une méthode reposant sur de l'approximation polynomiale et de la réduction de réseaux permet de casser le schéma randomisé. Les fonctions réelles semblent ainsi mal appropriées pour construire des primitives cryptographiques. Cependant, tout n'est pas à jeter dans cette idée. Le schéma de Littlewood est intimement lié au problème du dilemme du fabricant de tables (décrit au chapitre III-1), que nous rappelons ici :

DFT : Soient n, n', m des paramètres entiers. On se donne une fonction $\mathcal{C}^\infty$ $f : [0,1] \to [0,1]$ calculable en temps polynomial. Le but est de trouver toutes les solutions $x \in [0,1]_n$ de l'équation :

$$\text{bits}_{(n'+1)..(n'+m)}[f(x)] = 0.$$

Nous avons étudié cette équation dans le détail au chapitre III-1 car elle est la clé pour l'arrondi correct des fonctions mathématiques élémentaires. Nous avons vu que quand m est de l'ordre de grandeur de n, alors le meilleur algorithme connu (donné au chapitre III-1) a une complexité en $\mathcal{P}\text{ol}(n)2^{n/2+o(1)}$. Ceci motive la question suivante, illustrée par la figure 3.3 :

Problème ouvert 1. *Soit $n \geqslant 1$ un entier. Soit $f : [0,1] \to [0,1]$ une fonction $\mathcal{C}^\infty$ calculable en temps polynomial. On définit la fonction $\Phi : [|0, 2^n|] \to [|0, 2^n|]$ par :*

$$\Phi(x) = \text{bits}_{(n+1)..(2n)} \left[f\left(\frac{x}{2^n}\right) \right].$$

Peut-on considérer Φ comme une fonction à sens unique ?

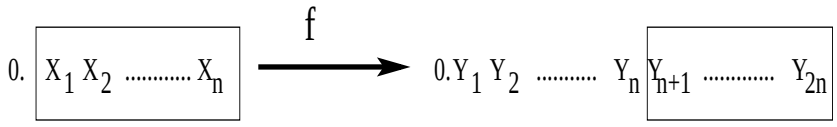


FIG. 3.3 – Une proposition de fonction à sens unique.

Les arguments ci-dessus suggèrent que bien que Φ soit facilement calculable, elle serait difficile à inverser. Un gros avantage pratique de cette fonction est qu'elle est très facile à implanter à partir d'un logiciel de calcul multi-précision. Asymptotiquement, calculer Φ se fait quasiment aussi rapidement que multiplier deux entiers, mais le meilleur algorithme pour l'inverser est exponentiel. S'il s'agit en effet du meilleur algorithme, cette fonction serait tout à fait intéressante. Par exemple, pour atteindre une sécurité de 2^{80}, $n = 160$ suffit, ce qui implique de calculer $f(\frac{x}{2^n})$ avec 320 bits de précision. Remarquons finalement que même pour la fonction $x \to 1/x$, la fonction Φ associée semble difficile à inverser : en effet l'algorithme classique de reconstruction rationnelle (voir par exemple [64] ou [24]) nécessite la connaissance de $2n$ bits consécutifs, et ici on ne dispose que de n bits. Dans ce contexte, Shparlinski [163], montre que si l'on a un peu moins de n bits, alors on ne

peut rien dire sur x. Entre n et $2n$ bits rien n'est connu. Nous proposons ici deux défis, le premier pour la fonction cos, le deuxième pour la fonction inverse.

Trouver $x_1, x_2 \in [|2^{159}, 2^{160} - 1|]$ tels que :

$$\begin{aligned} \mathrm{bits}_{161..320}\left(\cos\left(\frac{x_1}{2^{160}}\right)\right) &= 858386833426836933645234112375633941836693318987, \\ \mathrm{bits}_{161..320}\left(\frac{2^{160}}{x_2}\right) &= 946436456336607322266843999547133410116713716156. \end{aligned}$$

Conclusion.

Dans cette thèse, nous avons étudié l'algorithmique des réseaux euclidiens et son application à un problème lié à l'arithmétique des ordinateurs, le dilemme du fabricant de tables. Nous avons adopté une approche verticale, en nous intéressant aussi bien à l'outil algorithmique, la géométrie des nombres, qu'à l'application que nous avions choisie, c'est-à-dire le calcul des pires cas pour l'arrondi d'une fonction, quand le mode d'arrondi, la fonction et la précision sont donnés. Pour aboutir aux résultats, nous avons fait appel à des techniques très diverses, du classique calcul de complexité à la méthode de Coppersmith, en passant par quelques considérations de géométrie élémentaire et des calculs d'erreurs numériques.

D'un point de vue algorithmique, nos contributions sont les suivantes. Nous avons donné un algorithme simple de calcul de pgcd en temps quasi-linéaire (plus simple que ceux qui existaient déjà, mais légèrement plus lent en pratique). Pour aboutir à ce résultat, nous avons introduit une nouvelle division binaire, naturelle d'un point de vue 2-adique. Un calcul de pgcd peut être considéré comme une réduction d'un réseau de dimension 1. En petite dimension (inférieure ou égale à 4), nous avons généralisé de manière naturelle et unifié les descriptions des algorithmes de Lagrange (en dimension 2), Vallée et Semaev (en dimension 3). Nous avons démontré la correction de cet algorithme unificateur et prouvé qu'il avait une complexité quadratique en la taille de la donnée. Cette amélioration a été rendue possible à l'aide d'une analyse précise des propriétés géométriques des réseaux en petite dimension, en particulier de leurs cellules de Voronoï, ainsi qu'à l'aide d'une analyse amortie des coûts des étapes successives. En dimension quelconque, nous avons construit une variante, utilisant de l'arithmétique flottante, du célèbre algorithme LLL. Cette variante est la seule connue à ce jour qui admette une complexité quadratique en dimension fixée comme cela est le cas pour l'algorithme d'Euclide naïf, l'algorithme de Lagrange et sa généralisation en petite dimension. Pour analyser cet algorithme, nous avons réalisé un calcul de la stabilité numérique de l'algorithme de factorisation de Cholesky, et nous avons adapté au cas d'une dimension quelconque l'analyse amortie que nous avions effectuée en petite dimension. Nous avons implanté cet algorithme, et notre code est compétitif par rapport à ceux disponibles dans les principaux logiciels d'algorithmique de la théorie des nombres (NTL, Magma, Pari GP). Cette implantation nous a permis de faire un certain nombre d'observations expérimentales sur le comportement pratique de l'algorithme LLL, qui serviront peut-être à l'améliorer.

En ce qui concerne l'étude des pires cas pour l'arrondi des fonctions mathématiques, nous avons amélioré la méthode de Lefèvre en utilisant la technique de Coppersmith

pour trouver les petites racines de polynômes modulaires multivariés, qui elle-même utilise la réduction des réseaux. Par ce biais, les améliorations apportées à l'algorithmique de la géométrie des nombres ont naturellement servi pour expérimenter l'algorithme de recherche de pires cas. Pour optimiser l'utilisation de la technique de Coppersmith dans notre situation, nous avons construit une méthode pour borner le déterminant des matrices rectangulaires qui apparaissent. Pour une fonction f, un mode d'arrondi et une précision n fixés, notre algorithme trouvera tous les $x \in [1/2, 1[$ avec n bits après la virgule qui maximisent la difficulté d'arrondir $f(x)$, en temps heuristique $2^{n/2+\varepsilon}$. Ces valeurs de x sont celles pour lesquelles il faudra un maximum de bits après le bit d'arrondi pour garantir l'arrondi correct de $f(x)$. Cette méthode permet aussi de montrer qu'avec les $\approx n^2$ premiers bits de $f(x)$ sauf les n premiers, on peut déterminer tous les $x \in [1/2, 1[$ avec au plus n bits fractionnaires donnant cette séquence, en temps heuristique polynomial en n. Cela permet de quantifier l'alea des suites de bits générées par les fonctions mathématiques : si significativement moins de n^2 bits sont donnés, il est vraisemblablement difficile de déterminer la graine x, les bits significatifs et les bits qui suivent la séquence donnée (en tout cas, la présente méthode aura une complexité non polynomiale, et il semble que ce soit la meilleure connue à ce jour pour résoudre ce type de questions) ; si $\approx n^2$ bits sont donnés, alors il devient « facile » (on peut le faire en temps polynomial) de retrouver la graine, les bits significatifs manquants et les bits suivants. En particulier, pour les fonctions pour lesquelles la méthode fonctionne, les suites infinies de bits générées ne sont pas aléatoires. Nous avons généralisé notre méthode pour trouver les mauvais cas simultanés de deux fonctions, ce qui nous a permis d'améliorer la méthode des tables de Gal, qui est une étape centrale dans les implantations de certaines fonctions mathématiques usuelles en précision fixée, comme par exemple en double précision. Ces techniques de recherche de mauvais cas pour l'arrondi et de construction des tables de Gal ont fait l'objet d'implantations expérimentales. Nous avons aussi utilisé la méthode développée pour résoudre le dilemme du fabricant de tables pour casser un cryptosystème qui avait été proposé par J. E. Littlewood dans les années 1950.

De nombreuses extensions des travaux décrits dans cette thèse sont possibles. Nous en avons donné certaines dans les conclusions de chaque chapitre. Du point de vue de l'efficacité de la méthode de résolution du dilemme du fabricant de tables, l'écrasante majorité du temps d'exécution est consacrée à des réductions de réseaux. Il s'agirait donc de construire une variante plus efficace de l'algorithme LLL, ou de diminuer de manière drastique la taille des entrées données à ce dernier. Une autre question ouverte liée à notre méthode de résolution du dilemme du fabricant de tables est de déterminer quand elle est valide, c'est-à-dire quand l'hypothèse heuristique s'avère correcte. Cela permettrait d'expliquer certains échecs observés en pratique, et peut-être de les contourner, en particulier dans le cas de fonctions algébriques. Un point délicat consisterait à déterminer dans quelle mesure la méthode de Coppersmith pour les polynômes modulaires multivariés, elle-même heuristique, peut être prouvée correcte. Cette question a déjà été soulevée plusieurs fois en cryptologie, dans le contexte d'attaques contre des variantes efficaces du schéma de chiffrement RSA [26, 46, 123].

Des améliorations importantes en algorithmique des réseaux sont elles aussi envisageables. L'algorithme L^2 que nous avons introduit représente un progrès significatif, mais

il reste avant tout un algorithme naïf : d'une part il généralise en dimension quelconque la version la plus naïve de l'algorithme d'Euclide, et il repose sur une algèbre linéaire élémentaire. La marge de manœuvre pour obtenir des variantes plus efficaces est sans aucun doute considérable. Une première piste serait de construire un algorithme LLL de complexité quasi-linéaire en dimension fixée quelconque, de manière analogue aux algorithmes de Knuth [91] et Schönhage [155] pour le pgcd, et de Yap [189] et Schönhage [157] pour la réduction de réseaux planaires. Dans cette direction, une réponse a déjà été apportée par Eisenbrand et Rote [57], mais la complexité de leur algorithme explose avec la dimension. Une amélioration de ce type serait intéressante pour des bases de dimensions modérées mais avec des vecteurs extrêmement longs. Une autre voie serait d'adapter les variantes de l'algorithme LLL de Schönhage [156], Storjohann [174], Koy et Schnorr [94, 95], à l'algorithme L^2. En évitant d'effectuer les proprifications entièrement à chaque itération de boucle, ces variantes économisent de nombreuses opérations arithmétiques. Elles sont particulièrement pertinentes en vue de réduire des bases en grande dimension. Enfin, il serait intéressant d'améliorer les algorithmes de réduction calculant des bases de meilleure qualité que celles renvoyées par LLL, comme par exemple les algorithmes de Kannan [88], Helfrich [75], Schnorr [150], Ajtai, Kumar et Sivakumar [11]. Obtenir des bases d'excellente qualité semble être la voie privilégiée pour casser plusieurs cryptosystèmes, parmi lesquels on peut citer NTRU [80]. De manière plus générale, la place centrale des réseaux euclidiens en cryptologie et en théorie algorithmique des nombres n'est plus à prouver.

Bibliographie

[1] LIDIA 2.1.3. A C++ library for computational number theory. Disponible à l'url `http://www.informatik.tu-darmstadt.de/TI/LiDIA/`.

[2] IEEE Standards Committee 754. ANSI/IEEE standard 754-1985 for binary floating-point arithmetic. Disponible dans les SIGPLAN Notices, 22(2):9–25, 1987.

[3] L. Adleman. On breaking the iterated Merkle-Hellman public key cryptosystem. Dans *Proceedings of Crypto 1982*, pages 303–308. Plenum Press, 1983.

[4] M. Aharoni, S. Asaf, R. Maharik, I. Nehama, I. Nikulshin, and A. Ziv. Solving constraints on the invisible bits of the intermediate result for floating-point verification. Dans *Proceedings of the 17th IEEE Symposium on Computer Arithmetic (ARITH'17)*, pages 76–83. IEEE Computer Society Press, 2005.

[5] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[6] M. Ajtai. Generating hard instances of lattice problems (extended abstract). Dans *Proceedings of the 28th Symposium on the Theory of Computing (STOC 1996)*, pages 99–108. ACM Press, 1996.

[7] M. Ajtai. The shortest vector problem in l_2 is NP-hard for randomized reductions (extended abstract). Dans *Proceedings of the 30th Symposium on the Theory of Computing (STOC 1998)*, pages 284–293. ACM Press, 1998.

[8] M. Ajtai. Random lattices and a conjectured 0-1 law about their polynomial time computable properties. Dans *Proceedings of the 2002 Symposium on Foundations of Computer Science (FOCS 2002)*, pages 13–39. IEEE Computer Society Press, 2002.

[9] M. Ajtai. The worst-case behavior of schnorr's algorithm approximating the shortest nonzero vector in a lattice. Dans *Proceedings of the 35th Symposium on the Theory of Computing (STOC 2003)*, pages 396–406. ACM Press, 2003.

[10] M. Ajtai and C. Dwork. A public-key cryptosystem with worst-case/average-case equivalence. Dans *Proceedings of the 29th Symposium on the Theory of Computing (STOC 1997)*, pages 284–293. ACM Press, 1997.

[11] M. Ajtai, R. Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. Dans *Proceedings of the 33rd Symposium on the Theory of Computing (STOC 2001)*, pages 601–610. ACM Press, 2001.

[12] M. Ajtai, R. Kumar, and D. Sivakumar. Sampling short lattice vectors and the closest lattice vector problem. Dans *Proceedings of the 17th Annual IEEE Conference on Computational Complexity (CCC 17)*, pages 53–57, 2002.

[13] A. Akhavi. *Analyse comparative d'algorithmes de réduction sur les réseaux aléatoires*. PhD thesis, Université de Caen, 1999.

[14] A. Akhavi. Worst-case complexity of the optimal LLL algorithm. Dans *Proceedings of the 2000 Latin American Theoretical Informatics (LATIN 2000)*, volume 1776 of *Lecture Notes in Computer Science*, pages 355–366. Springer-Verlag, 2000.

[15] A. Akhavi. Random lattices, threshold phenomena and efficient reduction algorithms. *Theoretical Computer Science*, 287(2):359–385, 2002.

[16] A. Akhavi and C. Moreira dos Santos. Another view of the Gaussian algorithm. Dans *Proceedings of the 2004 Latin American Theoretical Informatics (LATIN 2004)*, volume 2976 of *Lecture Notes in Computer Science*, pages 474–487. Springer-Verlag, 2004.

[17] K. E. Atkinson. *An introduction to Numerical Analysis*. John Wiley, 1989.

[18] L. Babai. On Lovász lattice reduction and the nearest lattice point problem. *Combinatorica*, 6:1–13, 1986.

[19] W. Backes and S. Wetzel. Heuristics on lattice reduction in practice. *ACM Journal of Experimental Algorithms*, 7:1, 2002.

[20] C. Batut, K. Belabas, D. Bernardi, H. Cohen, and M. Olivier. PARI/GP computer package version 2. Disponible à l'url `http://pari.math.u-bordeaux.fr/`.

[21] Å. Björck. *Numerical Methods or Least Squares Problems*. SIAM Publications, 1996.

[22] J. Blömer and A. May. New partial key exposure attacks on RSA. Dans *Proceedings of Crypto 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 27–43. Springer-Verlag, 2003.

[23] L. Blum, M. Blum, and M. Shub. A simple unpredictable pseudo-random number generator. *SIAM Journal on Computing*, 15(2):364–383, 1986.

[24] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4):850–864, 1984.

[25] S. Boldo. *Preuves formelles en arithmétiques à virgule flottante*. PhD thesis, École Normale Supérieure de Lyon, 2004.

[26] D. Boneh and G. Durfee. Cryptanalysis of RSA with private key d less than $N^{0.292}$. *IEEE Transactions on Information Theory*, 46(4):233–260, 2000.

[27] D. Boneh, G. Durfee, and N. Howgrave-Graham. Factoring $n = pq^r$ for large r. Dans *Proceedings of Eurocrypt 1999*, volume 1666 of *Lecture Notes in Computer Science*, pages 326–337. Springer-Verlag, 1999.

[28] D. Boneh, S. Halevi, and N. Howgrave-Graham. The modular inversion hidden number problem. Dans *Proceedings of Asiacrypt 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 36–51. Springer-Verlag, 2001.

[29] R. Brent. Fast multiple precision zero-finding methods and the complexity of elementary function evaluation. *Journal of the ACM*, 23:242–251, 1976.

[30] R. P. Brent. Twenty year's analysis of the binary Euclidean algorithm. Dans *Millennial Perspectives in Computer Science : Proceedings of the 1999 Oxford-Microsoft Symposium in honour of Professor Sir Anthony Hoare*, pages 41–53, 2000.

[31] R. P. Brent, F. G. Gustavson, and D. Y. Y. Yun. Fast solution of Toeplitz systems of equations and computation of Padé approximants. *Journal of Algorithms*, 1:259–295, 1980.

[32] R. P. Brent and H. T. Kung. A systolic VLSI array for integer GCD computation. Dans *Proceedings of the 7th IEEE Symposium on Computer Arithmetic (ARITH'7)*, pages 118–125. IEEE Computer Society Press, 1985.

[33] E. F. Brickell. Solving low-density knapsacks. Dans *Proceedings of Crypto 1983*, pages 25–37. Plenum Press, 1984.

[34] E. F. Brickell. Breaking iterated knapsacks. Dans *Proceedings of Crypto 1984*, volume 196 of *Lecture Notes in Computer Science*, pages 342–358. Springer-Verlag, 1985.

[35] E. F. Brickell, J. C. Lagarias, and A. M. Odlyzko. Evaluation of Adleman's attack on multiply iterated knapsacks (abstract). Dans *Proceedings of Crypto 1983*, pages 39–42. Plenum Press, 1984.

[36] N. Brisebarre, D. Defour, P. Kornerup, J.-M. Muller, and N. Revol. A new range reduction algorithm. *IEEE Transactions on Computers*, 54(3):331–339, 2005.

[37] N. Brisebarre, J.-M. Muller, and A. Tisserand. Computing machine-efficient polynomial approximations. *À paraître dans ACM Transactions on Mathematical Software*, 2005.

[38] J. W. S. Cassels. *An Introduction to the Geometry of Numbers, 2nd edition.* Springer-Verlag, 1971.

[39] G. Cesari. Parallel implementation of schönhage's integer GCD algorithm. Dans *Proceedings of the 3rd Algorithmic Number Theory Symposium (ANTS III)*, volume 1423 of *Lecture Notes in Computer Science*, pages 65–76. Springer-Verlag, 1998.

[40] H. Cohen. *A Course in Computational Algebraic Number Theory, 2nd edition.* Springer-Verlag, 1995.

[41] H. Cohn and A. Kumar. The densest lattice in twenty-four dimensions. *American Mathematical Society*, 10:58–67, 2004.

[42] J. H. Conway and N. J. A. Sloane. *Sphere Packings, Lattices and Groups.* Springer-Verlag, 1988.

[43] D. Coppersmith. Finding a small root of a bivariate integer equation. Dans *Proceedings of Eurocrypt 1996*, volume 1070 of *Lecture Notes in Computer Science*, pages 178–189. Springer-Verlag, 1996.

[44] D. Coppersmith. Finding a small root of a univariate modular equation. Dans *Proceedings of Eurocrypt 1996*, volume 1070 of *Lecture Notes in Computer Science*, pages 155–165. Springer-Verlag, 1996.

[45] D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *Journal of Cryptology*, 10(4):233–260, 1997.

[46] D. Coppersmith. Finding small solutions to small degree polynomials. Dans *Proceedings of the 2001 Cryptography and Lattices Conference (CALC'01)*, volume 2146 of *Lecture Notes in Computer Science*, pages 20–31. Springer-Verlag, 2001.

[47] B. Daireaux. *Analyse des algorithmes d'Euclide : une approche dynamique.* PhD thesis, Université de Caen, 2005.

[48] B. Daireaux, V. Maume-Deschamps, and B. Vallée. The Lyapunov tortoise and the dyadic hare. Dans *Proceedings of the 2005 International Conference on the Analysis of Algorithm (AofA'05), DMTCS proc. AD*, pages 71–94. Discrete Mathematics and Theoretical Computer Science, 2005. Disponible à l'url `http://www.dmtcs.org/`.

[49] H. Daudé and B. Vallée. An upper bound on the average number of iterations of the LLL algorithm. *Theoretical Computer Science*, 123(1):95–115, 1994.

[50] D. Defour. Cache-optimized methods for the evaluation of elementary functions. Rapport du LIP, École Normale Supérieure de Lyon. Disponible à l'url `http://gala.univ-perp.fr/~ddefour/publications.html`, 2002.

[51] D. Defour, F. de Dinechin, and J.-M. Muller. Correctly rounded exponential function in double precision arithmetic. Dans *SPIE, 46th Annual Meeting International Symposium on Optical Science and Technology*, pages 156–167, 2001.

[52] D. Defour, G. Hanrot, V. Lefèvre, J.-M. Muller, N. Revol, and P. Zimmermann. Proposal for a standardization of mathematical function implementations in floating-point arithmetic. *Numerical Algorithms*, 37(1-4):367–375, 2004.

[53] B. N. Delone and N. N. Sandakova. Theory of stereohedra. *Trudy Mathematics Institute Steklov*, 64:28–51, 1961.

[54] J. Demmel and Y. Hida. Accurate floating-point summation. Technical report of the computer science department of the University of California Berkeley, 02-1180, 2002.

[55] F. de Dinechin, A. V. Ershov, and N. Gast. Towards the post-ultimate libm. Dans *Proceedings of the 17th IEEE Symposium on Computer Arithmetic (ARITH'17)*, pages 288–295. IEEE Computer Society Press, 2005.

[56] I. Dinur, G. Kindler, and S. Safra. Approximating CVP to within almost polynomial factors is NP-hard. Dans *Proceedings of the 1998 Symposium on Foundations of Computer Science (FOCS 1998)*, pages 99–109. IEEE Computer Society Press, 1998.

[57] F. Eisenbrand and G. Rote. Fast reduction of ternary quadratic forms. Dans *Proceedings of the 2001 Cryptography and Lattices Conference (CALC'01)*, volume 2146 of *Lecture Notes in Computer Science*, pages 32–44. Springer-Verlag, 2001.

[58] N. D. Elkies. Rational points near curves and small nonzero $|x^3 - y^2|$ via lattice reduction. Dans *Proceedings of the 4th Algorithmic Number Theory Symposium (ANTS IV)*, volume 1838 of *Lecture Notes in Computer Science*, pages 33–63. Springer-Verlag, 2000.

[59] P. van Emde Boas. Another NP-complete partition problem and the complexity of computing short vectors in a lattice. Technical report 81-04, Mathematisch Instituut, Universiteit van Amsterdam, 1981.

[60] M. Ernst, E. Jochens, A. May, and B. de Weger. Partial key exposure attacks on RSA up to full size exponents. Dans *Proceedings of Eurocrypt 2005*, number 3494 in Lecture Notes in Computer Science, pages 371–386. Springer-Verlag, 2005.

[61] L. Fousse and P. Zimmermann. Accurate summation : towards a simpler and formal proof. Dans *Proceedings of the international conference on real numbers and computers (RNC'5)*, pages 97–108, 2003.

[62] S. Gal. Computing elementary functions : a new approach for achieving high accuracy and good performance. Dans *Proceedings of Accurate Scientific Computations*, volume 235 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, 1986.

[63] S. Gal and B. Bachelis. An accurate elementary mathematical library for the IEEE floating point standard. *ACM Transactions on Mathematical Software*, 17(1):16–45, 1991.

[64] J. von zur Gathen and J. Gerhardt. *Modern Computer Algebra, 2nd edition.* Cambridge University Press, 2003.

[65] C. F. Gauss. *Disquisitiones Arithmeticæ.* Springer-Verlag, 1801.

[66] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–47, 1991.

[67] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness.* Springer-Verlag, 1999.

[68] O. Goldreich and S. Goldwasser. On the limits of non-approximability of lattice problems. *Journal of Computer and System Sciences*, 60(3):540–563, 2000.

[69] O. Goldreich, S. Goldwasser, and S. Halevi. Public-key cryptosystems from lattice reduction problems. Dans *Proceedings of Crypto 1997*, volume 1294 of *Lecture Notes in Computer Science*, pages 112–131. Springer-Verlag, 1997.

[70] D. Goldstein and A. Mayer. On the equidistribution of Hecke points. *Forum Mathematicum*, 15:165–189, 2003.

[71] G. Golub and C. van Loan. *Matrix Computations.* John Hopkins University Press, 1996.

[72] G. Gonnet. A note on finding difficult values to evaluate numerically. Disponible à l'url `http://www.inf.ethz.ch/personal/gonnet/FPAccuracy/NastyValues.ps`, 2002.

[73] T. Granlund. The GNU MP Bignum Library. Disponible à l'url `http://www.swox.com/gmp/`.

[74] L. Groetschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization.* Springer-Verlag, 1988.

[75] B. Helfrich. Algorithms to construct Minkowski reduced and Hermite reduced lattice bases. *Theoretical Computer Science*, 41:125–139, 1985.

[76] C. Hermite. Extraits de lettres de M. Hermite à M. Jacobi sur différents objets de la théorie des nombres, deuxième lettre. *Journal für die reine und angewandte Mathematik*, 40:279–290, 1850.

[77] C. Hermite. *Œuvres.* Gauthiers-Villars, 1905.

[78] N. J. Higham. The accuracy of floating point summation. *SIAM Journal on Scientific Computing*, 14:783–799, 1993.

[79] M. van Hoeij. Factoring polynomials and 0-1 vectors. Dans *Proceedings of the 2001 Cryptography and Lattices Conference (CALC'01)*, volume 2146 of *Lecture Notes in Computer Science*, pages 45–50. Springer-Verlag, 2001.

[80] J. Hoffstein, J. Pipher, and J. H. Silverman. NTRU : a ring based public key cryptosystem. Dans *Proceedings of the 3rd Algorithmic Number Theory Symposium (ANTS III)*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288. Springer-Verlag, 1998.

[81] N. Howgrave-Graham. *Computational Mathematics inspired by RSA*. PhD thesis, University of Bath, 1998.

[82] N. A. Howgrave-Graham and N. P. Smart. Lattice attacks on digital signature schemes. *Design, Codes and Cryptography*, 23:283–290, 2001.

[83] C. S. Iordache and D. W. Matula. Infinitely precise rounding for division, square root, and square root reciprocal. Dans *Proceedings of the 14th IEEE Symposium on Computer Arithmetic (ARITH'14)*, pages 233–240. IEEE Computer Society Press, 1999.

[84] T. Jebelean. A double-digit Lehmer-Euclid algorithm for finding the GCD of long integers. *Journal of Symbolic Computation*, 19:145–157, 1995.

[85] A. Joux and J. Stern. Lattice reduction : a toolbox for the cryptanalyst. *Journal of Cryptology*, 11(3):161–185, 1998.

[86] M. Kaib and C. P. Schnorr. The generalized Gauss reduction algorithm. *Journal of Algorithms*, 21(3):565–578, 1996.

[87] E. Kaltofen. On the complexity of finding short vectors in integer lattices. Dans *Proceedings of EUROCAL'83*, volume 162 of *Lecture Notes in Computer Science*, pages 236–244. Springer-Verlag, 1983.

[88] R. Kannan. Improved algorithms for integer programming and related lattice problems. Dans *Proceedings of the 15th Symposium on the Theory of Computing (STOC 1983)*, pages 99–108. ACM Press, 1983.

[89] R. Kannan, A. K. Lenstra, and L. Lovász. Polynomial factorization and nonrandomness of bits of algebraic and some transcendental numbers. Dans *Proceedings of the 16th Symposium on the Theory of Computing (STOC 1984)*, pages 191–200. ACM Press, 1984.

[90] S. Khot. Hardness of approximating the shortest vector problem in lattices. Dans *Proceedings of the 2004 Symposium on Foundations of Computer Science (FOCS 2004)*, pages 126–135. IEEE Computer Society Press, 2004.

[91] D. Knuth. The analysis of algorithms. Dans *Actes du Congrès International des Mathématiciens de 1970*, volume 3, pages 269–274. Gauthiers-Villars, 1971.

[92] D. Knuth. *The Art of Computer Programming, vol. 2.* Addison-Wesley, 1981.

[93] A. Korkine and G. Zolotarev. Sur les formes quadratiques. *Mathematische Annalen*, 6:336–389, 1873.

[94] H. Koy and C. P. Schnorr. Segment LLL-reduction of lattice bases. Dans *Proceedings of the 2001 Cryptography and Lattices Conference (CALC'01)*, volume 2146 of *Lecture Notes in Computer Science*, pages 67–80. Springer-Verlag, 2001.

[95] H. Koy and C. P. Schnorr. Segment LLL-reduction of lattice bases with floating-point orthogonalization. Dans *Proceedings of the 2001 Cryptography and Lattices Conference (CALC'01)*, volume 2146 of *Lecture Notes in Computer Science*, pages 81–96. Springer-Verlag, 2001.

[96] J. C. Lagarias. Worst-case complexity bounds for algorithms in the theory of integral quadratic forms. *Journal of Algorithms*, 1:142–186, 1980.

[97] J. C. Lagarias. The computational complexity of simultaneous diophantine approximation problems. Dans *Proceedings of the 1983 Symposium on the Foundations of Computer Science (FOCS 1983)*, pages 32–39. IEEE Computer Society Press, 1983.

[98] J. C. Lagarias. Knapsack public key cryptosystems and Diophantine approximation. Dans *Proceedings of Crypto 1983*, pages 3–23. Plenum Press, 1984.

[99] J. C. Lagarias, W. H. Lenstra, and C. P. Schnorr. Korkine-Zolotarev bases and successive minimal of a lattice and its reciprocal lattice. *Combinatorica*, 10:333–348, 1990.

[100] J. C. Lagarias and A. M. Odlyzko. Solving low-density subset sum problems. Dans *Proceedings of the 1983 Symposium on the Foundations of Computer Science (FOCS 1983)*, pages 1–10. IEEE Computer Society Press, 1983.

[101] J. C. Lagarias and A. M. Odlyzko. Solving low-density subset sum problems. *Journal of the ACM*, 32:229–246, 1985.

[102] J. L. Lagrange. Recherches d'arithmétique. *Nouveaux Mémoires de l'Académie de Berlin*, 1773.

[103] T. Lang and J.-M. Muller. Bounds on runs of zeros and ones for algebraic functions. Dans *Proceedings of the 15th IEEE Symposium on Computer Arithmetic (ARITH'15)*, pages 13–20. IEEE Computer Society Press, 2001.

[104] C. L. Lawson and R. J. Hanson. *Solving Least Squares Problems*. SIAM Publications, 1995.

[105] V. Lefèvre. An algorithm that computes a lower bound on the distance between a segment and $\mathbb{Z}^2$. Rapport de recherche RR1997-18 du Laboratoire de l'Informatique du Parallélisme, 1997.

[106] V. Lefèvre. An algorithm that computes a lower bound on the distance between a segment and $\mathbb{Z}^2$. *Developments in Reliable Computing*, pages 203–212, 1999.

[107] V. Lefèvre. *Moyens arithmétiques pour un calcul fiable*. PhD thesis, École Normale Supérieure de Lyon, 2000.

[108] V. Lefèvre. New results on the distance between a segment and $\mathbb{Z}^2$. Application to the exact rounding. Dans *Proceedings of the 17th IEEE Symposium on Computer Arithmetic (ARITH'17)*, pages 68–75. IEEE Computer Society Press, 2005.

[109] V. Lefèvre and J.-M. Muller. Worst cases for correct rounding of the elementary functions in double precision. Dans *Proceedings of the 15th IEEE Symposium on Computer Arithmetic (ARITH'15)*, pages 111–118. IEEE Computer Society Press, 2001.

[110] V. Lefèvre and J.-M. Muller. Worst cases for correct rounding in double precision. Disponible à l'url `http://perso.ens-lyon.fr/jean-michel.muller/TMDworstcases.pdf`, 2004.

[111] D. H. Lehmer. Euclid's algorithm for large numbers. *American Mathematical Monthly*, 45:227–233, 1938.

[112] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:513–534, 1982.

[113] H. W. Lenstra, Jr. Integer programming with a fixed number of variables. Technical report 81-03, Mathematisch Instituut, Universiteit van Amsterdam, 1981.

[114] H. W. Lenstra, Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.

[115] H. W. Lenstra, Jr. Flags and lattice basis reduction. Dans *Proceedings of the third European congress of mathematics, volume 1*. Birkhäuser, 2001.

[116] D. Lichtblau. Half-GCD and fast rational recovery. Dans *Proceedings of the 2005 International Symposium on Symbolic and Algebraic Computation (ISSAC'05)*. ACM Press, 2005.

[117] J. E. Littlewood. *A Mathematician's Miscellany*. Methuen & Co. Ltd. London, 1953.

[118] L. Lovász. *An Algorithmic Theory of Numbers, Graphs and Convexity*. SIAM Publications, 1986. CBMS-NSF Regional Conference Series in Applied Mathematics.

[119] Magma. The Magma computational algebra system for algebra, number theory and geometry. Disponible à l'url `http://www.maths.usyd.edu.au:8000/u/magma/`.

[120] Maplesoft. Maple 10. Disponible à l'url `http://www.maplesoft.com/`.

[121] P. Markstein. A fast-start method for computing the inverse tangent. Dans *Proceedings of the 17th IEEE Symposium on Computer Arithmetic (ARITH'17)*, pages 266–271. IEEE Computer Society Press, 2005.

[122] J. Martinet. *Perfect Lattices in Euclidean Spaces*. Springer-Verlag, 2002.

[123] A. May. Cryptanalysis of unbalanced RSA with small CRT-exponent. Dans *Proceedings of Crypto 2002*, volume 2448 of *Lecture Notes in Computer Science*, pages 242–256. Springer-Verlag, 2002.

[124] A. May. *New RSA Vulnerabilities Using Lattice Reduction Methods*. PhD thesis, University of Paderborn, 2003.

[125] A. May. Computing the RSA secret key is determinisitic polynomial time equivalent to factoring. Dans *Proceedings of Crypto 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 213–219. Springer-Verlag, 2004.

[126] R. Merkle and M. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory*, 24(5):525–530, 1978.

[127] D. Micciancio. The hardness of the closest vector problem with preprocessing. *IEEE Transactions on Information Theory*, 47(3):1212–1215, 2001.

[128] D. Micciancio. Improving lattice-based cryptosystems using the Hermite normal form. Dans *Proceedings of the 2001 Cryptography and Lattices Conference (CALC'01)*, volume 2146 of *Lecture Notes in Computer Science*, pages 126–145. Springer-Verlag, 2001.

[129] D. Micciancio. The shortest vector problem is NP-hard to approximate to within some constant. *SIAM Journal on Computing*, 30(6):2008–2035, 2001.

[130] D. Micciancio and S. Goldwasser. *Complexity of lattice problems : a cryptographic perspective*. Kluwer Academic Press, 2002.

[131] H. Minkowski. *Geometrie der Zahlen*. Teubner-Verlag, 1896.

[132] N. Möller. On Schönhage's algorithm, and subquadratic integer gcd computation. *Soumis*, 2005.

[133] P. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.

[134] J.-M. Muller. *Elementary Functions, Algorithms and Implementation*. Birkhäuser, 1997.

[135] Yu. V. Nesterenko and M. Waldschmidt. On the approximation of the values of exponential function and logarithm by algebraic numbers. *Matematicheskie Zapiski*, 2:23–42, 1996. Disponible à l'url `http://arxiv.org/abs/math.NT/0002047`.

[136] P. Nguyễn. Cryptanalysis of the Goldreich-Goldwasser-Halevi cryptosystem from Crypto'97. Dans *Proceedings of Crypto 1999*, volume 1666 of *Lecture Notes in Computer Science*, pages 288–304. Springer-Verlag, 1999.

[137] P. Nguyễn. Can we trust cryptographic software? Cryptographic flaws in GNU privacy guard v1.2.3. Dans *Proceedings of Eurocrypt 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 555–570. Springer-Verlag, 2004.

[138] P. Nguyễn and D. Stehlé. Low-dimensional lattice basis reduction revisited (extended abstract). Dans *Proceedings of the 6th Algorithmic Number Theory Symposium (ANTS VI)*, volume 3076 of *Lecture Notes in Computer Science*, pages 338–357. Springer-Verlag, 2004.

[139] P. Nguyễn and D. Stehlé. Floating-point LLL revisited. Dans *Proceedings of Eurocrypt 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 215–233. Springer-Verlag, 2005.

[140] P. Nguyễn and J. Stern. Cryptanalysis of the Ajtai-Dwork cryptosystem. Dans *Proceedings of Crypto 1998*, volume 1462 of *Lecture Notes in Computer Science*, pages 223–242. Springer-Verlag, 1998.

[141] P. Nguyễn and J. Stern. The two faces of lattices in cryptology. Dans *Proceedings of the 2001 Cryptography and Lattices Conference (CALC'01)*, volume 2146 of *Lecture Notes in Computer Science*, pages 146–180. Springer-Verlag, 2001.

[142] A. M. Odlyzko. The rise and fall of knapsack cryptosystems. Dans *Proceedings of Cryptology and Computational Number Theory*, volume 42 of *Proceedings of Symposia in Applied Mathematics*, pages 75–88. American Mathematical Society, 1989.

[143] A. M. Odlyzko and H. te Riele. Dispoof of Mertens conjecture. *Journal für die reine und angewandte Mathematik*, 357:138–160, 1985.

[144] V. Y. Pan and X. Wang. Acceleration of euclidean algorithm and extension. Dans *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation (ISSAC'02)*, pages 207–213. ACM Press, 2002.

[145] M. Payne and R. Hanek. Radian reduction for trigonometric functions. *SIGNUM Newsletter*, 18:19–24, 1983.

[146] M. Pohst. A modification of the LLL reduction algorithm. *Journal of Symbolic Computation*, 4(1):123–127, 1987.

[147] The SPACES Project. MPFR, a LGPL-library for multiple-precision floating-point computations with exact rounding. Disponible à l'url `http://www.mpfr.org/`.

[148] G. B. Purdy. A carry-free algorithm for finding the greatest common divisor of two integers. *Computers and Mathematics with Applications*, 9(2):311–316, 1982.

[149] S. S. Ryskov. On Hermite, Minkowski and Venkov reduction of positive quadratic forms in n variables. *Soviet Mathematics Doklady*, 13:1676–1679, 1972.

[150] C. P. Schnorr. A hierarchy of polynomial lattice basis reduction algorithms. *Theoretical Computer Science*, 53:201–224, 1987.

[151] C. P. Schnorr. A more efficient algorithm for lattice basis reduction. *Journal of Algorithms*, 9(1):47–62, 1988.

[152] C. P. Schnorr. Lattice reduction by random sampling and birthday methods. Dans *Proceedings of the annual symposium on theoretical aspects of computer science (STACS 2003)*, volume 2607 of *Lecture Notes in Computer Science*, pages 145–156. Springer-Verlag, 2003.

[153] C. P. Schnorr. Fast LLL-type lattice reduction. *À paraître*, 2005. Version préliminaire disponible à l'url `http://www.mi.informatik.uni-frankfurt.de/research/papers.html`.

[154] C. P. Schnorr and M. Euchner. Lattice basis reduction : improved practical algorithms and solving subset sum problems. *Mathematics of Programming*, 66:181–199, 1994.

[155] A. Schönhage. Schnelle Berechnung von Kettenbruchentwicklungen. *Acta Informatica*, 1:139–144, 1971.

[156] A. Schönhage. Factorization of univariate integer polynomials by Diophantine approximation and improved basis reduction algorithm. Dans *Proceedings of the 1984 International Colloquium on Automata, Languages and Programming (ICALP 1984)*, volume 172 of *Lecture Notes in Computer Science*, pages 436–447. Springer-Verlag, 1984.

[157] A. Schönhage. Fast reduction and composition of binary quadratic forms. Dans *Proceedings of the 1991 International Symposium on Symbolic and Algebraic Computation (ISSAC'91)*, pages 128–133. ACM Press, 1991.

[158] A. Schönhage and V. Strassen. Schnelle Multiplikation grosser Zahlen. *Computing*, 7:281–292, 1971.

[159] I. Semaev. A 3-dimensional lattice reduction algorithm. Dans *Proceedings of the 2001 Cryptography and Lattices Conference (CALC'01)*, volume 2146 of *Lecture Notes in Computer Science*, pages 181–193. Springer-Verlag, 2001.

[160] A. Shamir. A polynomial time algorithm for breaking the basic Merkle-Hellman cryptosystem. Dans *Proceedings of the 1982 Symposium on the Foundations of Computer Science (FOCS 1982)*, pages 145–152. IEEE Computer Society Press, 1982.

[161] M. Shand and Vuillemin J. Fast implementations of RSA cryptography. Dans *Proceedings of the 11th IEEE Symposium on Computer Arithmetic (ARITH'11)*, pages 252–259. IEEE Computer Society Press, 1993.

[162] V. Shoup. NTL, Number Theory C++ Library. Disponible à l'url `http://www.shoup.net/ntl/`.

[163] I. Shparlinski. *Cryptographic Applications of Analytic Number Theory*. Birkhäuser, 2003.

[164] C. L. Siegel. *Lectures on the Geometry of Numbers*. Springer-Verlag, 1989.

[165] D. Simon. Solving quadratic equations using reduced unimodular quadratic forms. *Mathematics of Computation*, 74(251):1531–1543, 2005.

[166] D. Stehlé. Breaking Littlewood's cipher. *Cryptologia*, XXVIII(4):341–357, 2004.

[167] D. Stehlé, V. Lefèvre, and P. Zimmermann. Worst cases and lattice reduction. Dans *Proceedings of the 16th Symposium on Computer Arithmetic (ARITH'16)*, pages 142–147. IEEE Computer Society Press, 2003.

[168] D. Stehlé, V. Lefèvre, and P. Zimmermann. Searching worst cases of a one-variable function. *IEEE Transactions on Computers*, 54(3):340–346, 2005.

[169] D. Stehlé and P. Zimmermann. A binary recursive gcd algorithm. Dans *Proceedings of the 6th Algorithmic Number Theory Symposium (ANTS VI)*, volume 3076 of *Lecture Notes in Computer Science*, pages 411–425. Springer-Verlag, 2004.

[170] D. Stehlé and P. Zimmermann. Gal's accurate tables method revisited. Dans *Proceedings of the 17th Symposium on Computer Arithmetic (ARITH'17)*, pages 257–264. IEEE Computer Society Press, 2005.

[171] J. Stein. Computational problems associated to Racah algebra. *Journal of Computational Physics*, 1(3):397–405, 1967.

[172] P. H. Sterbenz. *Floating Point Computation*. Prentice-Hall, 1974.

[173] M. I. Stogrin. Regular Dirichlet-Voronoï partitions for the second triclinic group. *American Mathematical Society*, 1977. Traduction en Anglais des actes de l'institut Steklov de mathématiques, numéro 123 (1973).

[174] A. Storjohann. Faster algorithms for integer lattice basis reduction. Technical report, ETH Zürich, 1996.

[175] V. Strassen. Gaussian elimination is not optimal. *Numerical Mathematics*, 13:354–356, 1969.

[176] P. P. Tammela. On the reduction theory of positive quadratic forms. *Soviet Mathematics Doklady*, 14:651–655, 1973.

[177] The ARENAIRE Project. CR-Libm, a library of correctly rounded elementary functions in double-precision. Disponible à l'url `http://lipforge.ens-lyon.fr/www/crlibm/`.

[178] K. Thull and C. K. Yap. A unified approach to HGCD algorithms for polynomials and integers. Disponible à l'url `http://cs.nyu.edu/cs/faculty/yap/papers`, 1990.

[179] A. L. Toom. The complexity of a scheme of functional elements realizing the multiplication of integers. *Soviet Mathematics Doklady*, 3:714–716, 1963.

[180] B. Vallée. *Une Approche Géométrique de la Réduction des Réseaux en Petite Dimension*. PhD thesis, Université de Caen, 1986.

[181] B. Vallée. Gauss' algorithm revisited. *Journal of Algorithms*, 12:556–572, 1991.

[182] B. Vallée. Dynamical analysis of a class of Euclidean algorithms. *Theoretical Computer Science*, 297(1-3):447–486, 2003.

[183] G. Voronoï. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. *Journal für die reine und angewandte Mathematik*, 134:198–287, 1908.

[184] B. L. van der Waerden. Die Reduktionstheorie der positiven quadratischen Formen. *Acta Mathematica*, 96:265–309, 1956.

[185] A. Weilert. Asymptotically fast GCD computation in $\mathbb{Z}[i]$. Dans *Proceedings of the 4th Algorithmic Number Theory Symposium (ANTS IV)*, volume 1838 of *Lecture Notes in Computer Science*, pages 595–613. Springer-Verlag, 2000.

[186] J. H. Wilkinson. *The Algebraic Eigenvalue Problem*. Oxford University Press, 1988.

[187] D. Wilson. Littlewood's cipher. *Cryptologia*, 3(2):120–121, 1979.

[188] D. Wilson. Littlewood's cipher. *Cryptologia*, 3(3):172–176, 1979.

[189] C. K. Yap. Fast unimodular reduction : planar integer lattices. Dans *Proceedings of the 1992 Symposium on the Foundations of Computer Science (FOCS 1992)*, pages 437–446. IEEE Computer Society Press, 1992.

[190] C. K. Yap. *Fundamental Problems in Algorithmic Algebra*. Oxford University Press, 2000.

[191] A. Ziv. Fast evaluation of elementary mathematical functions with correctly rounded last bit. *ACM Transactions on Mathematical Software*, 17(3):410–423, 1991.

Résumé

Les réseaux euclidiens sont un outil particulièrement puissant dans plusieurs domaines de l'algorithmique, en cryptographie et en théorie algorithmique des nombres par exemple. L'objet du présent mémoire est dual : nous améliorons les algorithmes de réduction des réseaux, et nous développons une nouvelle application dans le domaine de l'arithmétique des ordinateurs. En ce qui concerne l'aspect algorithmique, nous nous intéressons aux cas des petites dimensions (en dimension un, où il s'agit du calcul de pgcd, et aussi en dimensions 2 à 4), ainsi qu'à la description d'une nouvelle variante de l'algorithme LLL, en dimension quelconque. Du point de vue de l'application, nous utilisons la méthode de Coppersmith permettant de trouver les petites racines de polynômes modulaires multivariés, pour calculer les pires cas pour l'arrondi des fonctions mathématiques, quand la fonction, le mode d'arrondi, et la précision sont donnés. Nous adaptons aussi notre technique aux mauvais cas simultanés pour deux fonctions. Ces deux méthodes sont des pré-calculs coûteux, qui une fois effectués permettent d'accélérer les implantations des fonctions mathématiques élémentaires en précision fixée, par exemple en double précision.

La plupart des algorithmes décrits dans ce mémoire ont été validés expérimentalement par des implantations, qui sont disponibles à l'url `http://www.loria.fr/~stehle`.

Mots-clés: Réseaux euclidiens, algorithme LLL, calcul de pgcd, arithmétique flottante, implantation de fonctions mathématiques élémentaires, dilemme du fabricant de tables, méthode de Coppersmith, analyse de complexité.

Abstract

Euclidean lattices are a particularly powerful tool for several algorithmic topics, among which are cryptography and algorithmic number theory. The contributions of this thesis are twofold : we improve lattice basis reduction algorithms, and we introduce a new application of lattice reduction, in computer arithmetic. Concerning lattices, we consider both small dimensions (in dimension one, where the problem degenerates to a gcd calculation, and in dimensions 2 to 4), and arbitrary dimensions, for which we improve the classical LLL algorithm. Concerning the application, we make use of Coppersmith's method for computing the small roots of multivariate modular polynomials, in order to find the worst cases for the rounding of mathematical functions, when the function, the rounding mode and the precision are fixed. We also generalise our technique to find input numbers that are simultaneously bad for two functions. These two methods are expensive pre-computations, but once performed, they help speeding up the implementations of elementary mathematical functions in fixed precision, for example in double precision.

Most of the algorithms described in this thesis have been validated experimentally. These implementations are available at the url `http://www.loria.fr/~stehle`.

Keywords: Euclidean lattices, LLL algorithm, gcd computation, floating-point arithmetic, elementary functions implementation, tables' maker dilemma, Coppersmith technique, complexity analysis.