La Réduction de Réseaux en Cryptographie

Antoine Joux

30 avril 1993

Remerciements

Je tiens, tout d'abord, à remercier J. Stern pour avoir encadré mon travail depuis le D.E.A. et m'avoir dirigé vers ce sujet de thèse. Grâce à lui, j'ai pu assister à diverses conférences, qui m'ont permis d'imaginer de nouvelles orientations pour mes travaux, et de rencontrer les personnalités qui dominent la recherche mondiale en cryptographie. Il m'a énormément appris, non seulement dans le domaine purement scientifique, mais aussi dans les aspects quotidiens du métier de chercheur. Enfin, je le remercie pour son amitié, et pour l'ambiance de travail agréable qu'il a su insuffler au GRECC.

Je remercie A.M. Odlyzko pour ses précieux conseils sur les problèmes de sacs-à-dos, et pour avoir accepté d'être rapporteur de ma thèse.

C.-P. Schnorr a beaucoup apporté à cette thèse, à la fois par l'intermédiaire de ses nombreux articles traitant de la réduction de réseau, et plus directement, en m'accueillant quelques semaines dans son équipe de Francfort. Je le remercie d'avoir accepté d'être rapporteur, et de participer à mon jury de thèse.

Merci aussi à J.-M. Couveignes, qui m'a donné l'occasion d'appliquer LLL aux dessins d'enfants.

Je remercie, encore, tous les membres de mon jury de thèse, P. Camion, P. Flajolet, M. Minoux, C.-P. Schnorr, J.-M. Steyaert et B. Vallée, qui ont tous, à divers titres contribué à l'élaboration de cette thèse.

Je suis, également, très reconnaisant à la Délégation Générale pour l'Armement, qui a contribué, à double titre, à l'élaboration de cette thèse, en m'accueillant dans l'Option Recherche, sous la direction de l'ICA Quenzer, ainsi que par l'intermédiaire de contrats avec le GRECC. Je remercie mon parrain scientifique, le professeur Minoux, et mon parrain DGA, l'IPA Wolf, pour l'aide qu'ils m'ont apportée.

Enfin, je remercie mon épouse, Katia, pour son soutien quotidien, et pour son aide inestimable.

Première partie

Résultats théoriques

Chapitre 1

Introduction à la réduction de réseau

1.1 Historique

La théorie de la réduction des réseaux a une origine assez ancienne. Elle fut développée en terme de réduction des formes quadratiques, par de grands mathématiciens comme Lagrange [22], Hermite [16], Korkine et Zolotarev [20] qui ont cherché à définir la notion de forme réduite associée à une forme quadratique. Le cas particulier des formes quadratiques à deux variables a été résolu par Gauss [13], qui a donné un algorithme capable d'en calculer une forme réduite. Malheureusement, ces travaux ne purent apporter une réponse définitive au problème dans le cas général, car lorsque le nombre de variables augmente, il est possible de définir la notion de réduction de diverses façons, sans qu'aucune des définitions n'apparaisse naturellement comme la meilleure. Au début du XX^{e} siècle, Minkowski introduit la géométrie des nombres [27], domaine dans lequel il étudie les intersections entre un réseau et une région convexe de l'espace à n dimensions. Dans ses travaux, il définit la notion de minima successifs d'un réseau et celle de famille de vecteurs réalisant ces minima. Cependant, cela ne permet pas de résoudre le problème de réduction de réseau, car une telle famille de vecteurs n'est pas nécessairement une base.

La théorie de la réduction réapparaît en 1982, après la publication par Lenstra, Lentra et Lovász [24] d'un algorithme capable en temps polynomial, de transformer une base quelconque d'un réseau, en une base réduite au sens de Lovász. Cette notion de réduction, bien que faible, suffit à garantir, par des majorations de la taille des vecteurs obtenus, la qualité de la base réduite. Cette base n'est, bien sûr, pas parfaite, mais elle suffit à résoudre de nombreux problèmes, comme par exemple, la factorisation de polynômes. Après cette première publication, de nombreux résultats font leur apparition. Par exemple, Kannan [19] donne un algorithme (non polynomial) permettant de trouver le vecteur le plus court d'un réseau, à partir d'une base réduite au sens de Lovász, Babai étend cet algorithme à la recherche du point d'un réseau le plus proche d'un point quelconque donné. De son côté, Schnorr propose de nombreuses améliorations de LLL: du point de vue théorique, il montre comment faire plus rapide [34], en remplaçant les rationnels par des flottants dans LLL, et comment faire mieux [33], en définissant une hiérarchie d'algorithmes tous polynomiaux, mais de plus en plus lents,

qui permettent d'approcher le vecteur le plus court d'un réseau, mieux que LLL ne le fait. Enfin, conjoitement avec Euchner [35], il montre comment faire en pratique, pour réduire des réseaux, le plus rapidement possible. En même temps, les applications possibles des algorithmes de réductions de réseaux se sont diversifiées.

Cependant, de nombreuses questions restent encore sans réponse. Par exemple, le problème de la recherche du vecteur le plus court est-il NP-dur; est-il possible de trouver un vecteur dont la longueur soit bornée dans un rapport polynomial du vecteur le plus court, et cela, en temps polynomial?

1.2 Quelques Définitions

Définition 1.1 *Un réseau est un sous-groupe discret de $\mathbf{R}^n$, c'est-à-dire un ensemble de vecteurs vérifiant:*

- *Le vecteur nul est dans le réseau*
- *L'opposé de tout vecteur du réseau est dans le réseau*
- *La somme de deux vecteurs du réseau est encore dans le réseau*
- *Tous les points du réseau sont isolés*

Définition 1.2 *Une base d'un réseau est un ensemble de vecteurs linéairement indépendants qui engendrent le réseau, c'est-à-dire que tout vecteur du réseau est une combinaison linéaire à coefficients entiers des vecteurs de la base, et réciproquement, toute combinaison linéaire à coefficients entiers des vecteurs de la base est dans le réseau.*

En fait, nous nous intéresserons essentiellement aux réseaux entiers, c'est-à-dire aux réseaux formés uniquement de vecteurs à coordonnées entières, en d'autres termes:

Définition 1.3 *Un réseau entier est un sous-groupe de $\mathbf{Z}^n$, c'est-à-dire un ensemble de vecteurs à coordonnées entières vérifiant:*

- *Le vecteur nul est dans le réseau*
- *L'opposé de tout vecteur du réseau est dans le réseau*
- *La somme de deux vecteurs du réseau est encore dans le réseau*

Nous verrons par la suite que de nombreuses définitions de réduction sont possibles, mais informellement, une base réduite d'un réseau est une base du réseau formée de vecteurs courts et presque orthogonaux entre eux. Par chance, en dimension 2, tout se simplifie, et il devient possible de définir précisement la notion de réduction. Cette notion de réduction peut s'exprimer en terme de minima successifs du réseau.

Définition 1.4 *On appelle premier minimum d'un réseau la longueur du plus court vecteur non nul. Le $i^{ème}$ minimum d'un réseau L est le plus petit réel positif λ, pour lequel il existe dans le réseau i vecteurs linéairement indépendants $\vec{v}_j$ vérifiant tous $\|\vec{v}_j\|^2 \leq \lambda$. On note ce réel $\Lambda_i(L)$.*

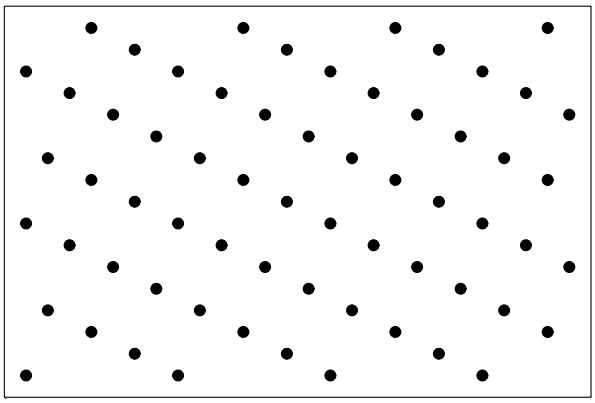


FIG. 1.1 - Un exemple de Réseau

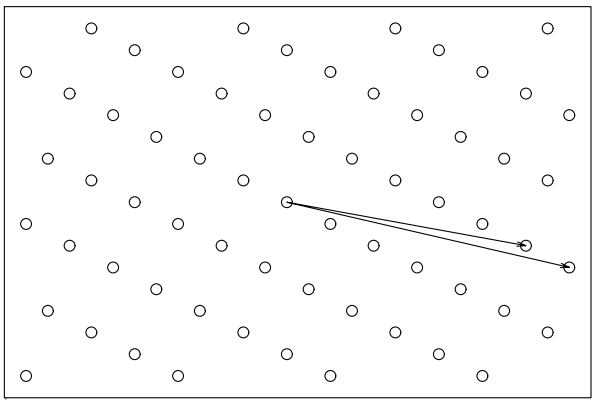


FIG. 1.2 - Une base quelconque

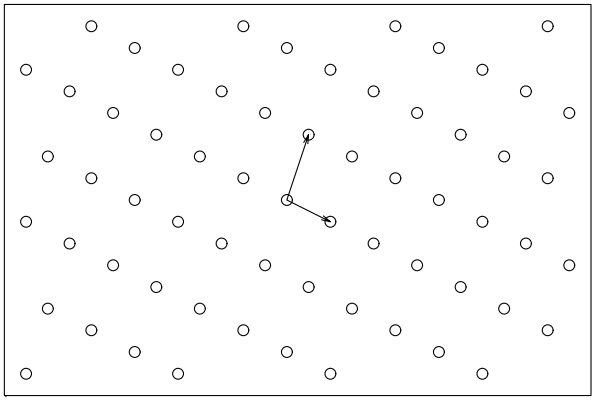


FIG. 1.3 - Une base réduite

En dimension 2, on dit qu'une base $(\vec{u}, \vec{v})$ d'un réseau L est réduite, si elle réalise les minima $\Lambda_1(L)$ et $\Lambda_2(L)$. Dans ce cas, une base réduite est donc une base formée de vecteurs aussi courts que possible. De plus, en dimension 2, il est possible de dessiner des réseaux, et d'illustrer ainsi les diverses notions que nous venons de définir, voir les figures 1.1, 1.2 et 1.3.

Il serait naturel d'étendre en dimension plus grande cette notion de réduction, et de dire qu'une base B d'un réseau L est réduite, si et seulement si, elle réalise les minima successifs du réseau. Cependant, en grande dimension (≥ 5), cette approche ne fonctionne plus, car une famille de vecteurs linéairement indépendants réalisant les minima successifs, n'est plus nécéssairement une base du réseau. Considérons, par exemple, le réseau suivant, en dimension 5:

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

Base du réseau (en colonne)

Ce réseau, contient le vecteur suivant:

$$V = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Or, il est facile de constater que les minima successifs sont:

$$\begin{aligned} \Lambda_1 &= 2 \\ \Lambda_2 &= 2 \\ \Lambda_3 &= 2 \\ \Lambda_4 &= 2 \\ \Lambda_5 &= 2 \end{aligned}$$

et qu'une famille réalisant ces minima, ne contient pas forcément V, comme par exemple:

$$\begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

Famille réalisant les minima du réseau

1.3 L'algorithme de Gauss

Dans cette section, nous allons détailler l'algorithme de Gauss, et sa variante t-Gauss. Ensuite, nous présenterons quelques propriétés connues de cet algorithme.

Voici une brève description de l'algorithme de Gauss:

Algorithme de Gauss
Entrée: $(\vec{u}, \vec{v})$ base du réseau
Sortie: $(\vec{u}, \vec{v})$ base réduite
Si $\|\vec{u}\| < \|\vec{v}\|$ alors échanger $\vec{u}$ et $\vec{v}$
Faire:
$m \leftarrow \left\lfloor \frac{\vec{u}.\vec{v}}{\|\vec{v}\|^2} \right\rceil$
$\vec{u} \leftarrow \vec{u} - m\vec{v}$
échanger $\vec{u}$ et $\vec{v}$
jusqu'à ce que $\|\vec{u}\| \leq \|\vec{v}\|$

Le principe de cet algorithme consiste, à chaque étape, à diminuer, autant que possible, la longueur du vecteur le plus long de la base $(\vec{u}, \vec{v})$, par translation parallèlement à l'autre vecteur. L'algorithme s'arrête, lorsque le vecteur ainsi obtenu, est plus long que le vecteur resté fixe, en effet, dans ce cas, l'opération de translation ne peut plus rien apporter. Pour illustrer le comportement de cet algorithme, un exemple de réduction est présenté dans les figures 1.4, 1.5, 1.6 et 1.7.

Montrons que cet algorithme calcule bien une base $(\vec{u}, \vec{v})$, réalisant les premier et second minima du réseau (définition 1.4). Tout d'abord, l'algorithme produit forcément une base, car à chaque étape, la transformation appliquée est unimodulaire. De plus, il est facile de vérifier, que dans la base obtenue, on a:

$$\|\vec{u}\| \leq \|\vec{v}\| \tag{1.1}$$
$$2|\vec{u}.\vec{v}| \leq \|\vec{u}\|^2 \tag{1.2}$$

Par conséquent, pour tout vecteur du réseau, de la forme $\alpha\vec{u} + \beta\vec{v}$, avec α et β entiers, on a:

$$\|\alpha\vec{u} + \beta\vec{v}\|^2 = \alpha^2\|\vec{u}\|^2 + 2\alpha\beta\vec{u}.\vec{v} + \beta^2\|\vec{v}\|^2 \tag{1.3}$$
$$\geq (\alpha^2 - |\alpha\beta| + \beta^2)\|\vec{u}\|^2 \tag{1.4}$$
$$\geq ((|\alpha| - |\beta|)^2 + |\alpha\beta|)\|\vec{u}\|^2 \tag{1.5}$$
$$\geq \|\vec{u}\|^2. \tag{1.6}$$

Ainsi, $\vec{u}$ est bien le vecteur le plus court du réseau. Si, de plus, on appelle $\vec{v}^*$, le projeté de $\vec{v}$, perpendiculairement, on a $\vec{v}^* = \vec{v} - \mu\vec{u}$, avec $|\mu| \leq 1/2$, donc,

$$\frac{3}{4}\|\vec{v}\|^2 \leq \|\vec{v}^*\|^2 \leq \|\vec{v}\|^2.$$

Déduisons-en que $\vec{v}$, réalise bien le second minimun du réseau, choisissons donc $\alpha\vec{u}+\beta\vec{v}$, linéairement indépendant de $\vec{u}$, i.e, avec $\beta \neq 0$. La norme de ce vecteur est supérieure à $\beta^2\|\vec{v}^*\|^2$, et donc à $\frac{3\beta^2}{4}\|\vec{v}\|^2$. Dès que, $|\beta| > 1$, ce vecteur est plus grand que $\vec{v}$, par conséquent, quitte à changer les signes de α et β, on peut supposer que $\beta = 1$, et dans ce cas, la dernière étape de l'algorithme de Gauss, permet de garantir que $\vec{v}$ est

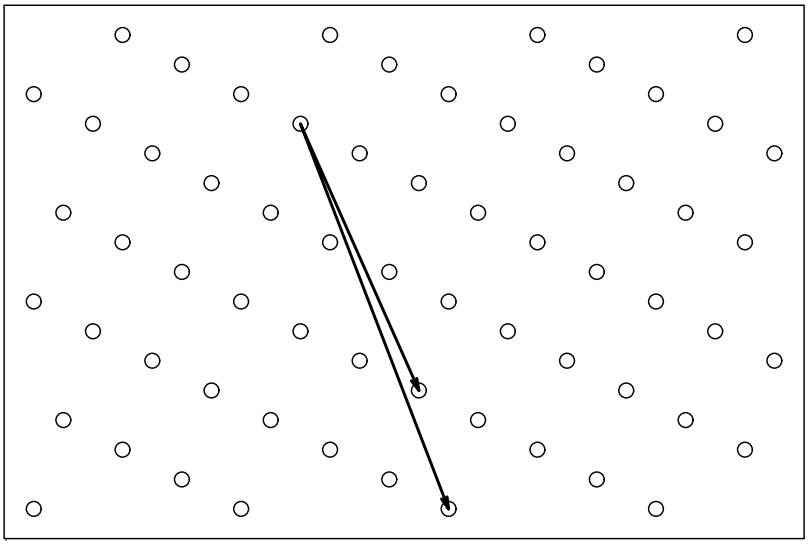


FIG. 1.4 - Base avant la réduction

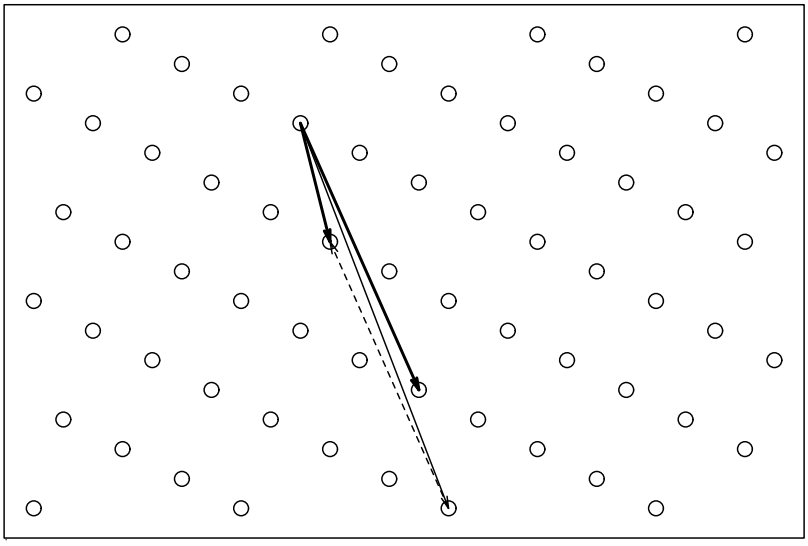


FIG. 1.5 - Première étape de réduction

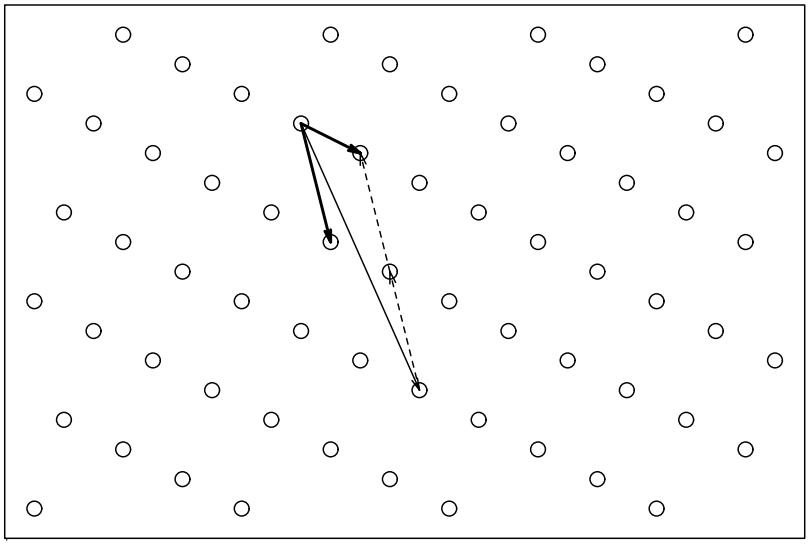


FIG. 1.6 - Deuxième étape de réduction

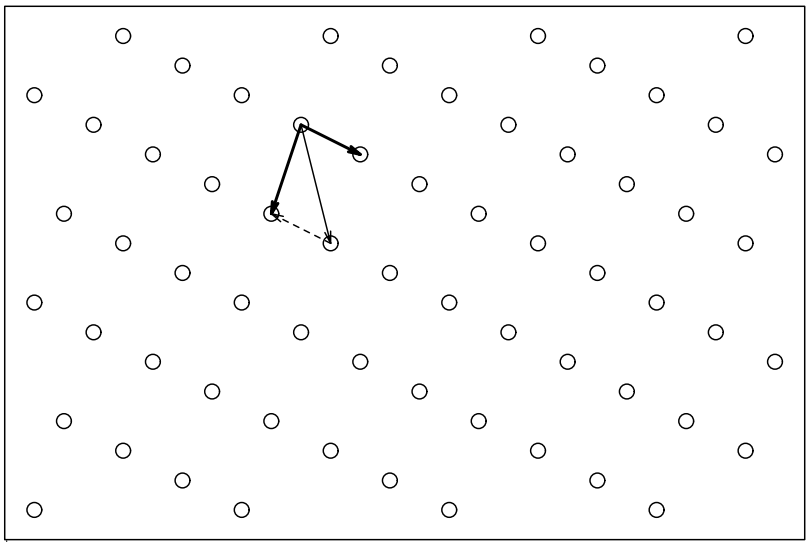


FIG. 1.7 - Dernière étape et base réduite

le vecteur le plus court, parmi ceux de la forme $\vec{v} - m\vec{u}$. Tout ceci prouve bien, que l'algorithme calcule une base du réseau, qui en réalise les minima.

De plus, cet algorithme est très rapide; en effet, son temps d'exécution dans le cas le pire est linéaire en la taille des nombres apparaissant dans la base à réduire. Pour le voir, nous allons tout d'abord introduire une variante de l'algorithme de Gauss, paramétrée par une constante $t \geq 1$.

Algorithme de t-Gauss
Entrée: $(\vec{u}, \vec{v})$ base du réseau
Sortie: $(\vec{u}, \vec{v})$ base réduite
Si $\|\vec{u}\| < \|\vec{v}\|$ alors échanger $\vec{u}$ et $\vec{v}$
Faire:
$m \leftarrow \left\lfloor \frac{\vec{u}.\vec{v}}{\|\vec{v}\|^2} \right\rceil$
$\vec{u} \leftarrow \vec{u} - m\vec{v}$
échanger $\vec{u}$ et $\vec{v}$
jusqu'à ce que $\|\vec{u}\| \leq t\|\vec{v}\|$

La seule différence entre cet algorithme et l'algorithme d'origine, se situe dans la condition d'arrêt, en effet, on n'impose plus que le plus long des deux vecteurs ne puisse pas être raccourci, mais juste, qu'il ne puisse pas être raccourci d'un facteur plus grand que t. Ainsi, la base obtenue, n'est qu'une approximation d'une base réduite. En revanche, il maintenant évident, que l'algorithme, appliqué aux réseaux entiers, est polynomial, en la taille du plus long vecteur de la base initiale. En effet, à chaque étape, sauf, peut-être, à la dernière, la norme du vecteur le plus long décroît d'un facteur t au moins. Ce vecteur est toujours à coordonnées entières, donc la norme du vecteur le plus grand reste toujours supérieure à 1. Donc, si M désigne la norme du vecteur le plus long dans la base initiale, le nombre d'itérations $k(t)$ de t-Gauss est inférieure à $1 + \log_t M$. De plus, le cas de l'algorithme de Gauss, i.e, le cas $t = 1$, se traite en remarquant que, si $t \leq \sqrt{3}$ alors:

$$k(t) \leq k(1) \leq k(t) + 1.$$

Sans perte de généralité, on peut supposer que quand l'algorithme de t-Gauss s'arrête, on a $\vec{u}.\vec{v} \geq 0$. Choisissons maintenant, λ et α, tels que:

$$\|\vec{u}\| = \lambda\|\vec{v}\| \tag{1.7}$$
$$\vec{u}.\vec{v} = \alpha\|\vec{v}\|^2 \tag{1.8}$$
$$\alpha \geq 0 \tag{1.9}$$

Trois cas sont alors possibles:

1. Si $\lambda \leq 1$, même 1-Gauss s'arrête là.
2. Si $\lambda > 1$, et $\alpha \leq 1/2$, 1-Gauss fait un dernier tour qui se contente d'échanger $\vec{u}$ et $\vec{v}$.
3. Si $\lambda > 1$, et $\alpha > 1/2$, on a, plus précisement:

$$1 < \lambda \leq t \leq \sqrt{3} \tag{1.10}$$
$$1/2 < \alpha \leq \lambda^2/2 \leq 3/2 \tag{1.11}$$

Dans ce cas, l'étape suivante de 1-Gauss, remplace $\vec{u}$ par $\vec{u}-\vec{v}$. Comme, de plus:

$$\begin{aligned} \|\vec{u}-\vec{v}\|^2 &= \|\vec{u}\| - 2\vec{u}.\vec{v} + \|\vec{v}\|^2 && (1.12)\\ &= (\lambda^2 + 1 - 2\alpha)\|\vec{v}\|^2 && (1.13)\\ &\geq \|v\|^2 && (1.14) \end{aligned}$$

cette étape suffit à achever la réduction de Gauss.

L'algorithme de Gauss est donc bien polynomial, bien que cela ne soit pas complétement évident à première vue. Cependant, la borne que nous venons de donner n'est pas la meilleure possible, en effet, B. Vallée montre [39] qu'en fait:

$$k(1) \leq 3 + \log_{1+\sqrt{2}} M,$$

et que, de plus, cette borne est atteinte. Mieux encore, P. Flajolet et B. Vallée ont prouvé récemment [9] que le temps d'exécution en moyenne de cet algorithme est constant.

1.4 L'orthogonalisation de Gram Schmidt

Avant de présenter l'algorithme de Lenstra, Lenstra et Lovász, il nous faut rappeler le procédé d'orthogonalisation de Gram Schmidt. Ce procédé transforme une base $B = (b_1, b_2, \cdots, b_n)$ d'un sous-espace vectoriel de $\mathbf{R}^m$, en une autre base $B^* = (b_1^*, b_2^*, \cdots, b_n^*)$ du même espace vectoriel vérifiant les conditions suivantes:

- $b_1^* = b_1$
- b_i^* est le projeté de b_i orthogonalement au sous-espace vectoriel engendré par les $i-1$ premiers vecteurs de B.

L'algorithme calcule également une matrice M triangulaire inférieure avec une diagonale de 1 telle que $B = B^{*\,t}M$.

Algorithme de Gram-Schmidt
Entrée: B
Sortie: B^* et M
Pour i allant de 1 à n
    $j = 1$
    $b_i^* \leftarrow b_i$
    Tant que $j < i$
        $m_{i,j} = \frac{(b_i|b_j^*)}{\|b_j^*\|^2}$
        $b_i^* \leftarrow b_i^* - m_{i,j} b_j^*$

Pour construire, la base B^*, cet algorithme opère séquentiellement, en commençant par prendre b_1 comme premier vecteur, puis le projeté de b_2 orthogonalement à b_1 comme second vecteur, et ainsi de suite. Le vecteur b_i^* est donc le projeté de b_i orthogonalement à l'espace engendré par $(b_1, \cdots, b_{i-1})$. La figure 1.8 illustre le passage de b_3 à b_3^*.

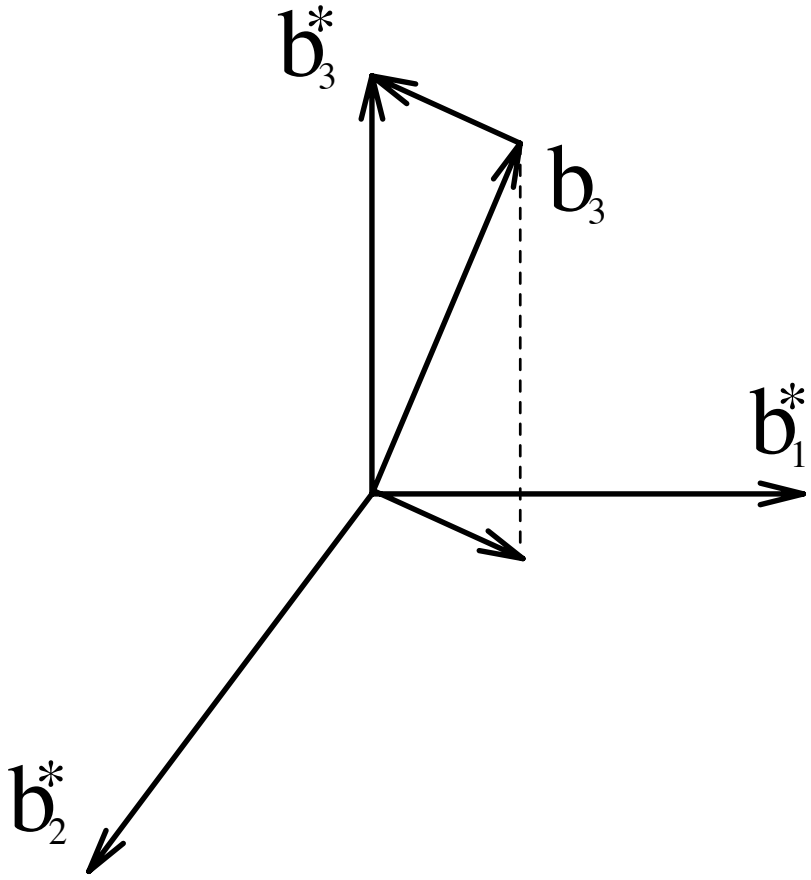


FIG. 1.8 - Passage de b_3 à b_3^*

1.5 L'algorithme LLL

En fait, LLL est une combinaison de Gauss et de Gram-Schmidt, dont le principe consiste à appliquer l'algorithme de Gauss à des sous-réseaux projetés de dimension 2. Plus précisement, on commence par projeter b_i et b_{i+1} orthogonalement à l'espace engendré par $(b_1, \cdots, b_{i-1})$, et on effectue une étape de l'algorithme de Gauss sur le réseau projeté. Bien entendu, les opérations de translation et d'échange dans l'espace projeté sont relevées, et s'appliquent en fait sur les vecteurs b_i et b_{i+1} eux-mêmes. De plus, si nécessaire, le nouveau vecteur obtenu est aussi translaté par rapport à $(b_1, \cdots, b_{i-1})$, pour se rapprocher de son projeté. La figure 1.9 illustre un exemple de translation en dimension 3. Voici maintenant, une description complète de l'algorithme de Lenstra, Lenstra et Lovász d'origine. Dans cet algorithme, les nombres sont représentés de manière exacte, sous forme de rationnels.

LLL
Entrée: B base d'un réseau entier, t un paramètre réel
Sortie: B base réduite du réseau
Calcul de B^* et M par **Gram-Schmidt**
Pour i allant de 1 à n: $L_i \leftarrow \|b_i^*\|^2$
Effacer B^*
$k = 2$
Tant que $k \leq n$
    RED$(k, k-1)$
    Si $L_k \geq (t - m_{k,k-1}^2) L_{k-1}$ alors:
        $l \leftarrow k-2$
        Tant que $l \geq 1$:
            RED(k, l)
            $l \leftarrow l - 1$
        $k \leftarrow k + 1$
    sinon: *(* échange et mise à jour *)*
        $m \leftarrow m_{k,k-1}$
        $L \leftarrow L_k + m^2 L_{k-1}$
        $m_{k,k-1} \leftarrow m L_{k-1}/L$
        $L_k \leftarrow L_{k-1} L_k / L$
        $L_{k-1} \leftarrow L$
        Echanger b_k et b_{k-1}
        Si $k > 2$ alors:
            $i \leftarrow 1$
            Tant que $i \leq k-2$
              Echanger $m_{k,i}$ et $m_{k-1,i}$
              $i \leftarrow i + 1$
        $i \leftarrow k + 1$
        Tant que $i \leq n$:
            $\mu \leftarrow m_{i,k}$
            $m_{i,k} \leftarrow m_{i,k-1} - m\mu$
            $m_{i,k-1} \leftarrow \mu + m_{k,k-1} m_{i,k}$
            $i \leftarrow i + 1$

Si $k > 2$ alors $k \leftarrow k - 1$

Voici le sous-programme **RED** utilisé par LLL:

RED
Entrée: i et j entiers
Si $|m_{i,j}| > 1/2$ alors:
$r \leftarrow \lfloor m_{i,j} \rceil$
$b_i \leftarrow b_i - r b_j$
$m_{i,j} \leftarrow m_{i,j} - r$
$w \leftarrow 1$
Tant que $w \leq l - 1$:
$m_{i,w} \leftarrow m_{i,w} - r m_{j,w}$
$w \leftarrow w + 1$

Essayons de comprendre un peu mieux le principe de cet algorithme, tout d'abord on peut constater que comme dans l'algorithme de Gauss, il y a essentiellement deux types d'opérations, des translations et des échanges de vecteurs. Les translations se limitent à déplacer un vecteur parallèlement à ses prédécesseurs, et les échanges n'agissent que sur des vecteurs voisins. Les translations visent à ramener les coefficients de M à une valeur absolue inférieure à $1/2$, donc à rapprocher b_k de b_k^* pour rendre b_k presque orthogonal aux vecteurs qui le précédent. Chacune de ces translations est réalisée par un appel à **RED**. Appel qui translate un vecteur b_i parallèlement à un b_j qui le précéde, de plus comme une telle translation modifie M, **RED** met à jour cette matrice. En revanche, les b_i^* ne sont pas modifiés par les translations. Les échanges, eux, permettent de diminuer la norme de b_{k-1}^* aux dépens de celle de b_k^*, ils visent donc à déplacer le poids dans B^* des premiers vecteurs vers les derniers, permettant ainsi de raccourcir les premiers vecteurs et d'améliorer la qualité de la base. En fait, afin de réduire la complexité de l'algorithme, B^* n'est pas conservée, car seules les normes de b_i^*, notées L_i dans l'algorithme, et M sont vraiment utiles. Lors des translations et des échanges, ces quantités doivent être mises à jour, d'où toutes les opérations dans l'algorithme sur les $m_{i,j}$ et les L_i. Nous avons déjà vu que **RED** s'occupe de la mise à jour pendant les translations et nous détaillerons un peu plus loin le cas des échanges.

Lorsqu'une base d'un réseau a été réduite par LLL, elle est réduite au sens de Lovász.

Définition 1.5 *On dit qu'une base B d'un réseau entier L est réduite au sens de Lovász lorsque les conditions suivantes portant sur B, B^* et M sont vérifiées:*

- *Conditions de Lovász:*
  1. $\forall i < j : |(b_j | b_i^*)| \leq \frac{\|b_i^*\|^2}{2}$
  2. $\forall i : \|b_i^*\|^2 \leq t \left(\|b_{i+1}^*\|^2 + \frac{(b_{i+1}|b_i^*)^2}{\|b_i^*\|^2} \right)$

Il facile de voir, étant donnée une base $(b_1, \cdots, b_n)$ réduite au sens de Lovász d'un réseau L, que la norme du vecteur non nul le plus court de ce réseau est au moins aussi grande que la norme du plus petit des b_i^*. Or, des conditions de Lovász on tire:

$$t\|b_{i+1}^*\|^2 \geq \|b_i^*\|^2 (1 - \frac{t}{4})$$

d'où:

$$\|b_{i+1}^*\|^2 \geq \|b_i^*\|^2(\frac{1}{t} - \frac{1}{4}).$$

Par conséquent, si l_1 désigne la norme du vecteur le plus court de L, on a:

$$l_1 \geq (\frac{1}{t} - \frac{1}{4})^{n/2}\|b_1\|.$$

Ainsi, dans une base réduite au sens de Lovàsz, le premier vecteur est "assez court". En pratique, les bases obtenues par LLL sont bien meilleures que ne le laisse croire le résultat précédent, de plus l'algorithme LLL est un algorithme polynomial. Afin de comprendre pourquoi, il nous faut introduire la notion de déterminant d'un réseau. Etant donné, une base $(b_1, \cdots, b_n)$ de L, on pose:

$$d(L) = \prod_{i=1}^{n} \|b_i^*\|$$

cette quantité semble dépendre de la base choisie, mais il n'en est rien, c'est en fait un invariant du réseau. Dans le cas particulier des réseaux de même dimension que l'espace vectoriel ambiant, cela se vérifie en constatant que:

$$d(L) = |\det(B)|$$

or, si B et B' sont deux bases du même réseau, il existe une matrice unimodulaire U ($\det(U) = \pm 1$), telle que $C = UB$, par conséquent $\det(C) = \pm\det(B)$. Considérons maintenant les n sous-réseaux initiaux de L, que nous noterons L_1, L_2, ..., L_n, et posons:

$$\forall i : D_i = d(L_i).$$

Remarquons maintenant que lors d'un échange entre b_i et b_{i+1}, tous les D_j pour $j \neq i$ restent inchangés, car les L_j ne changent pas. De plus, les conditions d'échanges imposent à D_i de diminuer d'un facteur $\sqrt{t}$ au moins. Par conséquent, lors d'un échange, quel qu'il soit, la quantité

$$D = \prod_{i=1}^{n} D_i$$

décroît d'un facteur $\sqrt{t}$ au moins. Initialement $D \leq A^{n(n+1)/2}$, et à la fin de la réduction $D \geq 1$, le nombre d'échanges de l'algorithme est donc majoré par $n(n+1)\log(A)/\log(t)$. De plus, le nombre de tours de l'algorithme vaut $2e + n - 1$, si e désigne le nombre d'échanges. Le nombre de tours est donc majoré par $n - 1 + 2n(n+1)\log(A)/\log(t)$, et le nombre d'opérations arithmétiques est $O(mn^3 \log(A))$.

Pour compléter la description de l'algorithme, il nous reste à expliquer les formules de mise à jour au moment des échanges. Soit u et v les projections respectives de b_i et b_{i+1} orthogonalement à L_{i-1}. Dans le cas particulier $i = 1$, on pose $u = b_1$ et $v = b_2$. On a les identités:

$$\begin{aligned} b_i^* &= u \\ b_{i+1}^* &= v - \frac{(v|u)}{\|u\|^2}u \end{aligned}$$

$$\begin{aligned}
\mu_{i+1,i} &= \frac{(v|u)}{\|u\|^2} \\
\forall k > i+1 :\ \mu_{k,i} &= \frac{(b_k|b_i^*)}{\|b_i^*\|^2} \\
\forall k > i+1 :\ \mu_{k,i+1} &= \frac{(b_k|b_{i+1}^*)}{\|b_{i+1}^*\|^2}
\end{aligned}$$

et sur les nouvelles valeurs:

$$\begin{aligned}
\hat{b}_i^* &= v \\
\hat{b}_{i+1}^* &= u - \frac{(v|u)}{\|v\|^2} v \\
\hat{\mu}_{i+1,i} &= \frac{(v|u)}{\|v\|^2} \\
\forall k > i+1 :\ \hat{\mu}_{k,i} &= \frac{(b_k|\hat{b}_i^*)}{\|\hat{b}_i^*\|^2} \\
\forall k > i+1 :\ \hat{\mu}_{k,i+1} &= \frac{(b_k|\hat{b}_{i+1}^*)}{\|\hat{b}_{i+1}^*\|^2}
\end{aligned}$$

De plus, D_{i+1} étant invariant lors de l'échange de b_i et b_{i+1}:

$$\|\hat{b}_i^*\|^2 \|\hat{b}_{i+1}^*\|^2 = \|b_i^*\|^2 \|b_{i+1}^*\|^2$$

Par conséquent, on a les formules de mise à jour suivantes:

$$\begin{aligned}
\|\hat{b}_i^*\|^2 &= \|b_{i+1}^*\|^2 + \mu_{i+1,i}^2 \|b_i^*\|^2 \\
\hat{\mu}_{i+1,i} &= \frac{\|b_i^*\|^2}{\hat{b}_i^*\|^2 \mu_{i+1,i}} \\
\|\hat{b}_{i+1}^*\|^2 &= \frac{\|b_i^*\|^2 \|b_{i+1}^*\|^2}{\|\hat{b}_i^*\|^2} \\
\forall k < i :\ \hat{\mu}_{i+1,k} &= \mu_{i,k} \\
\forall k < i :\ \hat{\mu}_{i,k} &= \mu_{i+1,k} \\
\forall k > i+1 :\ \hat{\mu}_{k,i+1} &= \mu_{k,i} - \mu_{i+1,i}\mu_{k,i+1} \\
\forall k > i+1 :\ \hat{\mu}_{k,i} &= \mu_{k,i+1} + \hat{\mu}_{i+1,i}\hat{\mu}_{k,i+1}
\end{aligned}$$

Pour préciser le temps de calcul de LLL, il faut borner supérieurement la taille des nombres pouvant apparaître au cours de l'algorithme, que ce soit au numérateur ou au dénominateur des fractions rationnelles. L'analyse précise de l'algorithme conduit à une borne de la forme $O(n \log(A))$. En 1988, Schnorr a proposé [34] une variante de LLL, utilisant une représentation flottante de M, cette variante possède des propriétés intéressantes de stabilité numérique et elle peut donc travailler sur des représentations des nombres avec $O(n + \log(A))$ bits de précision. C'est à partir de cette variante qu'il a été possible d'écrire des programmes de réduction de réseaux très performants en pratique[35].

1.6 La réduction au sens de Korkine-Zolotarev

Parfois, pour une application donnée, LLL n'est pas assez performant. Heureusement, il existe d'autres algorithmes efficaces, qui, bien que plus lents, permettent d'obtenir de meilleurs résultats. Dans cette section, nous présentons brièvement ces résultats, tout en invitant le lecteur intéressé à se reporter à [19] et [33].

Définition 1.6 *On dit d'une base $B = (b_1, b_2, \cdots, b_n)$ d'un réseau L qu'elle est réduite au sens de Korkine et Zolotarev, si et seulement si:*

- *b_1 réalise le premier minimum de L*
- *Pour tout $i > 1$, le projeté orthogonal de b_i sur l'espace vectoriel engendré par $(b_1, b_2, \cdots, b_{i-1})$ réalise le premier minimum du réseau engendré par les projetés de b_i, b_{i+1}, ..., b_n.*
- $\forall i < j : \ |(b_j|b_i^*)| \leq \frac{\|b_i^*\|^2}{2}$

Pour savoir calculer une base réduite au sens de Korkine et Zolotarev, il suffit de savoir calculer un vecteur réalisant le premier minimum d'un réseau. En effet, cela permet de calculer b_1, il suffit alors de compléter b_1 en une base du réseau, de projeter orthogonalement à b_1, puis de réaliser le premier minimum de ce nouveau réseau pour calculer b_2. En itérant ce processus, on peut facilement construire une base réduite. Malheureusement, on ne connaît pas, à ce jour, d'algorithme capable de calculer un vecteur réalisant le premier minimum d'un réseau en temps polynomial. En revanche, un algorithme exponentiel mais relativement efficace, permettant de réaliser le premier minimum à partir d'un base réduite au sens de Lovász a été présenté par Kannan dans [19]. De plus, à partir de cet algorithme, Schnorr a développé toute une hiérarchie d'algorithmes de réduction polynomiaux intermédiaires entre LLL et la réduction de Korkine-Zolotarev (cf [33]). Présentons maintenant les grandes lignes de l'algorithme de Kannan. C'est une recherche exhaustive du vecteur le plus court dans un espace relativement restreint. En effet, pour que le vecteur $\lambda_1 b_1 + \cdots + \lambda_n b_n$ puisse réaliser le premier minimum du réseau, il doit nécessairement être plus court que b_1, par conséquent, pour tout i son projeté orthogonalement au sous-espace engendré par b_1, $\cdots$, b_i doit être plus court que b_1. En particulier, il faut que:

$$\|\lambda_n b_n^*\| < \|b_1\|$$

et donc que

$$|\lambda_n| < \frac{\|b_1\|}{\|b_n^*\|} < 2^{n/2}.$$

Ainsi λ_n prend au plus $2^{1+n/2}$ valeurs. En étendant ce raisonnement aux autres λ_i, on limite la recherche à un espace de $2^{n^2/2}$ vecteurs environ.

A partir de cet algorithme exhaustif, Schnorr a proposé une famille d'algorithmes de réduction intermédiaires entre LLL et la réduction de Korkine-Zolotarev. Le principe de LLL consistait à utiliser un algorithme de réduction efficace en dimension 2 dans des sous-réseaux projetés de dimension 2. Le principe de l'algorithme de Schnorr, qu'il a nommé BKZ pour Blockwise Korkine-Zolotarev, consiste à utiliser un algorithme

efficace en dimension $2k$, à savoir l'algorithme exhaustif dont nous venons de parler, dans des sous-réseaux projetés de dimension $2k$. Plus exactement, le réseau est découpé en blocs de taille k, et à chaque étape, le réseau projeté engendré par 2 blocs consécutifs est réduit par l'algorithme de Kannan. Cette manoeuvre est répétée jusqu'à ce que tous les groupes de 2 blocs soient réduits au sens de Korkine Zolotarev. Schnorr a montré que dans le réseau ainsi obtenu, le premier vecteur b_1 est au plus c_k^n fois plus long que le vecteur le plus court du réseau. Les premières valeurs de c_k vérifient:

$$\begin{aligned} c_1 &\approx 1.1547 \\ c_2 &\leq 1.1225 \\ c_3 &\leq 1.1132 \\ c_4 &\leq 1.1066 \end{aligned}$$

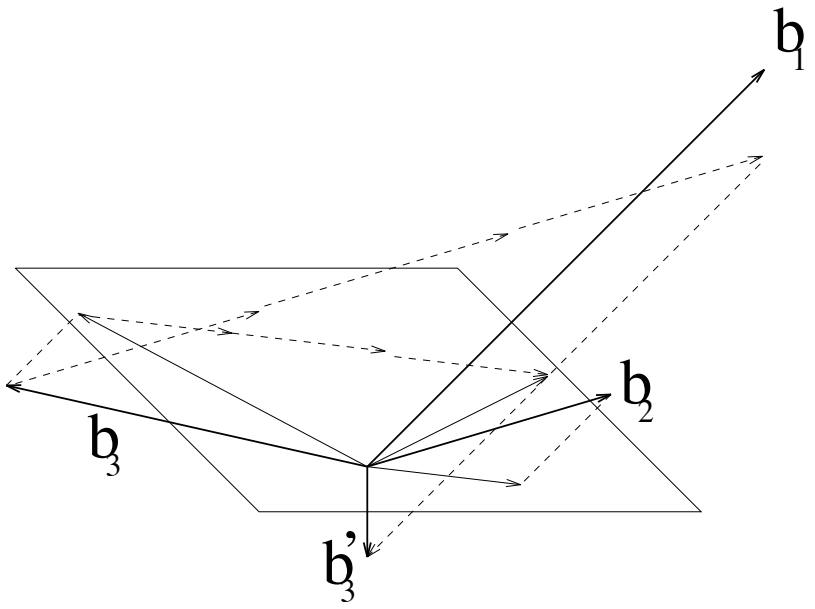


FIG. 1.9 - Exemple d'étape élémentaire de LLL

Chapitre 2

Les problèmes de sac-à-dos

2.1 Introduction

Le problème du sac-à-dos consiste, étant donnés des entiers positifs $a_1, \ldots, a_n$ et s à trouver un sous-ensemble de a dont la somme soit s. Ceci est équivalent à résoudre l'équation:

$$s = \sum_{i=1}^{n} e_i a_i$$

avec les e_i à valeur 0 ou 1. Ce problème est connu pour être NP-complet, il est donc considéré comme difficile dans le cas général. Cela a conduit à la conception de plusieurs systèmes cryptographiques basés sur le problème du sac-à-dos, comme par exemple, le cryptosystème de Merkle et Hellman[26]. Presque tous ces cryptosystèmes ont étés cryptanalysés avec succès, la plupart par des attaques spécifiques au système. De plus, deux algorithmes plus généraux ont été proposés, l'un par Brickell[1] et l'autre Lagarias et Odlyzko[21]. Ces algorithmes prouvent que presque tous les sac-à-dos de densité assez basse peuvent être résolus en temps polynômial.

Définition 2.1 *La densité d'un sac-à-dos $a_1, \ldots, a_n$ est le nombre*

$$d = \frac{n}{\log_2 \max_{1 \leq i \leq n} a_i}$$

Le cas intéressant est $d \leq 1$, car pour $d > 1$ il y a en général plusieurs solutions au même problème de sac-à-dos, il est donc difficile de coder ainsi une information utile. Les algorithmes de Brickell et Lagarias-Odlyzko permettent de résoudre presque tous les sac-à-dos pour d assez petit. Ces deux algorithmes ramènent le problème du sac-à-dos à un problème de recherche de vecteur court dans un réseau, puis utilisent LLL pour trouver ce vecteur. Associée à LLL, la méthode de Lagarias-Odlyzko permet de résoudre presque tous les sacs-à-dos de densité d inférieure à c/n, lorsque n est assez grand; la constante c est liée à l'utilisation de LLL. Elle peut être améliorée en utilisant à la place de LLL, l'un des algorithmes de réduction par blocs de Schnorr, cela augmente bien sûr le degré du polynôme déterminant le temps de calcul.

Bien que la recherche de vecteurs courts dans un réseau puisse être très difficile dans le cas général, les algorithmes connus s'avèrent en pratique bien plus performants que ne l'affirment leurs analyses théoriques. Il est donc fort possible qu'en moyenne la

recherche de vecteurs courts soit facile, même si dans le cas général elle est difficile. Il est donc parfaitement raisonnable de séparer l'efficacité de la recherche du vecteur court et la qualité de la transformation du sac-à-dos en recherche d'un vecteur court. Pour cela, nous allons utiliser un oracle, qui étant donnée une base d'un réseau entier calcule un vecteur réalisant le premier minimum de ce réseau. Dans les applications, cet oracle sera remplacé par un algorithme de réduction de réseau bien choisi. A partir d'un tel oracle, l'algorithme de Lagarias-Odlyzko est capable de résoudre presque tous les sac-à-dos de densité inférieure à une densité critique égale à $0.6463\cdots$, mais pas au-delà.

Nous décrirons ici deux modifications de l'algorithme de Lagarias-Odlyzko permettant de ramener la densité critique à $0.9408\cdots$. Des tests pratiques montrent que ces modifications améliorent considérablement les performances obtenues en utilisant à la place de l'oracle un algorithme approché comme LLL ou la réduction par blocs.

Les deux modifications que nous présentons ici résultent de deux recherches indépendantes, l'une par Jacques Stern et l'auteur de cette thèse, l'autre par Coster, LaMacchia, Odlyzko et Schnorr. Elles ont initialement été présentées séparément dans [7] et [18], puis ont été regroupées pour la publication finale. Nous présentons les deux variantes afin d'être le plus complet possible. Pour pouvoir prouver cette amélioration, nous allons utiliser un résultat de Mazo et Odlyzko sur les dénombrements de points dans les sphères. Ce résultat, paru dans [25] s'exprime par le théorème suivant, que nous ne démontrerons pas ici.

Théorème 2.1 *Si $N(x, n, \alpha)$ désigne le nombre de points à coordonnées entières dans la sphère n-dimensionnelle, centrée en x et de rayon $\sqrt{\alpha n}$. Alors, il existe une constante positive $c(\alpha)$ telle que:*

$$N(x, n, \alpha) \leq e^{c(\alpha)\sqrt{n}} N(x_0, n, \alpha),$$

où $x_0 = (0, 0, \cdots, 0)$. De plus, il existe une constante $C(\alpha)$ telle que:

$$N(x_0, n, \alpha) \leq C(\alpha)^n.$$

$C(\alpha)$ vaut $h(z_0)/z_0^\alpha$, où

$$h(z) = 1 + 2\sum_{k=1}^{\infty} z^{k^2},$$

et z_0 est l'unique solution dans $]0, 1[$ de l'équation

$$z\frac{h'}{h}(z) = \alpha.$$

2.2 Description de l'algorithme de Lagarias-Odlyzko

Dans leur article, Lagarias et Odlyzko ont montré que la densité critique de leur algorithme associé à un oracle pour les vecteurs courts est égale $0.6463\cdots$, une preuve plus simple à ensuite été décrite par Frieze. La preuve que nous présentons ici est inspirée par les méthodes de Frieze, mais en diffère légèrement.

Soient A un entier positif, et $a_1, \ldots, a_n$ des entiers positifs aléatoires $\leq A$. Soit $\mathbf{e} = (e_1, \cdots, e_n)$ un vecteur non nul formé de 0 et de 1 fixé. Posons

$$s = \sum_{i=1}^{n} e_i a_i, \quad \text{et} \quad t = \sum_{i=1}^{n} a_i.$$

On peut supposer, sans perte de généralité que $s \geq t/n$, sinon tous les $a_i \geq t/n$ peuvent être ignorés, diminuant ainsi n. De même, on peut supposer que $s \leq (1-1/n)t$, sinon tous les $a_i \geq t/n$ doivent être dans s et n peut encore être diminué. Donc,

$$\frac{1}{n}t \leq s \leq \frac{n-1}{n}t. \tag{2.1}$$

Rappelons maintenant le principe de l'attaque de Lagarias-Odlyzko. Soit B formée de vecteurs colonnes $b_1, \ldots, b_{n+1}$ et définie comme suit:

$$B = \begin{pmatrix} Ka_1 & Ka_2 & \cdots & Ka_n & Ks \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix}$$

où K est un entier positif que nous choisirons ultérieurement. Soit L le réseau engendré par B. On peut remarquer que le vecteur solution $\hat{\mathbf{e}} = {}^t(0, e_1, \cdots, e_n)$ est dans L. Tout comme dans [10], nous allons rechercher les vecteurs $\hat{\mathbf{x}} = (x_0, \cdots, x_n)$ vérifiant:

$$\begin{aligned} \|\hat{\mathbf{x}}\| &\leq \|\hat{\mathbf{e}}\|, \\ \hat{\mathbf{x}} &\in L, \\ \hat{\mathbf{x}} &\notin \{\mathbf{0}, \hat{\mathbf{e}}, -\hat{\mathbf{e}}\}. \end{aligned} \tag{2.2}$$

Quitte à remplacer s par $t-s$ et $\mathbf{e}$ par son complémentaire, on peut toujours supposer que

$$\sum_{i=1}^{n} e_i \leq \frac{n}{2}. \tag{2.3}$$

Ainsi, $\|\hat{\mathbf{e}}\| \leq \sqrt{\frac{n}{2}}$. Bien entendu, il devient alors nécessaire de réduire deux réseaux différents, l'un pour le sac-à-dos initial, l'autre pour le complémentaire. Deux appels à l'oracle seront donc nécessaires.

Posons $K > \sqrt{n}$, alors si $\hat{\mathbf{x}}$ satisfait les équations 2.2, nécessairement, on a $x_0 = 0$. Définissons maintenant y par:

$$ys = \sum_{i=1}^{n} x_i a_i, \tag{2.4}$$

alors:

$$|y|s = \left|\sum_{i=1}^{n} x_i a_i\right| \leq \|\hat{\mathbf{x}}\| \left|\sum_{i=1}^{n} a_i\right| \leq t\sqrt{\frac{n}{2}}. \tag{2.5}$$

Et donc, d'après l'équation 2.1:

$$|y| \leq n\sqrt{\frac{n}{2}}. \tag{2.6}$$

De plus, $-y$ étant le coefficient de b_{n+1} dans la décomposition de $\hat{\mathbf{x}}$ dans la base B du réseau, y est entier, il peut donc prendre au plus $2n\sqrt{n/2}+1$ valeurs différentes.

Nous allons montrer dans la suite que la probabilité P pour L de contenir un vecteur satisfaisant aux équations 2.2 vérifie:

$$P \leq n\left(2n\sqrt{\frac{1}{2}n}+1\right)\frac{2^{c_0 n}}{A}, \text{où } c_0 = 1.54724\cdots \tag{2.7}$$

Ceci implique que si $A = 2^{cn}$ avec $c > c_0$, alors $\lim_{n\to\infty} P = 0$. Et donc, si la densité d'un problème de sac-à-dos est inférieure à $0.6463\cdots$, alors

$$\begin{aligned} \frac{n}{\log_2 \max_{1\le i\le n} a_i} < 0.6463\cdots &\Rightarrow \max_{1\le i\le n} a_i > 2^{n/0.6463\cdots} \\ &\Rightarrow A > 2^{c_0 n} \end{aligned}$$

On en déduit donc, que tous les problèmes de densité inférieure à la densité critique $0.6463\cdots$ peuvent être résolus en temps polynômial à l'aide d'un oracle pour les vecteurs courts.

Prouvons maintenant l'équation 2.7. Soit $\mathbf{x} = (x_1, \cdots, x_n)$ à coordonnées entières. Il convient de noter que si $x_{n+1} = 0$, alors $\|\hat{\mathbf{x}}\| = \|\mathbf{x}\|$ et en particulier que $\|\hat{\mathbf{e}}\| = \|\mathbf{e}\|$. On a:

$$\begin{aligned} P &\le \Pr\left(\exists \mathbf{x}, y \text{ tq } \|\mathbf{x}\| \le \|\mathbf{e}\|, |y| \le n\sqrt{\frac{1}{2}n}, \mathbf{x} \notin \{\mathbf{0}, \mathbf{e}, -\mathbf{e}\}, \sum_{i=1}^{n} x_i a_i = ys\right), \\ &\le \Pr\left(\sum_{i=1}^{n} x_i a_i = ys : \ 0 < \|\mathbf{x}\| \le \|\mathbf{e}\|, |y| \le n\sqrt{\frac{1}{2}n}, \mathbf{x} \notin \{\mathbf{0}, \mathbf{e}, -\mathbf{e}\}\right) \\ &\quad \cdot |\{\mathbf{x} : \ \|\mathbf{x}\| \le \|\mathbf{e}\|\}| \cdot \left|\left\{y : \ |y| \le n\sqrt{\frac{1}{2}n}\right\}\right|. \end{aligned} \tag{2.8}$$

Nous allons estimer les 3 termes du membre de droite de l'équation 2.8, afin d'en déduire la borne désirée sur P. Dans le premier terme on peut remplacer $\sum_{i=1}^{n} x_i a_i = ys$ par

$$\sum_{i=1}^{n} z_i a_i = 0, \text{où } z_i = x_i - y e_i.$$

Les conditions 2.2 impliquent que $\mathbf{z}$ est non nul, on peut donc supposer, quitte à multiplier la borne sur P par un facteur au plus n, que $z_1 \neq 0$. Si l'on pose $z' = -\left(\sum_{i=2}^{n} a_i z_i / z_1\right)$, alors

$$\begin{aligned} \Pr\left(\sum_{i=1}^{n} a_i z_i = 0\right) &= \Pr\left(a_1 = z'\right), \\ &= \sum_{j=1}^{A} \Pr\left(a_1 = z' | z' = j\right) \Pr\left(z' = j\right), \\ &= \sum_{j=1}^{A} \Pr\left(a_1 = z'\right) \Pr\left(z' = j\right), (a_1 \text{ et } j \text{ sont indépendants}), \\ &= \sum_{j=1}^{A} \frac{1}{A} \Pr\left(z' = j\right), \\ &\le \frac{1}{A}. \end{aligned}$$

Considérons maintenant le second terme dans l'équation 2.8. D'après le théorème 2.1, on sait que:

$$|\{\mathbf{x} : \ \|\mathbf{x}\| \le \|\mathbf{e}\|\}| \le \left|\left\{\mathbf{x} : \ \|\mathbf{x}\| \le \sqrt{\frac{1}{2}n}\right\}\right| \le 2^{c_0 n}, \text{où } c_0 = \log_2(C(1/2)) = 1.54724\cdots \tag{2.9}$$

Le troisième et dernier terme vaut clairement $2n\sqrt{\frac{n}{2}}+1$, on en déduit donc 2.7.

En fait, la preuve ci-dessus est plus compliquée qu'elle ne devrait être. En effet, il serait bien plus naturel de considérer que:

$$B = \begin{pmatrix} Ka_1 & Ka_2 & \cdots & Ka_n & -Ks \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix}$$

Alors, il deviendrait inutile de s'interroger sur la valeur de y, cette valeur étant (au signe prés) la dernière coordonnée du vecteur x dans ce nouveau réseau, la preuve en serait donc simplifiée d'autant. Avec cette nouvelle approche, la recherche du vecteur court dans le réseau revient à chercher une solution entière de:

$$\sum_{i=1}^{n} a_i x_i = s x_{n+1}$$

minimisant

$$\sum_{i=1}^{n+1} x_i^2.$$

On comprend bien que quand les solutions sont assez rares, la solution à valeur dans $\{0,1\}$ est la plus petite. La première approche a été initialement préférée, car elle utilise une matrice carrée et qu'à l'origine LLL était conçu pour ce type de réseau. C'est cette approche qui, pour cette raison, est donc le plus souvent présentée. Nous n'avons mentionné la seconde que pour aider à la compréhension de la section suivante, présentant une amélioration de la densité critique.

2.3 Amélioration de la densité critique

Le résultat principal de ce chapitre est l'amélioration de la densité critique à la nouvelle valeur $0.9408\cdots$, par la méthode découverte par Jacques Stern et l'auteur. Nous venons de voir que l'idée de base de l'attaque de Lagarias-Odlyzko, consistait à rechercher une solution de l'équation:

$$\sum_{i=1}^{n} a_i x_i = s x_{n+1}$$

minimisant

$$\sum_{i=1}^{n+1} x_i^2.$$

Cela permettait d'éliminer les solutions inintéressantes avec x grand. Malheureusement, cette norme ne distingue pas les 1 des -1. Pour cette raison, on voit apparaître, au dessus de la densité critique des vecteurs courts formés à la fois de 1, de 0 et de -1,

qui ne répondent donc pas à la question posée. Pour régler ce problème nous proposons d'utiliser à la place de la norme usuelle la quantité suivante:

$$\sum_{1\leq i<j\leq n+1}(x_i-x_j)^2=(n+1)\sum_{i=1}^{n+1}x_i^2-(\sum_{i=1}^{n+1}x_i)^2.$$

En effet, cette quantité est minimale quand les x_i sont les plus proches possibles les uns des autres. Cependant, cette quantité n'est pas une norme, mais juste une semi-norme. Heureusement, pour en faire une norme, il suffit de lui ajouter une fraction de la norme usuelle, définissons donc:

$$\| < x > \|^2=(n+1+\lambda)\sum_{i=1}^{n+1}x_i^2-(\sum_{i=1}^{n+1}x_i)^2.$$

Pour pouvoir choisir λ, montrons comment traduire ce changement de norme en terme de réseau. Remplaçons B par:

$$B'=\begin{pmatrix} Ka_1 & Ka_2 & \cdots & Ka_n & -Ks \\ n+1 & -1 & \cdots & -1 & -1 \\ -1 & n+1 & \cdots & -1 & -1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ -1 & -1 & \cdots & n+1 & -1 \\ -1 & -1 & \cdots & -1 & n+1 \end{pmatrix}$$

Notons $\hat{\mathbf{x}}'=\sum_{i=1}^{n+1}x_i\mathbf{b}'_i$ et remarquons que si $\hat{\mathbf{x}}'_0=0$, alors

$$\begin{aligned}\|\hat{\mathbf{x}}'\|^2 &= \sum_{i=1}^{n+1}((n+1)x_i-\sum_{j\neq i}x_j)^2 \\ &= (n^2+3n+1)\sum_{i=1}^{n+1}x_i^2-2(n+3)\sum_{1\leq i<j\leq n+1}x_ix_j \\ &= (n+2)^2\sum_{i=1}^{n+1}x_i^2-(n+3)(\sum_{i=1}^{n+1}x_i)^2 \\ &= (n+3)\| < x > \|^2\end{aligned}$$

avec

$$\lambda=\frac{(n+2)^2}{n+3}-n-1=\frac{1}{n+3}.$$

Pour évaluer la nouvelle densité critique, il nous suffit de calculer la nouvelle valeur du second terme du membre droit de l'équation 2.8. Il nous suffit donc de borner supérieurement le nombre de x tels que:

$$\| < x > \|^2\leq\| < e > \|^2.$$

Soit α la proportion de 0 dans e alors:

$$\| < e > \|^2=\alpha(1-\alpha)(n+1)^2+\alpha\lambda(n+1)\leq\frac{(n+1)^2}{4}+\lambda(n+1).$$

Il suffit de choisir $K \geq n+1$ pour que

$$\| < x > \|^2 \leq \| < e > \|^2 \Rightarrow \hat{\mathbf{x}'}_0 = 0.$$

Donc

$$\begin{aligned} \| < x > \|^2 \leq \| < e > \|^2 \quad &\Rightarrow \quad \lambda \sum_{i=1}^{n+1} x_i^2 \leq \| < e > \|^2 \\ &\Rightarrow \quad \sum_{i=1}^{n+1} x_i^2 = O(n^3) \end{aligned}$$

si l'on pose

$$t = \frac{x_1 + x_2 + \cdots + x_{n+1}}{n+1}$$

l'inégalité de Schwarz implique:

$$|t| \leq \frac{\sqrt{n+1} O(n^{3/2})}{n+1} = O(n).$$

Nous pouvons maintenant minorer $\| < x > \|^2$ par:

$$\sum_{1 \leq i < j \leq n+1} ((x_i - t) - (x_j - t))^2 = (n+1) \sum_{i=1}^{n+1} (x_i - t)^2.$$

Nous avons donc prouvé que x se situe dans l'une des sphères de rayon $\xi\sqrt{n}$ centrée en $(t, t, \cdots, t)$. Il y a $O(n)$ telles sphères et ξ tend vers $1/4$ quand n tend vers l'infini. En utilisant l'analyse de Frieze, et l'étude du nombre de points dans les sphères de Mazo et Odlyzko, nous trouvons comme densité critique:

$$\lim_{n \to \infty, \xi \to 1/4} \frac{n}{\log_2(O(n) \exp(c'\sqrt{n}) C(\xi)^n)} = \frac{1}{\log_2(C(1/4))} = 0.9408 \cdots$$

De plus, on peut remarquer que si dans la spécification du problème de sac-à-dos, la proportion de 0 dans e est précisée, et vaut α, alors la densité critique devient $1/\log_2(C(\alpha(1-\alpha)))$.

2.4 Une autre amélioration de la densité critique

Dans la section précédente, nous avons montré comment atteindre une densité critique de $0.9408 \cdots$. Voici une seconde méthode, découverte par Coster, LaMacchia, Odlyzko et Schnorr, et permettant d'atteindre le même objectif. Nous utiliserons cette fois, le réseau engendré par:

$$B'' = \begin{pmatrix} Ka_1 & Ka_2 & \cdots & Ka_n & Ks \\ 1 & 0 & \cdots & 0 & 1/2 \\ 0 & 1 & \cdots & 0 & 1/2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 1/2 \end{pmatrix}$$

Dans ce réseau le vecteur solution devient $\hat{\mathbf{e}}'' = {}^t(0, e_1 - 1/2, \cdots, e_n - 1/2)$. Encore une fois, nous cherchons le nombre de vecteurs vérifiant:

$$\begin{aligned} \|\hat{\mathbf{x}}\| &\leq \|\hat{\mathbf{e}}''\|, \\ \hat{\mathbf{x}} &\in L'', \\ \hat{\mathbf{x}} &\notin \{\mathbf{0}, \hat{\mathbf{e}}'', -\hat{\mathbf{e}}''\}. \end{aligned} \tag{2.10}$$

En choisissant $K > \frac{1}{2}\sqrt{n}$, un tel $\hat{\mathbf{x}}$ vérifie $x_0 = 0$. Si $\hat{\mathbf{x}} = \sum_{i=1}^{n} y_1 \mathbf{b_i} + y\mathbf{b''_{n+1}}$ satisfait l'équation 2.10, on peut écrire:

$$x_i = y_i + \frac{1}{2}y, \text{ pour } 1 \leq i \leq n,$$

$$0 = x_0 = K \cdot \left\{ \sum_{i=1}^{n} a_i y_i + ys \right\}.$$

Alors:

$$\sum_{i=1}^{n} a_i y_i = -ys.$$

On peut donc remplacer l'équation 2.5 par:

$$\sum_{i=1}^{n} x_i a_i = \frac{1}{2} y(t - 2s), \tag{2.11}$$

puisque $(\sum_{i=1}^{n} \mathbf{b_i} - 2\mathbf{b''_{n+1}} = (K(t-2s), 0, 0, \cdots, 0)$. Majorons maintenant $|y|$:

$$\begin{aligned} |y(t-2s)| &= 2\left|\sum_{i=1}^{n} x_i a_i\right| \\ &\leq n\alpha\sqrt{n}, \text{ où } \alpha = \max_{1 \leq i \leq n} a_i. \end{aligned} \tag{2.12}$$

Si $|t - 2s| \geq \frac{1}{2}\alpha$, alors $|y||t - 2s| \geq \frac{1}{2}|y|\alpha$, et donc:

$$|y| \leq 2n\sqrt{n}, \tag{2.13}$$

d'après l'équation 2.12. Sinon, deux cas sont possibles, α est soit dans s, soit dans $t - s$. Dans le premier cas, en supprimant α le problème se transforme en un nouveau problème où $s' = s - \alpha, t' = t - \alpha$, et:

$$|t' - 2s'| = |t - \alpha - 2s + 2\alpha| = |t - 2s + \alpha| \geq \frac{1}{2}\alpha. \tag{2.14}$$

Dans le deuxième cas, $s' = s, t' = t - \alpha$, et:

$$|t' - 2s'| = |t - 2s - \alpha| \geq \frac{1}{2}\alpha. \tag{2.15}$$

Dans tous les cas, on peut donc supposer que $|t - 2s| \geq \frac{1}{2}\alpha$, et l'on peut donc utiliser la borne de l'équation 2.13.

Majorons maintenant la probabilité P d'existence d'un vecteur $\hat{\mathbf{x}}$ satisfaisant l'équation 2.10. Si $\mathbf{x} = (x_1, \cdots, x_n)$ désigne un vecteur demi-entier quelconque, on obtient la majoration suivante:

$$P \leq \Pr\left(\sum_{i=1}^{n} a_i x_i = \frac{1}{2}y(t-2s)\right) \cdot \left|\left\{\mathbf{x} \text{ tq } \|\mathbf{x}\| \leq \frac{1}{2}\sqrt{n}\right\}\right| \cdot (4n\sqrt{n}+1)\,. \tag{2.16}$$

Comme à la section 2.2, $\Pr(\sum_{i=1}^{n} a_i x_i = \frac{1}{2}y(t-2s)) \leq n/A$. Le facteur n dans cette borne, provient de l'hypothèse $x_1 \neq 0$. Le nombre de $\mathbf{x}$ vérifiant $\|\mathbf{x}\| \leq \frac{1}{2}\sqrt{n}$ est majoré par:

$$\left|\left\{\mathbf{w} \in \mathbf{Z}^n \text{ tq } \|\mathbf{w}\| \leq \frac{1}{2}\sqrt{n}\right\}\right| + \left|\left\{\mathbf{w} \in \mathbf{Z}^n \text{ tq } \|\mathbf{w} - (\frac{1}{2}, \cdots, \frac{1}{2})\| \leq \frac{1}{2}\sqrt{n}\right\}\right| \tag{2.17}$$

D'après le théorème 2.1, pour n assez grand, le second terme de l'équation précédente est plus petit que le premier, on obtient la majoration:

$$\left|\left\{\mathbf{x} \text{ tq } \|\mathbf{x}\| \leq \frac{1}{2}\sqrt{n}\right\}\right| \leq 2^{c'_0 n}, \text{ où } c'_0 = 1.0628\cdots,$$

et donc:

$$P \leq n\,(4n\sqrt{n}+1)\,\frac{2^{c'_0 n}}{A}.$$

Ceci prouve, de nouveau, que tous les problèmes de sac-à-dos de densité inférieure à $1/c'_0 = 0.9408\cdots$ peuvent être résolus en temps polynômial, à l'aide d'un oracle pour les vecteurs courts. Cependant, comme dans l'attaque de Lagarias-Odlyzko, on peut remarquer que la preuve est plus compliquée qu'elle ne devrait, en effet, prenons la variante suivante de B'':

$$B'' = \begin{pmatrix} Ka_1 & Ka_2 & \cdots & Ka_n & Ks \\ 1 & 0 & \cdots & 0 & 1/2 \\ 0 & 1 & \cdots & 0 & 1/2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 1/2 \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix}.$$

Alors, il devient inutile d'écrire des équations pour borner y. De plus, comme dans la preuve de cette attaque, on ne distingue plusieurs cas que pour borner y, avec cette nouvelle variante, cette distinction devient inutile. Une seule réduction de réseau par l'oracle est donc nécessaire, au lieu de deux avec le réseau B'' initial.

2.5 Influence de la proportion de 0 dans e

Dans cette section, nous travaillerons avec les variantes de B et B'' de dimension $(n+1)\times(n+2)$. Nous utiliserons aussi le réseau $\bar{B}$, obtenu comme B, mais en remplaçant le sac-à-dos par son complémentaire, c'est-à-dire, s par $t-s$. De plus, nous allons

considérer d'autres variantes de B'' définies par:

$$B''(\gamma) = \begin{pmatrix} Ka_1 & Ka_2 & \cdots & Ka_n & Ks \\ 1 & 0 & \cdots & 0 & 1-\gamma \\ 0 & 1 & \cdots & 0 & 1-\gamma \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 1-\gamma \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix}.$$

Bien évidemment, $B'' = B''(1/2)$.

Nous allons établir la densité critique de tous les réseaux que nous avons introduits quand la proportion α de 0 dans e varie, afin de mieux comparer les trois attaques que nous venons de voir. Le calcul de ces densités critiques est une variation très simple des calculs que nous avons faits dans les sections précédentes, nous donnons donc les résultats sans les assortir de leur démonstration. La densité critique de B est $c(\alpha) = 1/\log_2(C(\alpha))$. Celle de $\bar{B}$ est $\bar{c}(\alpha) = c(1-\alpha)$. Celle de B' est $c'(\alpha) = 1/\log_2(C(\alpha(1-\alpha)))$. Et enfin, celle de $B''(\gamma)$ est $c''(\gamma,\alpha) = 1/\log_2(C(\alpha(1-2\gamma)+\gamma^2))$. De plus, la densité critique $c_0(\alpha)$ séparant des autres, les sacs-à-dos ayant au plus une solution vaut $-1/\log 2(\alpha^\alpha(1-\alpha)^{1-\alpha})$. La qualité d'un algorithme de résolution de sac-à-dos du type de ceux que nous venons de décrire peut s'évaluer par comparaison de sa densité critique à $c_0(\alpha)$. Pour bien illustrer toutes les réductions que nous avons présentées, nous incluons plusieurs courbes représentant les densités critiques.

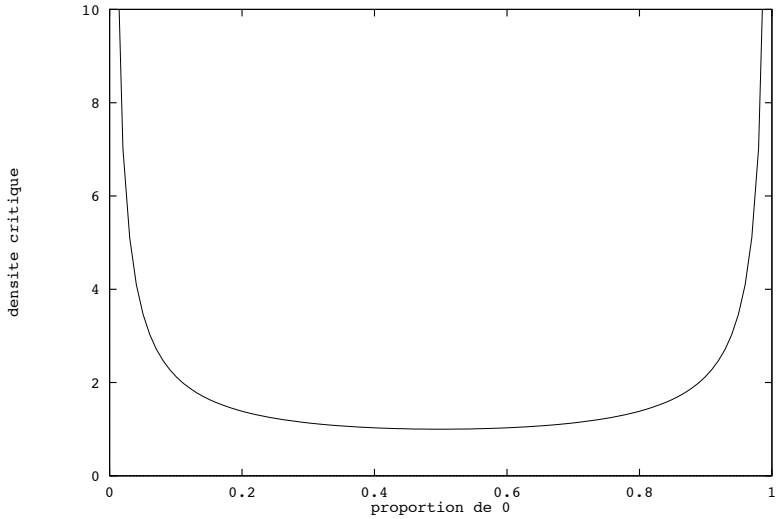


FIG. 2.1 - Densité critique absolue c_0

Commentons brièvement ces courbes, tout d'abord, on voit sur les figures 2.2 et 2.4, que l'attaque de Lagarias-Odlyzko et celle de Coster, LaMacchia, Odlyzko et Schnorr sont en fait formées de plusieurs réductions distinctes, et que la densité critique de ces attaques est en fait le maximum des densités critiques des sous-attaques qui les

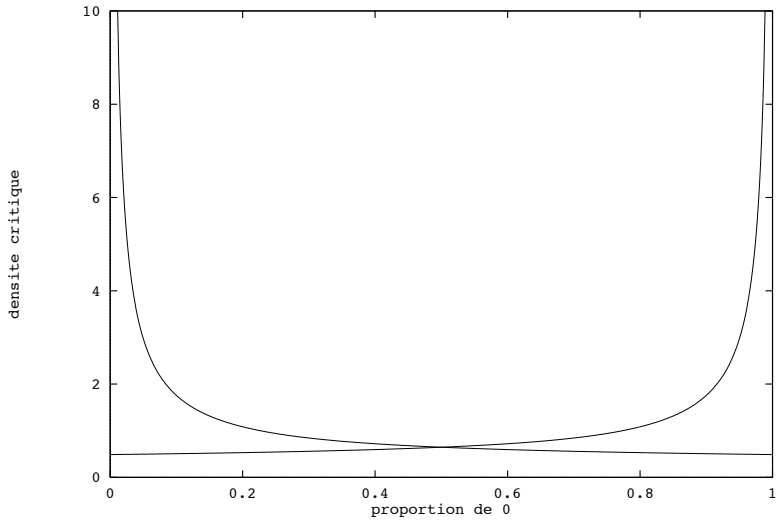


FIG. 2.2 - Densités critiques c et $\bar{c}$

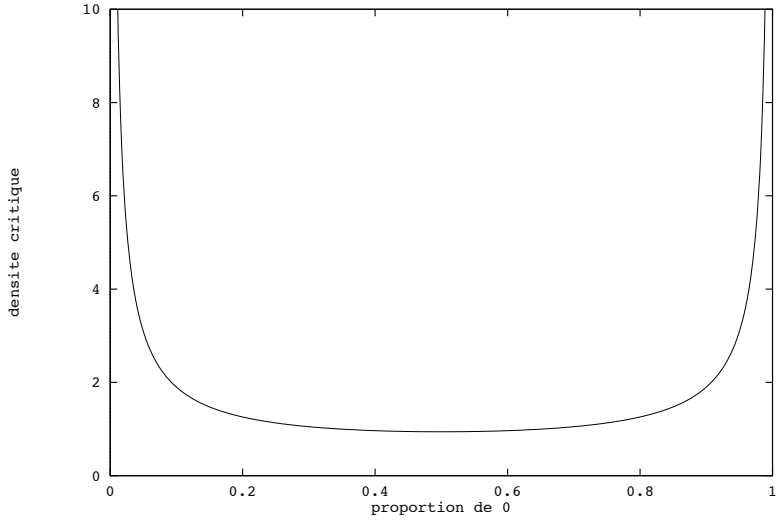


FIG. 2.3 - Densité critique c'

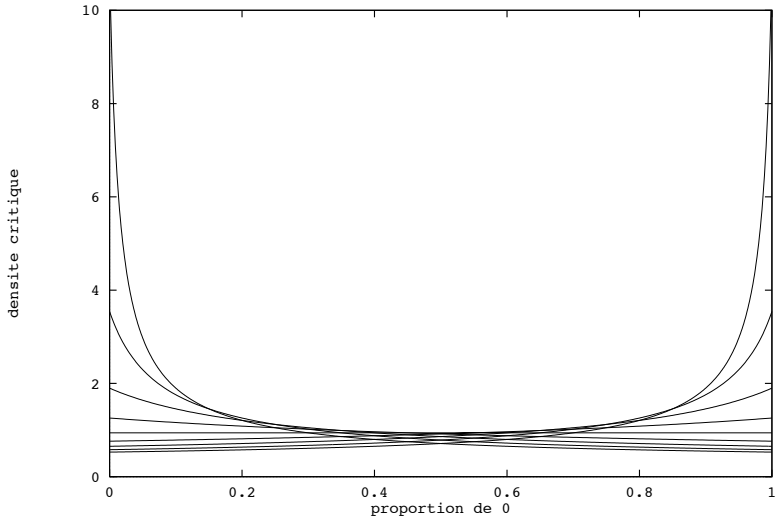


FIG. 2.4 - Densités critiques $c''(0.1 \cdots 0.9)$

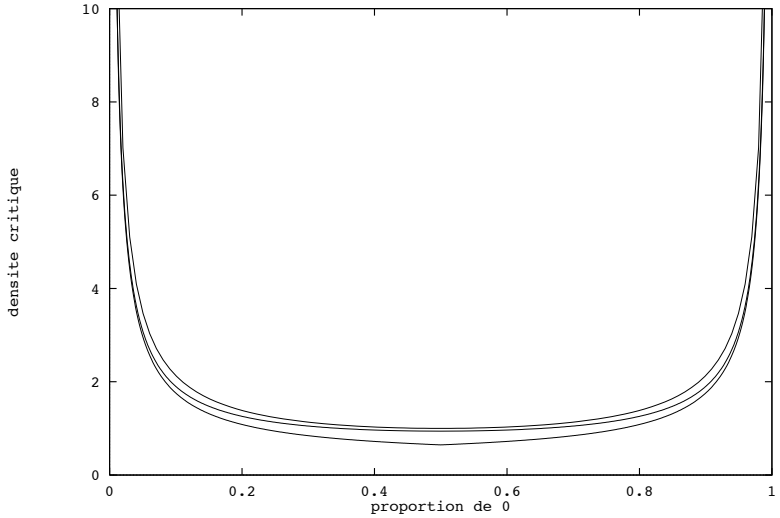


FIG. 2.5 - Densités critiques c_0, c' et $\max(c, \bar{c})$

composent. En fait, dans le second cas, pour un sac-à-dos de proportion α, la densité optimale est fournie par $B''(\alpha)$ et de plus, $c'(\alpha) = c''(\alpha, \alpha)$. Les deux attaques "moyenne densité" que nous avons décrites, ont donc la même courbe de densité critique. Dans les deux dernières figures, nous comparons, les courbes "basse densité" et "moyenne densité", à la courbe théorique c_0. On peut constater, que le gain apporté par les attaques "moyenne densité" est maximum pour $\alpha = 1/2$, de plus, quand α tend vers 0 ou vers 1, les courbes "moyenne" et "basse" se confondent.

Rappelons brièvement que la densité critique théorique c_0 s'obtient par un simple calcul de théorie de l'information en comparant les cardinalités des espaces de départ et d'arrivée de la fonction qui à e associe $\sum_{i=1}^{n} a_i e_i$. Si la proportion de 0 dans e est fixée et vaut α, l'espace de départ a pour cardinalité $C_n^{\alpha n}$. L'espace d'arrivée a lui une cardinalité de l'ordre de n/d. En égalant ces deux quantités, on obtient

$$c_0(\alpha) = \frac{-1}{x \log_2(x) + (1-x) \log_2(1-x)}.$$

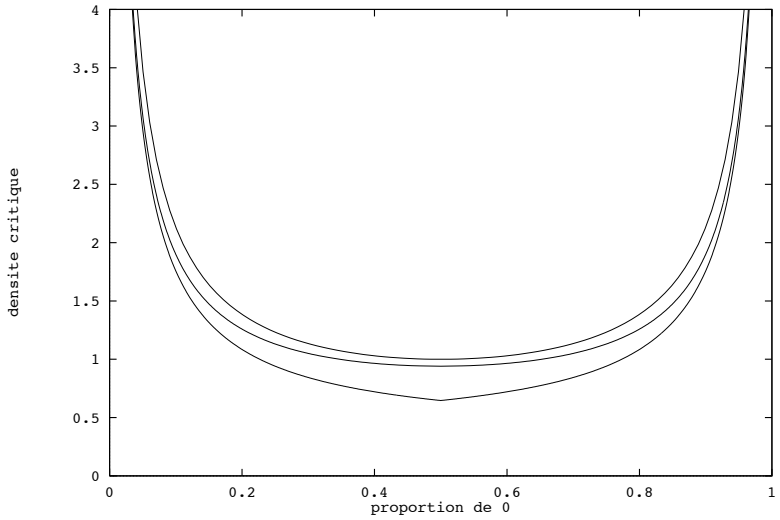


FIG. 2.6 - Densités critiques c_0, c' et $\max(c, \bar{c})$ (Gros Plan)

Chapitre 3

La parallélisation de LLL

L'algorithme LLL que nous avons décrit au chapitre 1 a deux défauts majeurs qui diminuent grandement son efficacité. D'une part, l'usage de nombres rationnels représentés exactement rend prohibitif le coût des opérations arithmétiques, de l'autre, l'algorithme tel qu'il est présenté peut facilement être vectorisé, mais il ne se prête pas à une parallélisation plus poussée. Le premier problème a été résolu autant d'un point de vue théorique que pratique dans [34] et [35]. En ce qui concerne la parallélisation, une première approche utilisant l'arithmétique rationnelle a été proposé par Villard, malheureusement, le coût de l'arithmétique fait que cette approche est moins rapide avec nm processeurs que l'algorithme en nombres flottants de Schnorr avec un seul. Nous présentons ici une façon de réconcilier les deux approches pour obtenir un algorithme parallèle travaillant sur des nombres flottants, et donc très rapide. Plus précisément, le résultat principal de ce chapitre est le suivant:

Théorème 3.1 *Il existe un algorithme de réduction de réseau entier, qui étant donnée une base B d'un réseau entier de dimension n d'un espace euclidien de dimension m vérifiant les propriétés suivantes:*

- *Pour tout i, $\|b_i\|^2 \leq A$*
- $\gamma = \min_{1 \leq i \leq n} \frac{\|b_i^*\|^2}{\|b_i\|^2}$
- $m \leq 2n$

calcule une base B' réduite au sens de Lovász du même réseau. Cet algorithme utilise nm processeurs et effectue $O(n^3 A/E(B))$ opérations arithmétiques sur des nombres de tailles $O(n + \log(A) + \log(1/\gamma))$. $E(B)$ est une quantité dépendante de la base B et vérifiant $1 \leq E(B) \leq n/2$.

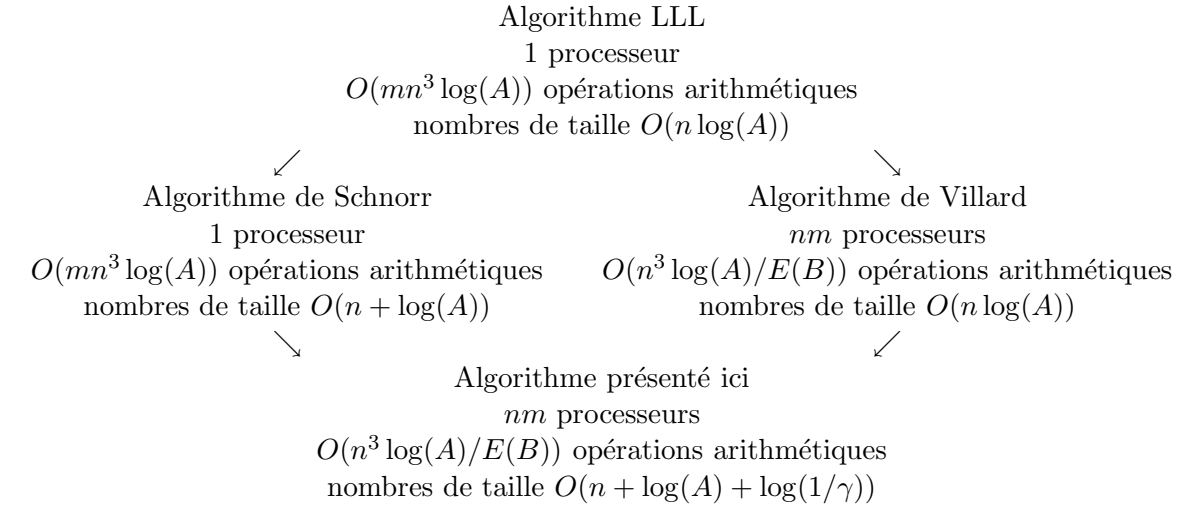
De plus, des expérimentations pratiques semblent suggérer que:

Conjecture 3.1 *Il existe une constante c telle que:*

$$\lim_{n \to \infty} Pr(E(B) < cn \mid dim(B) = n) = 0.$$

La quantité γ mesure en quelque sorte la qualité de la base à réduire, mais cette mesure est peu restrictive, et pour presque toutes les bases d'usage courant, la taille

des nombres est en fait $O(n + \log(A))$. Voici, maintenant un schéma comparatif des algorithmes de réduction au sens de Lovász, afin de replacer ce travail dans son contexte:



L'algorithme tel que nous le présentons ici n'est vraisemblablement pas l'algorithme à utiliser dans la pratique, en revanche, il montre qu'il est possible de parallèliser LLL sans perdre les propriétés de stabilité numérique introduites par Schnorr. La même dichotomie s'est déjà produite dans le cas séquentiel, en effet, Schnorr a tout d'abord présenté un algorithme théorique prouvable, avant de proposer une version pratique efficace mais heuristique.

Dans la suite de ce chapitre, nous utiliserons comme modèle de parallélisme pour la réduction d'une base B de n vecteurs dans un espace de dimension m, un réseau de nm processeurs groupés en n colonnes de m processeurs, et communiquant selon une géométrie torique:

$$\begin{array}{ccccccccc}
 & \updownarrow & & \updownarrow & & \cdots & & \updownarrow & \\
\leftrightarrow & b_{1,1} & \leftrightarrow & b_{2,1} & \leftrightarrow & \cdots & \leftrightarrow & b_{n,1} & \leftrightarrow \\
 & \updownarrow & & \updownarrow & & \cdots & & \updownarrow & \\
\leftrightarrow & b_{1,2} & \leftrightarrow & b_{2,2} & \leftrightarrow & \cdots & \leftrightarrow & b_{n,2} & \leftrightarrow \\
 & \updownarrow & & \updownarrow & & \cdots & & \updownarrow & \\
\leftrightarrow & \vdots & \leftrightarrow & \vdots & \leftrightarrow & \ddots & \leftrightarrow & \vdots & \leftrightarrow \\
 & \updownarrow & & \updownarrow & & \cdots & & \updownarrow & \\
\leftrightarrow & b_{1,m} & \leftrightarrow & b_{2,m} & \leftrightarrow & \cdots & \leftrightarrow & b_{n,m} & \leftrightarrow \\
 & \updownarrow & & \updownarrow & & \cdots & & \updownarrow &
\end{array}$$

Cette géométrie, peu restrictive, existe sur la plupart des machines parallèles, et ne sera donc pas un obstacle à l'implantation de LLL sur une machine parallèle. De plus, la relation entre le réseau de processeurs et la base B à réduire est très simple, en effet, chaque processeur s'occupe d'un élément de la matrice B, celui dont la position dans B coïncide avec sa position dans le réseau de processeurs. Cela nous permet d'alléger l'écriture des algorithmes dans la suite de ce chapitre. Ainsi, nous n'utiliserons que deux notations simples pour faire apparaître le parallélisme, la première consistera à écrire dans l'algorithme des opérations vectorielles, la seconde à utiliser une formule du type "Pour tous les i dans $1 \cdots n$ (tels que *condition*(i))". La notation "Pour i allant

de 1 à n" désignera toujours une boucle séquentielle, les opérations à l'intérieur de cette boucle seront bien entendu effectuées sur tous les processeurs en même temps. Parmi les opérations vectorielles, il faut noter que le produit scalaire de deux vecteurs de dimensions m, coûte le prix d'une multiplication de deux nombres, les m multiplications nécessaires étant faites en parallèle, plus celui d'une addition globale de m nombres, qui peut se faire en temps $m+t$ si t est la taille de ces nombres. Nous compterons toujours un produit scalaire comme $O(1)$ opérations arithmétiques, par conséquent, il nous faut supposer que m n'est pas trop grand, par exemple, que $m < 2n$, ce qui devrait suffire pour tous les cas usuels.

3.1 L'approche de Villard

Nous présentons ici l'idée très naturelle qui a été développée dans [41] et qui mène à un algorithme parallèle utilisant des nombres rationnels. Nous reprenons ici les notations du premier chapitre, où pour majorer le temps d'exécution de LLL, nous avions montré qu'à chaque échange D était multiplié par un facteur plus petit que $t < 1$. La stratégie de LLL consiste à choisir à chaque tour le premier échange permettant d'assurer cette décroissance. En fait, n'importe quel choix d'échange conviendrait, tant que l'on assure la décroissance de D. Mieux encore, rien n'interdit d'opérer plusieurs échanges simultanément, à condition bien sûr qu'ils n'interfèrent pas. Il est facile de constater, que si deux échanges mettent en jeu des paires de vecteurs adjacents disjointes, et s'ils améliorent D respectivement d'un facteur t_1 et d'un facteur t_2, on peut alors effectuer les deux échanges simultanément et ainsi améliorer D d'un facteur $t_1 t_2$. La première idée qui vient maintenant à l'esprit est de chercher à réaliser le plus d'échanges possibles simultanément. Nous obtenons alors un algorithme parallèle structuré par tour, chaque tour comprenant une phase de translation et une phase d'échange. Il suffit, en fait,de décrire notre stratégie de choix des échanges pour compléter ce premier LLL parallèle. Cette stratégie est très simple, il suffit de choisir le premier échange satisfaisant à la condition de décroissance de D, d'ignorer l'échange suivant même s'il est satisfaisant car il interfère, puis de recommencer ainsi jusqu'à avoir parcouru tous les échanges. Ceci coûte un peu de temps de communication, mais ce temps est négligeable par rapport au reste de l'algorithme. Un paramètre très important pour cet algorithme est le nombre moyen d'échanges par tours, en effet, c'est ce nombre qui détermine le gain de temps par rapport à un algorithme séquentiel. Nous le noterons $E(B)$, car il dépend de la base à réduire. Il est clair que l'algorithme parallèle rationnel ainsi obtenu permet en utilisant nm processeurs de gagner un facteur de temps $mE(B)$ par rapport à l'algorithme séquentiel classique. Cependant, comparé à l'algorithme de Schnorr, cet algorithme n'est pas rentable. Dans la suite de ce chapitre, nous allons montrer comment conjuguer les deux approches, afin de gagner à la fois par la parallélisation et par la stabilité numérique.

3.2 L'orthogonalisation de Givens

L'algorithme LLL étant basé sur l'orthogonalisation de Gram-Schmidt, il hérite de ses problèmes d'instabilité numérique. C'est pour cela que l'utilisation d'une arithmétique rationnelle est nécessaire. Dans l'algorithme de Schnorr, une autre méthode d'or-

thogonalisation est utilisée pour stabiliser l'algorithme et rendre possible l'utilisation de nombres flottants. Cependant cette méthode est intrinsèquement séquentielle, il nous faut donc utiliser autre chose. En fait, le problème de l'orthogonalisation de manière parallèle et stable numériquement est un problème qui a déjà été résolu. La méthode a été décrite dans [32], sous le nom d'orthogonalisation de Givens. Nous allons décrire en détail cette méthode, avant de passer à la parallélisation de LLL. L'algorithme d'orthogonalisation que nous allons décrire ici, calcule à partir d'une matrice A, carrée $n \times n$ une décomposition (Q, R) vérifiant:

- R triangulaire supérieure
- Q orthogonale ($^tQQ = I$)
- $QA = R$, ou de manière équivalente $A = {}^tQR$.

Il est clair qu'une telle décomposition (Q, R) équivaut à une décomposition (B^*, M), en effet, seule la normalisation diffère entre ces deux décompositions. Dans la première Q est normalisée par la condition $^tQQ = I$, dans la seconde M est normalisée avec une diagonale de 1. Pour passer d'une décomposition à l'autre, il suffit de poser $B^* = {}^tQ\Delta$ et $M = \Delta^{-1}R$, où Δ est la matrice diagonale formée des normes des b_i^*.

Le principe de l'algorithme est de fabriquer Q comme un produit de rotations planes bien choisies. Plus précisément, $Q = Q_{2n-3} \cdots Q_2 Q_1$ où chaque Q_k est une somme directe de rotations planes indépendantes, et s'écrit comme une matrice diagonale par blocs avec des blocs de taille 2×2 sur la diagonale. Quand k est dans l'intervalle 1, 2, ..., $n-1$:

$$Q_k = \prod_{j=1, i=n-k+2j-2}^{j=\lceil k/2 \rceil} P_{i,i+1}^{(j)}$$

et quand k est dans l'intervalle n, ..., $2n-3$:

$$Q_k = \prod_{j=k-n+2, i=2j-(k-n+2)}^{j=\lceil k/2 \rceil} P_{i,i+1}^{(j)}.$$

Chaque transformation $P_{i,i+1}^{(j)}$ est une rotation plane qui mélange les lignes i et $i+1$ pour faire apparaître un zéro en position $(i+1, j)$.

$$P_{i,i+1}^{(j)} = \left(\begin{array}{c|cc|c} I_{i-1} & 0 & & 0 \\ \hline 0 & \begin{matrix} c_i & s_i \\ -s_i & c_i \end{matrix} & & 0 \\ \hline 0 & 0 & & I_{n-i-1} \end{array}\right)$$

Les coefficients c_i et s_i vérifient $c_i^2 + s_i^2 = 1$ et assurent l'annulation du coefficient en position $(i+1, j)$. Tout ceci mène à l'algorithme suivant:

Ortho
Entrée: A
Sortie: Q et R

$Q \leftarrow I_n$
$R \leftarrow A$
Pour k allant de 1 à $n-1$:
    Pour tous les j dans $1 \dots \lceil k/2 \rceil$:
        Rotate$(n-k+2j-2, j)$
Pour k allant de n à $2n-3$:
    Pour tous les j dans $k-n+2 \dots \lceil k/2 \rceil$:
        Rotate$(2j-(k-n+2), j)$

Le sous-programme **Rotate**(i, j) est défini comme suit:

Rotate
Entrée: i et j entiers
$\lambda \leftarrow (R_{i,j}^2 + R_{i+1,j}^2)^{1/2}$
$c \leftarrow R_{i,j}/\lambda$
$s \leftarrow R_{i+1,j}/\lambda$
Remplacer les lignes (R_i, R_{i+1}) par $(cR_i + sR_{i+1}, -sR_i + cR_{i+1})$.
Remplacer les lignes (Q_i, Q_{i+1}) par $(cQ_i + sQ_{i+1}, -sQ_i + cQ_{i+1})$.

Rotate utilise n processeurs, et termine en temps constant. Donc la boucle indicée par j utilise n^2 processeurs et un temps constant, et l'orthogonalisation au complet n^2 processeurs et un temps linéaire $O(n)$.

Théorème 3.2 *Si l'algorithme* **Ortho** *décrit précédemment travaille en arithmétique flottante avec une précision ϵ, et si $\hat{Q}$ et $\hat{R}$ désigne le résultat de ce calcul approché, alors il existe une matrice d'erreur E telle que:*

- $\hat{Q}(A+E) = \hat{R}$
- $\|E\|_F \leq \alpha(n)\epsilon\|A\|_F$
- $\|Q - \hat{Q}\|_F \leq \alpha(n)\epsilon$
- $\|R - \hat{R}\| \leq \alpha(n)\epsilon\|R\|_F = \alpha(n)\epsilon\|A\|_F$

Où $\|.\|_F$ désigne la norme de Frobenius, et $\alpha(n)$ est un $o(n^2)$.

Ce type de résultat, assez général, qui s'applique également à l'orthogonalisation de Householder, est prouvé dans [42].

Afin de mieux illustrer cette technique d'orthogonalisation, voici un exemple en dimension 10 de l'ordre dans lequel s'effectue l'annulation des coefficients de R pendant l'algorithme:

$$\begin{pmatrix}
* & * & * & * & * & * & * & * & * & * \\
9 & * & * & * & * & * & * & * & * & * \\
8 & 10 & * & * & * & * & * & * & * & * \\
7 & 9 & 11 & * & * & * & * & * & * & * \\
6 & 8 & 10 & 12 & * & * & * & * & * & * \\
5 & 7 & 9 & 11 & 13 & * & * & * & * & * \\
4 & 6 & 8 & 10 & 12 & 14 & * & * & * & * \\
3 & 5 & 7 & 9 & 11 & 13 & 15 & * & * & * \\
2 & 4 & 6 & 8 & 10 & 12 & 14 & 16 & * & * \\
1 & 3 & 5 & 7 & 9 & 11 & 13 & 15 & 17 & *
\end{pmatrix}$$

L'algorithme de Givens, tel que nous venons de le décrire permet d'orthogonaliser des matrices carrées $n \times n$. Il est facile de le modifier pour obtenir un algorithme analogue pour des matrices $m \times n$, c'est à dire, des matrices de n vecteurs dans un espace de dimension m. Dans l'algorithme modifié, Q devient une matrice $n \times m$, et R reste une matrice triangulaire supérieure, bien évidemment dans la formule $QA = R$, R est carrée $n \times n$, alors que dans l'algorithme modifié R est $m \times n$, cependant comme en sortie de l'algorithme, les $m - n$ dernières lignes de R seront nulles, cela revient au même. L'algorithme **Ortho** devient alors:

Ortho
Entrée: A
Sortie: Q et R
$Q \leftarrow I_{n,m}$ (la matrice formée des n premières lignes de I_m)
$R \leftarrow A$
Pour k allant de 1 à $m - 1$:
    Pour tous les j dans $1 \dots \min(\lceil k/2 \rceil, n)$:
        $Rotate(m - k + 2j - 2, j)$
Pour k allant de m à $\min(n + m - 2, 2m - 3)$:
    Pour tous les j dans $k - m + 2 \dots \min(\lceil k/2 \rceil, n)$:
        $Rotate(2j - k + \min(m - 1, n - 2), j)$

Le sous-algorithme **Rotate**, lui, reste inchangé. De plus, si $m < 2n$, comme nous l'avons supposé, le théorème 3.2 reste vrai.

3.3 Retour à la réduction parallèle

Puisque l'orthogonalisation est très efficace en parallèle, et que plusieurs échanges en parallèle modifient beaucoup B^*, il est préférable de recalculer B^* entièrement plutôt que de chercher à le mettre à jour d'un tour sur l'autre. De plus, cette approche évite que les erreurs d'arrondi successives ne dégradent la précision de B^*. Ainsi on obtient la structure d'algorithme suivante:

- Répéter:
    - Phase d'Orthogonalisation
    - Phase de Translation
    - Phase d'échange
- Jusqu'à un round sans échange

Les phases d'orthogonalisation et d'échanges ne posent pas de problèmes, elles ont déjà été détaillées dans ce qui précéde. En revanche, il reste à définir la phase de translation. A priori, si l'on cherche durant cette phase à ramener tous les éléments non diagonaux de M entre $-1/2$ et $1/2$ (ou plutôt entre -0.55 et 0.55 pour tenir compte des erreurs d'arrondi), deux approches sont possibles, on peut soit aller de gauche à droite:

LRtranslation

Pour i allant de 2 à n:
  Pour j descendant de $i-1$ à 1:
    $\mu = \frac{(b_i|b_j^*)}{\|b_j^*\|^2}$
    $r = \lfloor \mu + 1/2 \rfloor$
    $b_i = b_i - r b_j$

Soit de droite à gauche:

RLtranslation
Pour i descendant de n à 2:
  Pour j descendant de $i-1$ à 1:
    $\mu = \frac{(b_i|b_j^*)}{\|b_j^*\|^2}$
    $r = \lfloor \mu + 1/2 \rfloor$
    $b_i = b_i - r b_j$

Malheureusement, la première méthode est intrinsèquement séquentielle et la seconde est numériquement instable. Puisqu'il ne semble pas possible d'effectuer la phase de translation efficacement si l'on veut ramener tous les éléments non diagonaux de M en dessous de 0.55, nous allons dans la section suivante décrire une autre stratégie de translation, moins contraignante, mais suffisante pour les besoins de LLL. En effet, le rôle principal de la réduction en longueur complète est d'empêcher l'apparition de nombres trop grands pendant le déroulement de l'algorithme.

3.4 L'algorithme

A la fin de la section précédente, nous avons posé le problème de trouver un stratégie de translation parallèlisable. En fait, ramener les coefficients de M en dessous de 0.55 permet de majorer les b_i pendant toute l'exécution de l'algorithme, et ainsi de permettre l'obtention d'un algorithme stable. Nous allons décrire une nouvelle approche, qui permet également une telle majoration, et qui de plus a le mérite d'être parallèlisable. L'idée principale consiste à associer à chaque vecteur b_i une base du sous-réseau L_{i-1}, formée de $i-1$ vecteurs, $b_{i,1}$, $b_{i,2}$, ..., $b_{i,i-1}$. Cette base locale, s'orthogonalise en $b_{i,1}^*$, ..., $b_{i,i-1}^*$. De plus, si on la complète par b_i en une base de L_i, pour obtenir la base orthogonale associée, il suffit de rajouter b_i^*. Cette propriété est très importante, et provient du fait que b_i^* ne dépend que de b_i et de L_{i-1}, mais pas de la base de L_{i-1} que l'on choisit. Afin de pouvoir borner les b_i, nous allons imposer la condition suivante:

$$\forall i > j > k > 0 : \left| \frac{(b_{i,j}|b_{i,k}^*)}{\|b_{i,k}^*\|^2} \right| \leq 0.55 \tag{3.1}$$

Cette condition est analogue à la condition sur M que souhaitions initialement, mais elle ne concerne plus que les bases locales. Ainsi, nous avons remplacé une condition globale par une condition locale, qui va nous permettre d'obtenir un algorithme parallèle.

Plus précisement, il est facile d'initialiser les bases locales, pour cela, il suffit d'effectuer une phase initiale de translation de gauche à droite, pour ramener les coefficients de M en dessous de 0.55 en valeur absolue. Cette phase n'ayant lieu qu'une fois, elle ne pénalise pas trop le temps de calcul. Après cela, les bases locales initiales sont obtenues comme segment initiaux de la bases B, et de même, leurs orthogonalisées sont

des segments initiaux de B^*. Il nous reste à comprendre comment mettre à jour ces bases locales quand B subit un échange. En fait, c'est assez simple, si l'on échange les vecteurs i et $i+1$, il faut oublier la base locale de $i+1$ et tout reconstruire avec la base locale de i. Dans B_i, seul $b_{i,i}$ va changer, en revanche, B_{i+1} sera obtenu en collant B_i et le nouveau vecteur en position $i+1$. Bien sûr, il faut aussi translater les vecteur $b_{i,i}$ et $b_{i+1,i+1}$ pour ramener leur coefficients en dessous de 0.55, mais cela ne pose pas de problèmes particulier. Enfin, il faut mettre à jour B_i^* et B_{i+1}^*, ce qui est facile, à condition de remarquer que $b_{i,i}^* = b_i^*$ et $b_{i+1,i+1}^* = b_{i+1}^*$.

Voici maintenant notre algorithme de réduction, il utilise 3 sous-algorithmes **Star**, **Exchange** et **Reduce** dont la description suit l'algorithme principal. Nous justifierons cet algorithme de réduction dans la section suivante.

Réduction Parallèle
Appeler $Star(L)$
Pour i allant de 2 à n:
  Pour j descendant de $i-1$ à 1:
    $b_{j,i} \leftarrow b_j$
    $b_{j,i}^* \leftarrow b_j^*$
    $b_i \leftarrow b_i - \left\lfloor \frac{(b_i|b_{j,i}^*)}{\|b_{j,i}^*\|^2} + 1/2 \right\rfloor b_{j,i}$

Boucle Principale Répéter:

1. Pour tous les i dans $2 \dots n$:
   $\mu_i \leftarrow \frac{(b_i|b_{i-1}^*)}{\|b_{i-1}^*\|}^2$
   $r_i \leftarrow \lfloor \mu_i + 1/2 \rfloor$
   $E_i \leftarrow \left[\|b_{i-1}^*\|^2 \geq 1.03(\|b_i^*\|^2 + (\mu_i - r_i)^2 * \|b_{i-1}^*\|^2)\right]$
2. Pour i allant de 2 à $n-1$
   Si E_i est vrai, alors $E_{i+1} \leftarrow Faux$
3. Pour tous les i dans $2 \dots n$ avec E_i vrai:
   $b_i \leftarrow b_i - r_i b_{i-1}$
   Appeler $Exchange(i)$
4. Appeler $Star(L)$
5. Pour tous les i dans $2 \dots n$ avec E_i vrai:
   $b_{i-1,i}^* \leftarrow b_{i-1}^*$
   Appeler $Reduce(i)$

Jusqu'à ce qu'un tour sans échange se produise.
Pour i allant de 2 à n:
  Pour j descendant de $i-1$ à 1:
    $b_i \leftarrow b_i - \left\lfloor \frac{(b_i|b_j^*)}{\|b_j^*\|^2} + 1/2 \right\rfloor b_j$

Star
Appeler $Ortho(L)$
$L^* \leftarrow {}^tQ$
Pour tous les i dans $1 \dots n$:

$b_i^* \leftarrow R_{i,i} b_i^*$

Exchange
$j \leftarrow 1$
Tant que $j \leq i-2$:
    $b_{j,i} \leftarrow b_{j,i-1}$ (communication de la colonne $i-1$ à i)
    $b^*_{j,i} \leftarrow b^*_{j,i-1}$ (communication de $i-1$ à i)
    Incrémenter j
$b_{i,i-1} \leftarrow b_i$
Echanger b_i et b_{i-1} (communication de $i-1$ à i et de i à $i-1$)
Appeler $Reduce(i-1)$

Reduce
$j \leftarrow i-1$
Tant que $j \geq 1$:
    $r \leftarrow \left\lfloor \frac{(b_i | b^*_{i,j})}{\|b^*_{i,j}\|^2} + 1/2 \right\rfloor$.
    $b_i \leftarrow b_i - r b_{i,j}$
    $j \leftarrow j-1$

3.5 Preuve de l'algorithme

Afin de prouver le théorème 3.1, il nous suffit de montrer que l'algorithme que nous venons de décrire fournit lorsqu'il opère sur des nombres en précision relative ϵ, telle que $1/\epsilon \geq 4n^5 A^3 1.6^n/\gamma^3$ une base réduite au sens de 1.05-Lovász en $O(n^3 A/E(B))$ opérations arithmétique.

Commençons par montrer que le sous-algorithme **Star**, étant donnée une matrice entière B calcule une valeur approchée $\hat{B}^*$ de la matrice B^* vérifiant:

$$\|B^* - \hat{B}^*\| \leq 2\alpha(n)\epsilon \|B\|_F. \tag{3.2}$$

Le calcul se fait en deux étapes, tout d'abord, on calcule R et Q, puis on les multiplie, pour connaître l'erreur sur B^*, il nous suffit donc d'ajouter les erreurs sur R, Q, et l'erreur commise durant la multiplication. L'équation 3.2 en découle immédiatement.

Montrons maintenant que la phase de précalcul calcule des bases locales convenables. Il est facile de voir que cela revient à vérifier qu'étant données B et B^* l'algorithme **LRtranslation** réduit B en longueur et fournit en sortie une matrice B telle que dans sa décomposition $B = B^* M$, les coefficients au-dessus de la diagonale de M soient inférieurs en valeurs absolues à 0.55. Pour le prouver, nous allons montrer par récurrence sur i, que tous les b_i obtenus après réduction sont en norme au carré inférieurs à nA, et que les $\mu_{i,j}$ correspondant sont en valeur absolue inférieurs à 0.55. Supposons donc que b_1, b_2, ..., b_{i-1} ont déjà été réduits, et examinons la réduction de b_i. Initialement,

$$b_i = b_i^* + \mu i - 1 b^*_{i-1} + \cdots + \mu_1 b_1^*,$$

les coefficients μ de cette décomposition se calculant par la formule $\mu_j = \frac{(b_i|b_j^*)}{\|b_j^*\|^2}$, ils sont donc inférieurs à A/γ. Lors de translation de b_i parallèlement à b_j, μ_j est ramené en dessous de 0.55, en revanche les μ qui le suivent peuvent éventuellement augmenter. Heureusement, b_j ayant déjà été réduit, cette augmentation de valeur absolue se limite à $0.55|\mu_j|$. A chaque translation, le plus grand des μ en valeur absolue augmente donc, au plus, d'un facteur 1.55, dans toute la réduction en longueur, les μ sont donc majorés par $1.55^n A/\gamma$, le calcul peut donc aboutir sans erreur car $1.55^n A\epsilon/\gamma$ reste inférieur à 0.01, et donc $r = \lfloor \mu + 1/2 \rfloor$ est donc assez précis pour que la translation de b_i parallèlement à b_j ramène effectivement μ_j en dessous de 0.55.

Nous allons examiner le déroulement de la boucle principale, pour ce faire, commençons par faire une hypothèse de récurrence sur les bases locales. Plus précisément, supposons que:

$$\forall i \geq j > k \geq 1 : \left| \frac{(b_{i,j}|b_{i,k}^*)}{\|b_{i,k}^*\|^2} \right| < 0.55.$$

Cette hypothèse, nous venons de le montrer, est vraie à l'entrée du premier tour, nous pouvons donc initier la récurrence. D'autre part, nous vérifierons facilement que pendant le déroulement de l'algorithme $\min_{i,j} \|b_{i,j}^*\|^2$ ne peut que croître et que $\max_{i,j} \|b_{i,j}^*\|^2$ ne peut que décroître. Les $b_{i,j}^*$ restent donc pendant tout l'algorithme avec une norme au carré comprise entre γ et A. La décomposition de b_i en combinaison linéaire de $b_{i,j}^*$, permet d'en déduire que b_i reste en norme au carré inférieur à nA. Ces quelques faits établis, nous pouvons passer à l'analyse détaillée de la boucle principale. Dans l'étape 1 de cette boucle, l'algorithme teste si l'échange de b_i et b_{i-1} peut permettre de gagner un facteur 1.03 sur D ou non. Ce test est bien sûr effectué de manière approchée. Il est facile de majorer l'erreur relative sur

$$\frac{\|b_i^*\|^2 + (\mu_i - r_i)\|b_{i-1}^*\|^2}{\|b_{i-1}^*\|^2}$$

par

$$\frac{3A^2}{\gamma^2}\epsilon \leq \frac{3\gamma}{4n^5 A 1.6^n} < 0.02$$

Par conséquent, si le test est positif, l'échange de b_i et b_{i-1} permet de gagner sur la valeur exacte de D un facteur supérieur à 1.01; et si le test est négatif, ce même échange ne peut pas faire gagner un facteur meilleur que 1.05. Dans l'étape 2, on élimine les échanges potentiels qui interfèrent avec leur prédécesseur. Cette étape ne manipule que des booléens, elle ne peut donc pas être numériquement instable. De plus, elle ne peut pas éliminer tous les échanges potentiels, l'algorithme ne peut donc s'arrêter que quand tous les tests d'échanges sont négatifs. C'est pourquoi, une fois la réduction en longueur finale effectué, on obtient une base réduite au sens de 1.05-Lovász.

Dans l'étape 3, les échanges finalement retenus sont effectués, en premier lieu, les b_i choisis sont translatés par rapport aux b_{i-1}, pour que la décroissance de D ait bien lieu comme prévu. Cette translation ne pose pas de problème de stabilité car tous les nombres utilisés sont entiers, connus exactement, et bien plus petits que $1/\epsilon$. Ensuite, cette troisième étape appelle le sous-programme **Exchange**, dont le but est d'échanger b_{i-1} et b_i et de créer les nouvelles bases locales en $i-1$ et i. Pour créer les nouvelles bases locales, on se base sur l'ancienne base locale de $i-1$, en effet elle constitue bien

une base de L_{i-2}, aussi bien avant qu'après les échanges. Par simple recopie, on obtient donc, tous les $b_{i-1,j}$ et $b_{i,j}$, pour $j < i-1$, ainsi que les b^* associés. De plus, on ajoute à la base locale en i le nouveau b_{i-1}, en position $b_{i,i-1}$.

Pour compléter la base locale en $i-1$, il faut encore réduire le nouveau b_{i-1} en longueur par rapport à sa base locale. Ceci est fait par **Reduce**, qui est une adaptation de **LRtranslation**, et donc numériquement stable. L'étape 4 sert simplement à calculer les nouveaux b_i^*, ce qui permet ensuite dans l'étape 5 de terminer la mise à jour des bases locales. On commence par compléter la base locale en i, en posant $b^*_{i,i-1} = b^*_{i-1}$. Puis on appelle de nouveau **Reduce**, cette fois sur la base locale en i, afin de réduire b_i en longueur. Ainsi, lorsque l'algorithme aborde le tour suivant de la boucle principale, les hypothèses de récurrence sont satisfaites.

Enfin, une fois la boucle terminée, on utilise **LRtranslation**, pour entièrement réduire la base en longueur, comme une base réduite par LLL doit l'être.

Nous avons donc prouver que cet algorithme lorsqu'il termine, produit une base réduite au sens de 1.05-Lovász. Majorons maintenant le temps de calcul. En premier lieu, chaque échange effectué gagne un facteur au moins 1.01 sur D. Par conséquent, le nombre total d'échanges est majoré par $\log_{1.01} D_{\text{init}}$. Si le nombre moyen d'échanges par tour de la réduction de B est $E(B)$, nécessairement, le nombre de tours de cette réduction est majoré par $\log_{1.01} D_{\text{init}}/E(B)$. Le nombre d'opérations arithmétique pour un tour de la boucle est $O(n)$, en effet, **Star** coûte $O(n)$, **Reduce** et **Exchange** aussi. Tout le reste coûte $O(1)$. Le nombre d'opérations arithmétiques est donc majoré par $O(n \log_{1.01} D_{\text{init}}/E(B))$, c'est-à-dire par $O(n^3 A/E(B))$. De plus, la taille des nombres intervenant est $1/\epsilon = O(n + \log(A) + \log(1/\gamma)$. D'où le théorème 3.1.

Deuxième partie

Programmation et applications

Chapitre 4

Réalisation d'un programme de réduction de réseau

Lorsque l'on travaille avec un algorithme comme LLL, il est très important de pouvoir tester les applications de cet algorithme. Pour cela, il faut absolument disposer d'une version rapide et efficace de l'algorithme. La programmation de LLL a donc été l'un des volets de cette thèse.

Après avoir écrit plusieurs versions de ce programme, nous avons abouti à la version actuelle qui comporte deux algorithmes de réductions LLL et BKZ, et est inspirée de l'article de Schnorr et Euchner. Elle est basée sur le programme BigNum de gestion des grands entiers et est écrite en langage C. Cependant, la taille des nombres pouvant apparaître dans les réseaux à réduire est limitée par la taille maximale des flottants représentables par la machine. En effet, le programme calcule la norme des vecteurs de base du réseau de manière approchée, et si ce calcul ne peut aboutir il lui est impossible de continuer. Bien entendu, nous aurions pu gérer des flottants avec un grand exposant, mais le ralentissement qui en découle nous en a dissuadé.

Dans tous les chapitres d'expérimentation qui suivent, les temps de calcul fournis sont ceux obtenus sur Sun 4/20 avec ce programme.

4.1 Mode d'emploi du programme

Ce programme de réduction a été écrit pour la système d'exploitation UNIX, et il accepte un certain nombre d'option sur la ligne de commande, en voici la liste:

- `-lll` Cette option demande au programme d'utiliser LLL comme algorithme de réduction, c'est la valeur par défaut. Cette option est incompatible avec `-bkz`.
- `-bkz` Cette option demande au programme d'utiliser BKZ comme algorithme de réduction. Cette option est incompatible avec `-lll`.
- `-f` *filename* Cette option demande au programme de lire le réseau à réduire dans le fichier *filename*. Par défaut, le réseau est entré au clavier. Une seule option `-f` est autorisé par appel au programme.
- `-format` Cette option demande au programme de formatter la sortie de la base réduite sous une forme réutilisable par l'option `-f`. Par défaut, la sortie des données

est présentée sous forme de matrice colonne, afin d'améliorer la lisibilité.

- `-autosave` Cette option demande au programme de sauvegarder le résultat partiel, en cours de réduction, tous les 1000 tours.
- `-autosave=`*num* Cette option demande au programme de sauvegarder le résultat partiel, en cours de réduction, tous les *num* tours.

Une fois lancé, ce programme demande, ou va chercher dans le fichier d'entrée, les informations suivantes:

- Taille des nombres. Comme le programme effectue les calculs en place, il demande au début de l'éxécution un majorant de l'espace mémoire nécessaire au stockage de tous les entiers pouvant apparaître pendant l'algorithme. Cette taille est donné en nombre de mots de 32 bits nécessaires au stockage de la mantisse des entiers en question.
- Dimension de l'espace.
- Nombre de vecteurs. Ce nombre peut être différent la dimension de l'espace, en effet, s'il est inférieur, cela signifie que le réseau est plongé dans un espace de dimension supérieur, et s'il est supérieur, cela prouve simplement que l'utilisateur fournit un système générateur du réseau plutôt qu'une base. Cela n'empêche pas le bon déroulement du programme qui éliminera les vecteurs inutiles pendant la réduction.
- Coordonnées des vecteurs. L'utilisation doit d'abord fournir toutes les coordonnées du premier vecteur, puis celle du second, ...
- Taille des blocs. Cette donnée n'est demandée que lors de l'utilisation de l'option `-bkz`, et elle correspond à la taille des blocs à réduire par Korkine-Zolotarev.

Chapitre 5

Expérimentation, réseaux aléatoires, sac-à-dos aléatoires

Afin de donner une idée des possibilités du programme que nous venons de présenter, nous avons réalisé plusieurs séries de tests portant d'une part sur des réseaux aléatoires, d'autre part sur les réseaux obtenus à partir de sac-à-dos aléatoires par les trois méthodes que nous avons décrites dans la première partie. A l'aide des diverses expérimentations que nous avons effectuées, nous allons chercher à comparer les temps de calcul de LLL et de l'algorithme de Korkine-Zolotarev par blocs sur divers types de réseau. Nous essaierons également pour chaque cas étudié de donner un polynôme d'interpolation du temps de calcul. De plus, nous allons comparer les valeurs pratiques des deux méthodes de résolution de sac-à-dos en densité moyenne, autant du point de vue des temps de calcul que de celui des taux de réussite sur des sac-à-dos aléatoires.

5.1 Les réseaux aléatoires

Dans cette section, nous allons considérer des réseaux obtenus en remplissant une matrice de la dimension voulue par des nombres pseudo-aléatoires de la taille souhaitée. Par analogie avec le cas des sacs-à-dos, nous appelerons densité le quotient de la dimension par la taille. Nous avons représenté sur une même figure les temps de calcul par LLL pour des dimensions allant de 10 à 130 et des densités allant de 0.6 à 1. Pour chaque couple dimension/densité, nous avons calculé le temps moyen de réduction sur 10 matrices distinctes. On peut constater que le temps de calcul semble indépendant de la densité. De plus, le temps de calcul est de degré 4.2 environ en fonction de la dimension de la matrice à réduire.

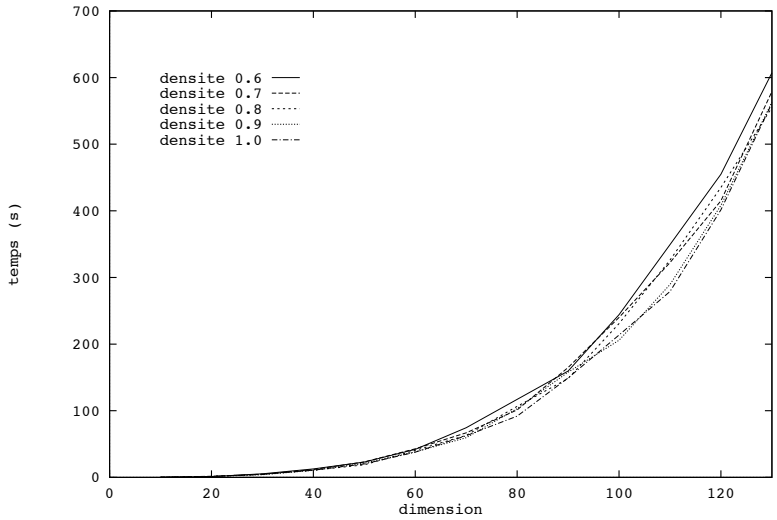


FIG. 5.1 - Temps de calcul par LLL, densité 0.6 à 1.0

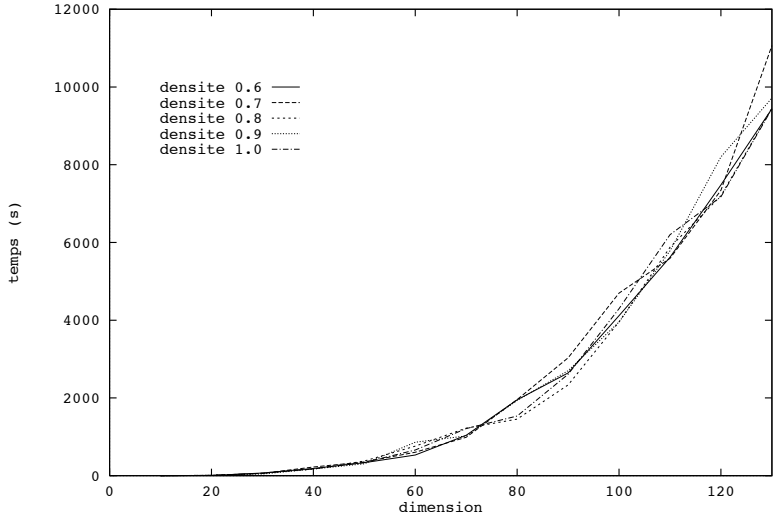


FIG. 5.2 - Temps de calcul par BKZ 10, densité 0.6 à 1.0

5.2 Les sacs-à-dos

Pour les résolutions des sacs-à-dos, nous avons à notre disposition les méthodes de Lagarias-Odlyzko, Coster-Odlyzko-Lamacchia-Schnorr et Joux-Stern. Pour abréger, nous les désignerons par LO, COLS et JS. Nous avons établi un grand nombre d'expérimentations avec ces trois méthodes couplées soit à LLL, soit à BKZ avec des blocs de taille 10 ou 20. Nous nous sommes intéressés aux dimensions de 10 à 100, et aux densités 0.6, 0.7, 0.8, 0.9 et 1.0. Nous avons évalué le temps de calcul moyen et le taux de réussite moyen de chaque méthode sur 10 réseaux par couple dimension/densité. Dans certains cas, nous nous sommes arrêtés avant la dimension 100, après avoir constaté que le taux de réussite était devenu nul. Nous avons remarqué que les méthodes COLS et JS ne réussissent pas forcément sur les mêmes sacs-à-dos, nous avons donc aussi évalué le taux de réussite cumulé de ces deux méthodes. Voici l'ensemble de nos résultats. Les temps de calcul sont donnés sous forme graphique, et pour les taux de réussite, nous donnons pour chaque cas, la valeur de la dimension de coupure, dernière dimension pour laquelle le taux de réussite est supérieur ou égal à 1/2.

		Dimension de coupure			
Densité	Algorithme	LO	JS	COLS	Cumulé
0.6	LLL	20	40	40	40
0.6	BKZ10	20	70	70	80
0.6	BKZ20	30	90	80	90
0.7	LLL	10	30	30	40
0.7	BKZ10	10	50	50	60
0.7	BKZ20	< 20	70	70	70
0.8	LLL	< 10	30	30	30
0.8	BKZ10	< 10	50	40	50
0.8	BKZ20	< 20	60	60	60
0.9	LLL	< 10	30	30	30
0.9	BKZ10	< 10	40	40	40
0.9	BKZ20	< 20	50	40	60
1.0	LLL	< 10	20	20	20
1.0	BKZ10	< 10	30	30	40
1.0	BKZ20	< 20	50	40	50

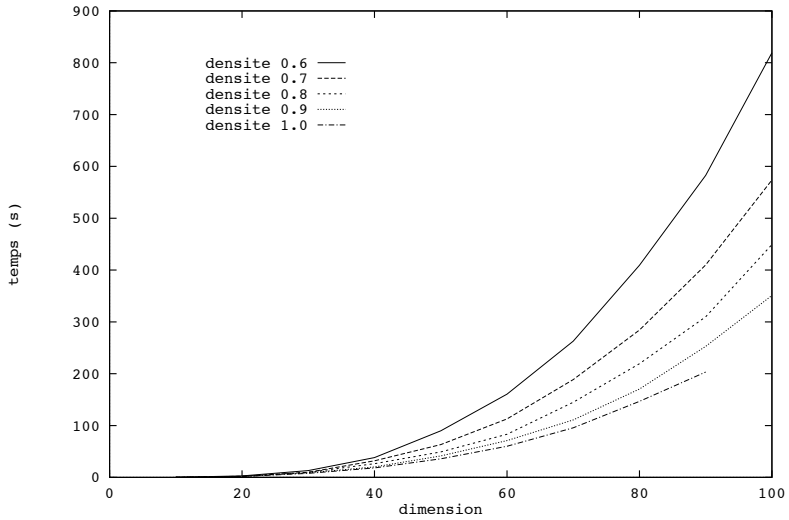


FIG. 5.3 - Temps de calcul par LO et LLL, densité 0.6 à 1.0

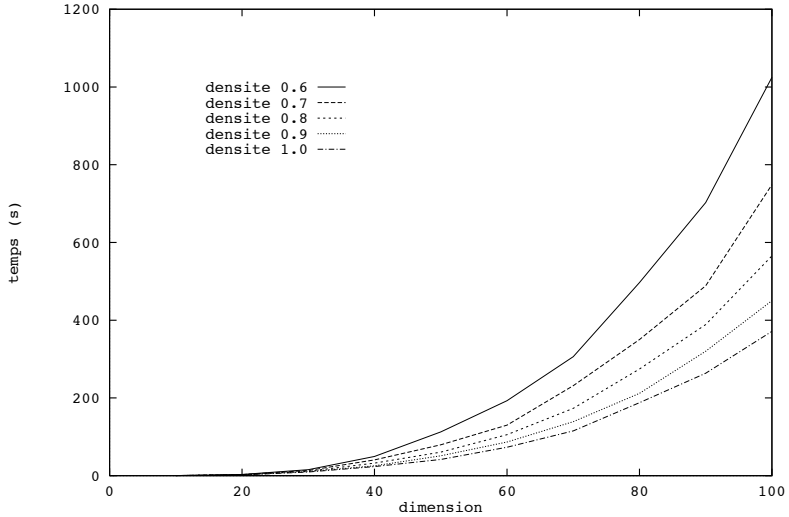


FIG. 5.4 - Temps de calcul par JS et LLL, densité 0.6 à 1.0

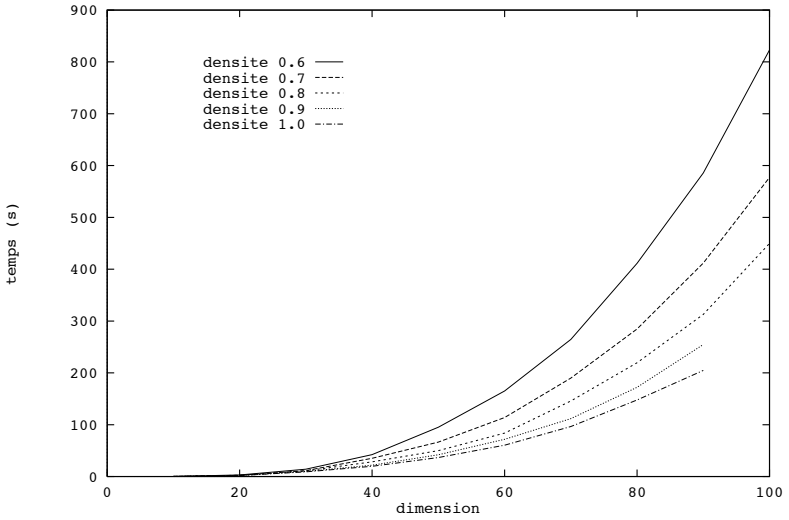


FIG. 5.5 - Temps de calcul par COLS et LLL, densité 0.6 à 1.0

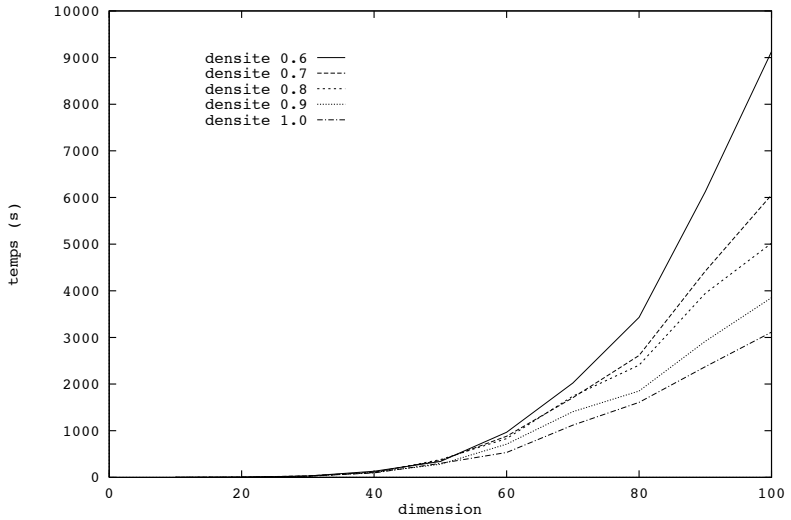


FIG. 5.6 - Temps de calcul par LO et BKZ10, densité 0.6 à 1.0

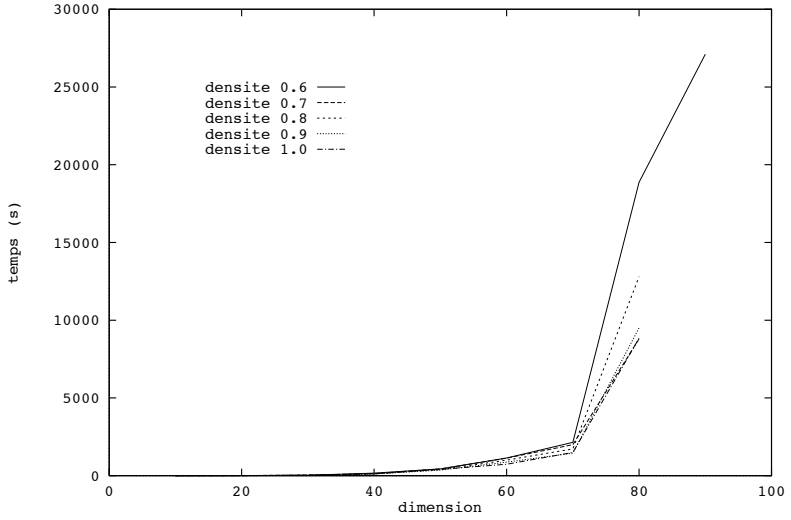


FIG. 5.7 - Temps de calcul par JS et BKZ10, densité 0.6 à 1.0

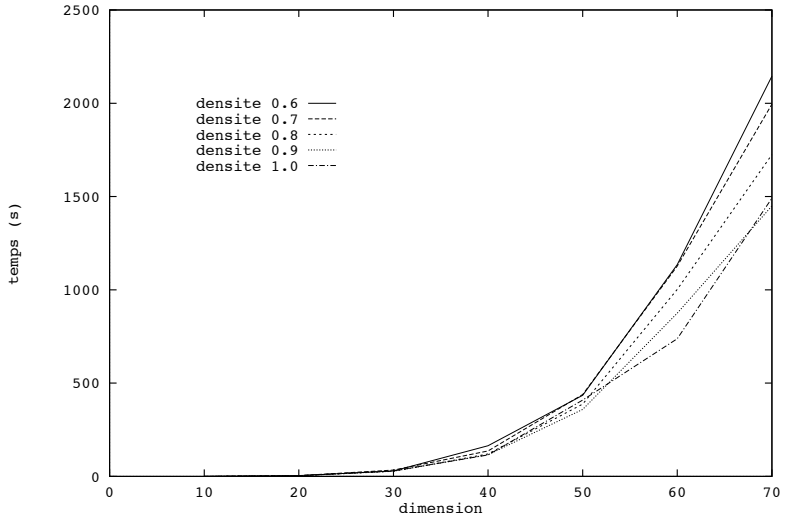


FIG. 5.8 - Temps de calcul par JS et BKZ10, densité 0.6 à 1.0, limité à la dimension de coupure

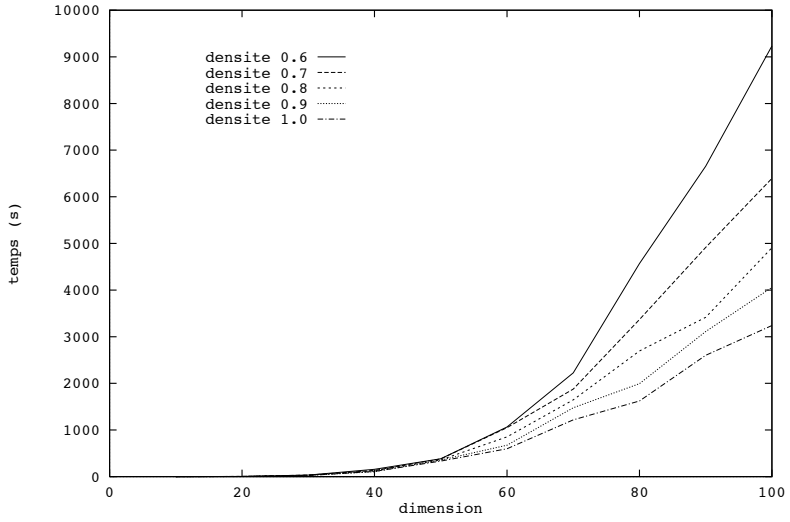


FIG. 5.9 - Temps de calcul par COLS et BKZ10, densité 0.6 à 1.0

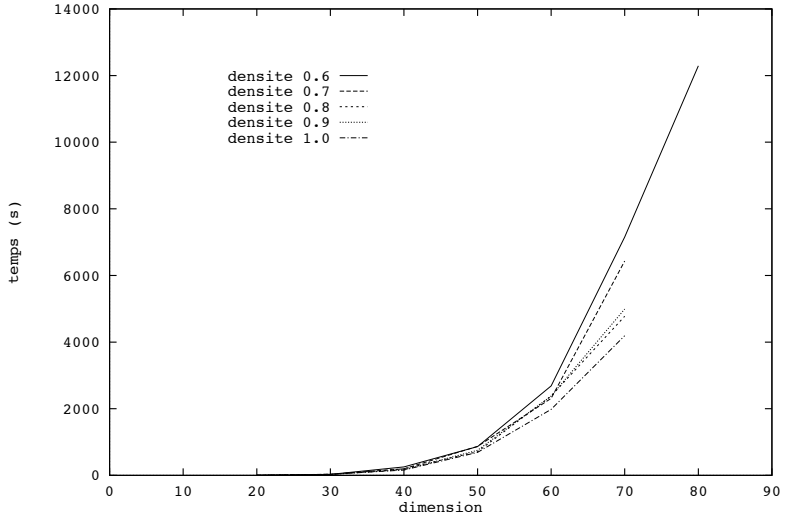


FIG. 5.10 - Temps de calcul par LO et BKZ20, densité 0.6 à 1.0

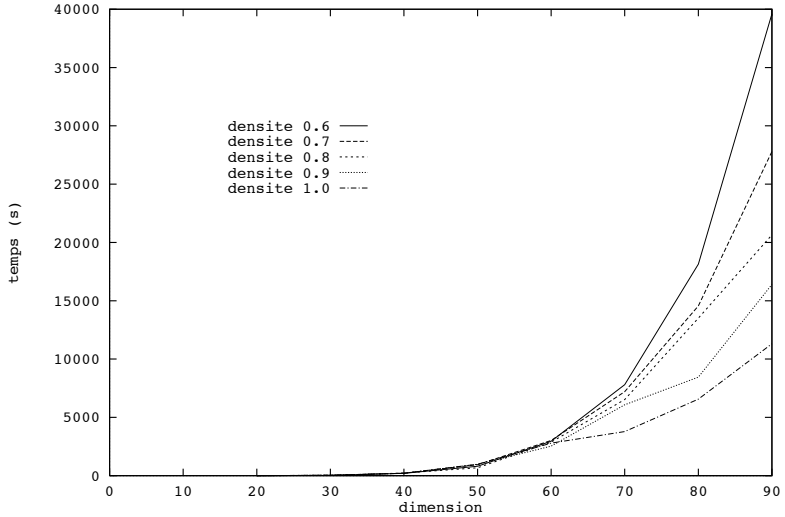


FIG. 5.11 - Temps de calcul par JS et BKZ20, densité 0.6 à 1.0

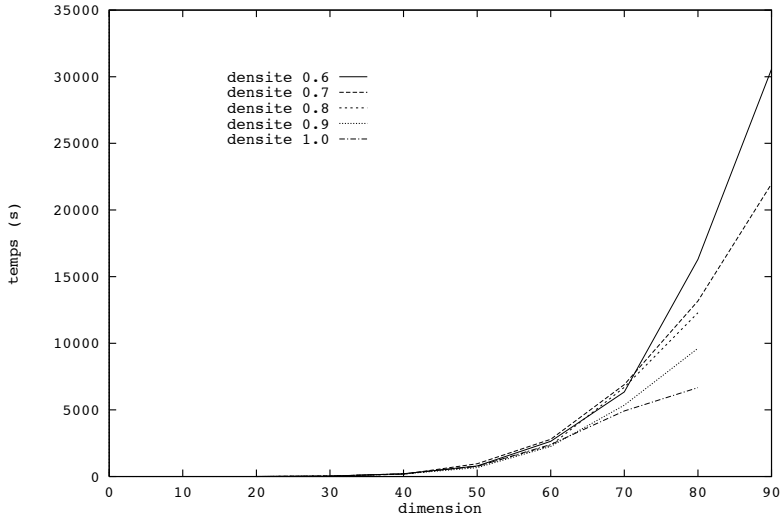


FIG. 5.12 - Temps de calcul par COLS et BKZ20, densité 0.6 à 1.0

Chapitre 6

Calculs de dessins d'enfant

6.1 Dessins d'enfants, Introduction

Dans ce chapitre, qui résulte d'une collaboration avec Jean-Marc Couveignes, on se propose d'illustrer la puissance de LLL en l'appliquant à la résolution d'une classe particulière de systèmes algébriques. Ces systèmes apparaissent dans la théorie des revêtements de la sphère, ramifiés seulement au dessus de trois points rationnels. Ces revêtements aussi appelés *dessins d'enfant* ont été étudiés sous diverses formes depuis le siècle dernier. Dans sa désormais célèbre *esquisse d'un programme* [14], Alexandre Grothendieck fut le premier à en souligner l'importance cruciale. En effet, ces revêtements établissent un lien entre topologie algébrique et arithmétique des corps de nombres. Selon le théorème de Belyi, en effet, sur toute courbe algébrique rationnelle (i.e. définie sur $\bar{\mathbf{Q}}$) il est possible de définir une application rationnelle à valeurs dans $\mathbf{P}_1(\bar{\mathbf{Q}})$ et ramifiée seulement au dessus de 0, 1 et ∞, c'est à dire, à peu de choses près, un dessin d'enfant. Comme on sait par ailleurs que toute application rationnelle dont les valeurs singulières sont dans $\bar{\mathbf{Q}}$ est elle même définie sur $\bar{\mathbf{Q}}$, on entrevoit une correspondance entre revêtements topologiques propres et finis de la sphère moins trois points d'une part, et corps de nombres d'autre part. Cette correspondance permet de "voir" l'action du groupe absolu de Galois de $\mathbf{Q}$ sur des revêtements. L'étude combinatoire de ces revêtements donne alors des renseignement sur le groupe de Galois.

Par fonction de Belyi on désigne une application rationnelle Π d'une courbe projective non singulière $\mathcal{C}$, à valeurs dans $\mathbf{P_1}(\mathbf{C})$ et ramifiée seulement au dessus de 0, 1 et ∞. On demande que Π et $\mathcal{C}$ soient définies sur $\bar{\mathbf{Q}}$. Un dessin d'enfant est un couple $(\mathcal{C}, \Pi)$ défini à $\bar{\mathbf{Q}}$-isomorphismes près.

Lorsque $\mathcal{C} = \mathbf{P_1}(\mathbf{C})$, l'application Π est donc définie à une homographie près(et en général à un automorphisme de $\mathcal{C}$ près).

Si on demande de plus que l'indice de ramification en chacun des points au dessus de 1 soit exactement 2, la fonction de Belyi et le dessin sont dits propres.

Alors, on peut considérer l'image réciproque du segment réel $[0, 1]$ dans $\mathbf{P_1}(\mathbf{C})$. On obtient un graphe connexe dont les sommets correspondent aux zéros de Π avec pour multiplicités le nombre de segments arrivant au sommet. Sur chaque segment reliant deux sommets la fonction Π prend une et une seule fois la valeur 1. Le graphe délimite des faces (cellules) au milieu desquelles se trouve un pôle dont la multiplicité est le nombre de segments qui bordent la face.

Dans le cas des arbres, on placera toujours l'unique face à l'infini de façon à obtenir un polynôme.

Comme on l'a vu, on adopte la convention suivante: les faces sont des pôles, les sommets des zéros et les cotés des uns. Notons que ce n'est pas la même convention que celle choisie dans [37].

On appelle α_i ces sommets et ν_i leurs multiplicités pour $1 \leq i \leq N$ où N est le nombre de sommets et $d = \sum \nu_i$ le degré de la fonction de Belyi.

On obtient alors une identité du type

$$\prod_{i \in I} (X - \alpha_i)^{\nu_i} + \lambda = Q^2(X) \tag{6.1}$$

où I est l'ensemble des indices correspondant aux sommets.

On notera que le polynôme

$$\Pi = \prod_{i \in I} (X - \alpha_i)^{\nu_i}$$

se factorise en

$$\Pi^+ = Q - \sqrt{\lambda} = \prod_{i \in I^+} (X - \alpha_i)^{\nu_i} \quad \text{et} \quad \Pi^- = Q + \sqrt{\lambda} = \prod_{i \in I^-} (X - \alpha_i)^{\nu_i}$$

Où on définit I^+ comme l'ensemble des indices correspondant aux sommets α_i tels que $Q(\alpha_i) = \sqrt{\lambda}$ (et que, par exemple, on colorie en rouge) et I^- l'ensemble des indices correspondant aux sommets α_i tels que $Q(\alpha_i) = -\sqrt{\lambda}$ (et que, par exemple, on colorie en bleu.)

Alors on a

$$\Pi = \Pi^+ \Pi^- = (Q - \sqrt{\lambda})(Q + \sqrt{\lambda})$$

Si on dérive l'identité 6.1 on obtient,

$$\Pi(\Sigma^+ + \Sigma^-) = 2QQ' \tag{6.2}$$

avec

$$\Sigma^+ = \sum_{i \in I^+} \frac{\nu_i}{X - \alpha_i} \quad , \quad \Sigma^- = \sum_{i \in I^-} \frac{\nu_i}{X - \alpha_i}$$

Comme Q est premier avec Π, on en déduit que

$$dQ = (\Sigma^+ + \Sigma^-)\Theta$$

où $\Theta = \prod_{i \in I} (X - \alpha_i)$ et si l'on pose $\sigma^+ = \Sigma^+ \Theta$ et $\sigma^- = \Sigma^- \Theta$, alors,

$$dQ = (\sigma^+ + \sigma^-) \tag{6.3}$$

On veut prouver maintenant l'identité

$$d\sqrt{\lambda} = (\sigma^+ - \sigma^-) \tag{6.4}$$

Pour cela il suffit d'observer que le membre de droite est un polynôme de degré inférieur à $N-1$ et qu'il prend la valeur $d\sqrt{\lambda}$ pour les N nombres α_i. Par exemple si α_i est un sommet rouge alors $\sigma^-(\alpha_i)=0$ et donc $(\sigma^+-\sigma^-)(\alpha_i)=(\sigma^++\sigma^-)(\alpha_i)=dQ(\alpha_i)=d\sqrt{\lambda}$.

Maintenant la somme et la différence de 6.3 et 6.4 donnent

$$2\sigma^+=d\Pi^- \text{ et } 2\sigma^-=d\Pi^+ \tag{6.5}$$

Ces équations ont l'avantage d'un degré plus faible et elles découplent en partie les rouges et les bleus. Ce sont elles que l'on peut utiliser avec Maple. En particulier il est toujours possible d'éliminer quelques inconnues qui apparaissent linéairement, et deux équations sont entièrement linéaires.

En revanche, l'équation 6.4 divisée par Θ et développée en $U=1/X$ donne

$$\frac{U^{N-1}}{\prod_{i\in I}(1-U\alpha_i)}=\sum_{i\in I^+}\frac{\nu_i}{1-U\alpha_i}-\sum_{i\in I^-}\frac{\nu_i}{1-U\alpha_i} \tag{6.6}$$

et les $N-1$ premiers termes du développement donnent les équations

$$\sum_{i\in I}\bar{\nu}_i\alpha_i^k=0 \quad \text{pour} \quad 0\le k\le N-2 \tag{6.7}$$

où $\bar{\nu}_i=\nu_i$ pour les rouges et $\bar{\nu}_i=-\nu_i$ pour les bleus.

La première de ces $N-1$ équations est triviale:

$$\sum_{i\in I}\bar{\nu}_i=0$$

Il reste $N-2$ équations non triviales. L'ensemble des solutions $\Upsilon=(\alpha_i)_{1\le i\le N}$ de ce système est comme on pouvait s'y attendre invariant par les transformations affines

$$\Upsilon\mapsto A\Upsilon+B$$

Remarques

– Dans le cas le plus simple où l'arbre est une chaîne de longueur N, alors Π est le polynôme de Tchebitchev normalisé de degré $2N+1$ et 6.7 donne de classiques mais toujours amusantes relations sur les cosinus correspondants.

– Les équations 6.7 forment un système de Vandermonde à l'envers puisque ce sont les α_i que l'on cherche ici, les ν_i étant connus.

On appelle Γ la fonction de $\mathbf{C^N}$ dans $\mathbf{C^{N-2}}$ définie par les membres de gauche de 6.7.

$$\Gamma(\alpha_1,...,\alpha_N)=(\sum_{i\in I}\bar{\nu}_i\alpha_i^k)_{1\le k\le N-2}$$

On note que Γ et sa différentielle Γ' sont très faciles à calculer. Le rang de Γ'_Υ est le minimum de $N-2$ et du nombre de α_i distincts.

Ces observations conduisent à la méthode numérique décrite au paragraphe suivant.

6.2 Méthode Numérique

Nous venons de décrire une classe de systèmes formels inaccessibles aux méthodes usuelles de réduction, tant par leur degré que par le nombre de leurs variables. Nous nous proposons ici d'exploiter la forme très particulière de ces systèmes, les relations qu'ils entretiennent et l'intuition géométrique que nous en avons.

Remarquons que les vecteurs de $\mathbf{C}^N$ définis à affinités près sont en fait des points de $\mathbf{P}_{N-2}$.

A toute suite $(\Upsilon_i)_i$ de vecteurs de $\mathbf{C}^N$ définis à affinité près on associe une suite de $\mathbf{P}_{N-2}$. Si cette seconde suite est convergente on dira abusivement que $(\Upsilon_i)_i$ converge.

Dans ce contexte la méthode de Newton est définie par la formule de récurrence

$$\Upsilon_{i+1} = \Upsilon_i - {\Gamma'_{\Upsilon_i}}^{-1}\Gamma(\Upsilon_i) \tag{6.8}$$

On note que Γ'_{Υ_i} n'est pas inversible. Ainsi ${\Gamma'_{\Upsilon_i}}^{-1}\Gamma(\Upsilon_i)$ est défini à un élément du noyau près. On choisi pour ${\Gamma'_{\Upsilon_i}}^{-1}\Gamma(\Upsilon_i)$ le vecteur orthogonal au noyau, autrement dit on se déplace selon la ligne de plus grande pente.

Ici la forme quadratique à utiliser peut être définie à partir de

- la dérivée seconde de Γ en Υ_i.
- une métrique à définir sur $\mathbf{P}_{N-2}$.

Pour le calcul numérique il faut choisir un représentant convenable de Υ_i dans $\mathbf{C}^N$, pour éviter une divergence artificielle due à un glissement le long des lignes de niveau de Γ.

Il reste maintenant à décrire une heuristique pour le calcul d' une approximation Υ_0 propre à faire converger la méthode.

Pour cela on remarque d'une part que dans les dessins calculés explicitement jusqu'à présent, les segments sont presque rectilignes et les angles autour des sommets sont égaux. Il ne manque plus à cette description schématique qu'une évaluation de la longueur des segments.

Or le système 6.7 varie peu lorsqu'on "éclate" un point, c'est-à-dire lorsque on remplace un point de multiplicité $\bar{\nu}$ par deux points contigus et donc de couleurs opposées et de multiplicités $\bar{\nu}_1$ et $\bar{\nu}_2$ telles que $\bar{\nu}_1 + \bar{\nu}_2 = \bar{\nu}$. Cela revient à greffer un nouveau segment sur le dessin qui pousse alors d'étape en étape comme un arbre. Si l'on appelle Υ^t la solution du premier arbre à N sommets, on peut par exemple obtenir un vecteur Υ^g_0 de dimension $N+1$ en dupliquant la valeur correspondant au sommet éclaté. Ce vecteur peut être pris comme valeur de départ dans la méthode itérative pour le calcul de l'arbre greffé à $N+1$ sommets. En effet, sur les $N-1$ équations 6.7 correspondantes, les $N-2$ premières sont déjà satisfaites parce que ce sont les équations 6.7 de l'arbre précédent.

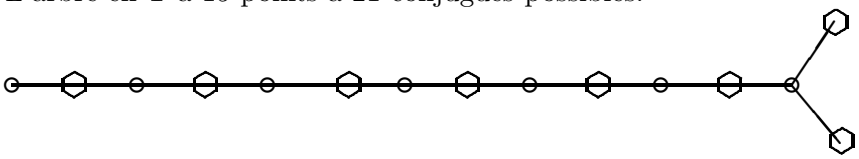
On peut aussi utiliser les longueurs de l'arbre précédent comme des approximations à celles de l'arbre greffé.

Une fois obtenue une bonne approximation de la limite Υ_∞ on a donc un élément de $\mathbf{P}_{N-2}$ que l'on cherche à relever dans $\mathbf{C}^N$ en un point défini sur un corps raisonnable. Pour cela il convient de définir des conditions de rationalité à priori. Si les

approximations calculées sont assez fines, on pourra retrouver les diverses dépendances algébriques à l'aide de l'algorithme LLL de Lenstra, Lenstra et Lovász. On obtiendra alors une solution formelle au problème 6.1.

Exemple

L'arbre en T à 15 points a 21 conjugués possibles.



Le calcul montre qu'il est en effet de degré 21. Son corps est défini par le polynôme

$$
\begin{aligned}
&x^{21} + 2793\,x^{20} + 3689651\,x^{19} + 2965437755\,x^{18} + 1619758438200\,x^{17} \\
&+637272354502000\,x^{16} + 186822742882440000\,x^{15} \\
&+41681786044975200000\,x^{14} + 7173505016349840000000\,x^{13} \\
&+960109537572314640000000\,x^{12} + 100326053969019244800000000\,x^{11} \\
&+8186418890868835008000000000\,x^{10} \\
&+519868623549147989760000000000\,x^{9} \\
&+25510821981353179430400000000000\,x^{8} \\
&+956424272906599190784000000000000\,x^{7} \\
&+26941884916906732872960000000000000\,x^{6} \\
&+556748421325173358848000000000000000\,x^{5} \\
&+8151810519010134928896000000000000000\,x^{4} \\
&+802552253597292693196800000000000000000\,x^{3} \\
&+48809414820672582451200000000000000000\,x^{2} \\
&+1565572545610697723904000000000000000000\,x \\
&+17632149483415558348800000000000000000000
\end{aligned}
$$

Le discriminant du corps, calculé avec Pari par Henri Cohen est

$$-2^{588}3^{382}5^{386}7^{61}11^{20}$$

Pour la méthode de Newton, les calculs numériques ont été menés sous Pari.

6.3 Performances, perspectives et remarques.

Etant donné les capacités de nos implémentations de l'algorithme *LLL*, on peut envisager de calculer des fonctions de Belyi définies sur des corps de degré 30 au moins. De tels résultats sont bien sûr tout à fait hors de portée des méthodes formelles générales, et de très loin. La partie délicate de cette méthode, est la résolution numérique par une méthode itérative dont la convergence peut nécessiter un peu de patience et d'intuition.

Par ailleurs, les équations 6.7 se généralisent au cas d'un dessin de genre 0 quelconque dont toutes les valences sont paires. Si on appelle α_i les sommets et β_j les faces, si $2\nu_i$ est la multiplicité de α_i et $2\mu_j$ celle de β_j, alors à tout sommet α_i, on associe une fonction $\zeta_{0,i}$ définie par

$$\zeta_{0,i}(x) = \sum_j \mu_j \frac{\alpha_i - \beta_j}{x - \beta_j}$$

et à toute face β_j, on associe une fonction $\zeta_{\infty,j}$ définie par

$$\zeta_{\infty,j}(x) = \sum_i \nu_i \frac{\beta_j - \alpha_i}{x - \alpha_i}$$

Alors la fonction $\zeta_{0,i}$ admet un zéro d'ordre $\nu_i - 1$ en α_i et la fonction $\zeta_{\infty,j}$ admet un zéro d'ordre $\mu_j - 1$ en β_j.

En d'autres termes on à les deux séries d'équations suivantes:

pour tout i, et pour $0 \leq k \leq \nu_i - 1$

$$\sum_j \frac{\mu_j}{(\beta_j - \alpha_i)^k} = 0$$

et pour tout j, et pour $0 \leq k \leq \mu_j - 1$

$$\sum_i \frac{\nu_i}{(\alpha_i - \beta_j)^k} = 0$$

La condition de parité sur les multiplicités n'est pas restrictive. En effet, si φ est une fonction de Belyi associée à un dessin quelconque, alors la fonction ψ définie par

$$\psi = -1/4 \left(\frac{\varphi(\varphi - 1)}{\varphi - 2} \right)^2$$

est une fonction de Belyi associée à un dessin dont toutes les multiplicités sont paires au dessus de 0 et ∞ et exactement égales à deux au dessus de 1.

Chapitre 7

Exemple de cryptanalyse par LLL

Dans ce chapitre, nous allons présenter la cryptanalyse d'un système cryptographique à base de sac-à-dos modulaire proposée en 1990 par Niemi. Cette cryptanalyse se décompose en deux parties qui ont été initialement présentées dans [5] et [17].

7.1 Introduction

Soit p un nombre premier et $\mathbf{Z}/p\mathbf{Z}$ le corps fini des entiers modulo p. Comme toujours, nous représenterons, sauf indication contraire, les vecteurs en colonne. Le problème du sac-à-dos modulaire est défini ainsi:

SAC-A-DOS MODULAIRE
INSTANCE: Un nombre premier p, une matrice $E \in (\mathbf{Z}/p\mathbf{Z})^{n\times m}$, et un vecteur $c \in (\mathbf{Z}/p\mathbf{Z})^n$.
QUESTION: Existe-t-il un vecteur $x \in \{0,1\}^m$ tel que $Ex = c$ sur $\mathbf{Z}/p\mathbf{Z}$?
Le SAC-A-DOS MODULAIRE est un problème de décision NP-complet au sens fort, même si l'on ajoute la condition $m = 2n$ [28]. Nous nous intéresserons ici au problème associé consistant à trouver un témoin x lors qu'il existe.

A la conférence Eurocrypt de 1990, Niemi a proposé un nouveau système à clef publique basé sur ce problème [28]. Y.M. Chee à Singapour d'une part, J. Stern et l'auteur de cette thèse à Paris d'autre part, ont découvert indépendamment comment cryptanalyser ce système, ce résultat a été présenté dans [5]. Cette première attaque permet de décrypter les messages, sans toutefois permettre de calculer la clef secrète du système. Elle a été complétée ultérieurement dans [17], par une méthode permettant de retrouver la clef secrète à partir de la clef publique. Ces deux cryptanalyses vont nous permettre de bien montrer la puissance de LLL dans les applications pratiques.

7.2 Le cryptosystème

Avant de cryptanalyser le système de Niemi, nous allons en donner une brève description. L'idée de base de ce système est la notion de valeur absolue dans $\mathbf{Z}/p\mathbf{Z}$. La *valeur absolue* $|g|$ de $g \in \mathbf{Z}/p\mathbf{Z}$ est le plus petit des représentants positifs ou nul de g et

$-g$. On dira de g qu'il est *k-petit* si $|g| \leq k$ et qu'il est *k-grand* si $|g| \geq \lceil p/2 \rceil - k$. Nous parlerons informellement de nombres *petits* et *grands*, et nous fixerons ultérieurement la valeur de k. Le cryptosystème de Niemi se construit alors comme suit, on commence par choisir p un nombre premier et deux entiers positifs, n et $k \ll p$. On choisit ensuite des matrices aléatoires C, D et S de $\mathbf{Z}/p\mathbf{Z}^{n\times n}$ formées de nombres *k-petits*, R une matrice inversible de $\mathbf{Z}/p\mathbf{Z}^{n\times n}$ et Δ une matrice diagonale de $\mathbf{Z}/p\mathbf{Z}^{n\times n}$ formée d'éléments *k-grands*. La clef publique se compose de p et de la matrice $n \times 2n$ $E = (\, A \quad B\,)$, où

$$A = R^{-1}(\Delta - SC) \tag{7.1}$$
$$B = -R^{-1}SD. \tag{7.2}$$

La clef privée est R. Les matrices C, D, S et Δ doivent aussi être gardées secrètes mais elle deviennent inutiles après la phase initiale de construction du cryptosystème. L'espace des messages clairs est $\mathcal{M} = \{x \in \{0,1\}^{2n}\}$, et l'espace des chiffrés est $\mathcal{C} = \{c \in (\mathbf{Z}/p\mathbf{Z})^n\}$. La fonction d'encryption $\mathcal{E}$ de $\mathcal{M}$ dans $\mathcal{C}$ est définie par $\mathcal{E}(x) = Ex$, le calcul étant bien sûr fait modulo p.

Pour déchiffrer c, il suffit de calculer $l = Rc$. D'après les équations 7.1 et 7.2 on a:

$$(\, \Delta \quad \mathbf{0}\,)\, x = l + S\,(\, C \quad D\,)\, x.$$

Comme C, D et S sont formées de petits nombres, et comme $x \in \{0,1\}^{2n}$, $S\,(\, C \quad D\,)\, x$ est aussi formée de petits nombres. Par conséquent, $\Delta_{i,i}x_i = l_i + \alpha_i$ avec α_i petit $(1 \leq i \leq n)$. Afin de déchiffrer, on pose donc:

$$x_i = \begin{cases} 0, & \text{si } l_i \text{ est petit;} \\ 1, & \text{si } l_i \text{ est grand;} \end{cases}$$

pour $1 \leq i \leq n$. Les x_i restants pour $n+1 \leq i \leq 2n$ se calculent par résolution de l'équation linéaire de n équations à n inconnues:

$$Bx^{(2)} = c - Ax^{(1)},$$

où $x^{(1)} = (x_1, \ldots, x_n)^T$ et $x^{(2)} = (x_{n+1}, \ldots, x_{2n})^T$.

Dans son article, Niemi se restreint au cas $S = I$, où I est la matrice identité, et il montre que dès que $p > 4kn$, la méthode de décryption que nous venons de décrire est correcte. De plus, il affirme que $k = 1$ est un bon choix.

Dans la suite, nous allons décrire deux attaques de ce système, la première cherche à reconstituer x à partir de c et E, la seconde à reconstituer R à partir de E. Nous parlerons donc de l'attaque sur les messages et de l'attaque sur les clefs.

7.3 Attaque sur les messages

Cette attaque peut se faire de deux façons différentes (au moins), mais nous ne présentons ici que celle découverte par Joux et Stern. Le lecteur intéressé peut se reporter à [5] pour celle de Chee. Cette attaque s'effectue en réduisant par LLL un réseau bien choisie, dans l'attaque de Chee, seul le choix du réseau est différent.

Etant donné un chiffré c, et l'information publique p et E, la recherche d'une solution de $Ex = c$ sur $\mathbf{Z}/p\mathbf{Z}$ avec $x \in \{0,1\}^{2n}$ est équivalente sur $\mathbf{Z}$ à la recherche de $x \in \{0,1\}^{2n}$ tel qu'il $y \in \mathbf{Z}^n$ vérifiant:

$$(\, E \quad -pI\,) \begin{pmatrix} x \\ y \end{pmatrix} = c, \tag{7.3}$$

nous choisissons λ une grande constante et nous posons:

$$H = \begin{pmatrix} \lambda c & \lambda E & \lambda p I \\ I & & \mathbf{0} \end{pmatrix}$$

Lemme 7.3.1 *Si $x \in \mathbf{Z}^{2n}$ satisfait l'équation 7.3 pour un certain y, alors $\begin{pmatrix} \mathbf{0} \\ 1 \\ -x \end{pmatrix}$ est un vecteur du réseau engendré par H.*

Preuve: Le réseau engendré par H contient:

$$\begin{pmatrix} \lambda c & \lambda E & \lambda p I \\ 1 & 0 & 0 \\ \mathbf{0} & I & \mathbf{0} \end{pmatrix} \begin{pmatrix} 1 \\ -x \\ y \end{pmatrix} = \begin{pmatrix} \lambda c - \lambda E x + \lambda p y \\ 1 \\ -x \end{pmatrix}.$$

Or, par hypothèse, $c - Ex + py = \mathbf{0}$. $\begin{pmatrix} \mathbf{0} \\ 1 \\ -x \end{pmatrix}$ est donc dans le réseau. ∎

De ce lemme, on déduit que si $x \in \{0,1\}^{2n}$ est le message clair associé au chiffré c, alors $v(x) = \begin{pmatrix} \mathbf{0} \\ 1 \\ -x \end{pmatrix}$ est un vecteur court du réseau de H. Plus précisément,

$$\|v(x)\|^2 = \|x\|^2 + 1 \leq 2n + 1.$$

Examinons maintenant les autres vecteurs courts de ce réseau. λ étant grand, les n premières composantes d'un vecteur sont toutes nulles. De plus, faisons l'hypothèse heuristique que les autres composantes (réduites modulo p), sont aléatoires dans l'intervalle $(1-p)/2 \cdots (p-1)/2$. La norme d'un tel vecteur vaut donc, en moyenne $\sqrt{n(p^2-1)/12}$. Dans le cas $p = 4kn$, $v(x)$ est donc environ $2kn\sqrt{2/3}$ fois plus court qu'un vecteur "moyen" ayant ses n premières composantes nulles. Heuristiquement, il est donc vraisemblable que $v(x)$ apparaît dans le réseau avec forte probabilité. Remarquons aussi, que d'autres vecteurs que $v(x)$ peuvent conduire à trouver x. Cette remarque dérive du fait que la somme de quelques petits nombres est encore petite (pour un k différent en général). Ainsi, si l'on trouve dans la base réduite un vecteur de la forme $\begin{pmatrix} \mathbf{0} \\ \mu \\ -x' \end{pmatrix}$, où μ est un petit entier, on peut écrire $E(\mu x - x') = \mathbf{0}$, par conséquent si $\mu x - x'$ est assez petit, on peut déduire qu'il est formé de nombres pairs, et donc que la parité des n premières composantes de x' fournit x.

De plus, il est possible en remplaçant la matrice I dans H, par une matrice formée comme au chapitre 2 pour améliorer la cryptanalyse.

7.4 Attaque sur les clefs

Nous montrons ici, comment à partir de la clef publique E, calculer la clef privée R, à l'aide de LLL. Nous utiliserons LLL n fois, une fois pour chaque ligne de R. Pour cela, nous choisissons de nouveau une grande constante λ et nous posons:

$$L_i = \begin{pmatrix} 0 & \cdots & 0 & \lambda p/2 & 0 & \cdots & 0 & & \\ \lambda A & & & & \lambda B & & & I & \\ & & & & & & & & \\ \lambda pI & & & & & & & 0 & \end{pmatrix}.$$

Le $\lambda p/2$ sur la première ligne de L_i est en $i^{\text{ème}}$ position. L_i est une matrice de taille $(3n+1) \times (3n+1)$. Dans ce cas, nous réduisons par LLL le réseau engendré par les vecteurs **lignes** de L_i. Examinons les vecteurs courts de ce réseau. Tout d'abord, on peut former un certain nombre de vecteurs très courts (et inintéressants) en multipliant la première ligne par 2, ou l'une des n suivantes par p, puis en réduisant les vecteurs obtenus modulo p, à l'aide des $2n$ dernières lignes de L_i. On obtient ainsi, $n+1$ vecteurs linéairement indépendants de petite norme (2 ou p), ces vecteurs sont si courts qu'ils apparaîtront presque sûrement dans la base réduite. De plus, il est facile de constater que si x_1, x_2, $\cdots$, x_{3n+1} représente la base canonique de $\mathbf{R}^{3n+1}$, les $n+1$ vecteurs que nous venons de décrire engendrent le même sous-espace vectoriel que x_{2n+1}, x_{2n+2}, $\cdots$, x_{3n+1}. Recherchons maintenant les vecteurs courts dans le réseau projeté orthogonalement à ce sous-espace. Comme λ intervient partout, nous diviserons toutes les normes par ce facteur pour le faire disparaître. De plus, il est possible de réduire tous les vecteurs ainsi obtenus modulo p, et donc d'en ramener les composantes dans l'intervalle $-(p-1)/2, \cdots, (p-1)/2$. Ceci permet d'estimer la taille d'un vecteur "moyen" de la base réduite par celle d'un vecteur aléatoire de $2n$ éléments dans $\{-(p-1)/2, \cdots, -1, 0, 1, \cdots, (p-1)/2\}$. On obtient très simplement une valeur moyenne de $n(p^2-1)$. Comme $p = 4kn$, cette valeur vaut environ $16k^2n^3$.

Examinons maintenant le vecteur qui nous intéresse, et fournit la ligne i de R. Si l'on note par X_i la $i^{\text{ème}}$ ligne de la matrice X, on peut écrire: $R_i(A, B) = (\delta_i - C_i, -D_i)$, car $S = I$. Donc, toutes les composantes de $R_i(A\ B)$ sauf la $i^{\text{ème}}$ sont petites, et cette dernière est grande. Si on lui ajoute $p/2$ (modulo p), elle aussi devient petite, plus

précisément, elle sera $2k$ petite et les autres k-petites. En relevant ces calculs dans L_i, on obtient un vecteur formé de petits nombres multiplié par λ et suivis de ± 1 et de R_i. Dans le réseau projeté, la taille de ce vecteur (sur λ) vaut au plus $(2n+3)k^2$. Il est donc environ $8n^2$ fois plus petit que les vecteurs décrits précédemment. Ce n'est pas suffisant pour être certain que LLL puisse trouver ce vecteur, mais donne un certaine assurance pour la réalisation pratique de la cryptanalyse.

Bibliographie

[1] E. F. Brickell. Solving low density knapsacks. In *Advances in Cryptology, Proceedings of Crypto'83*, pages 25–37, New York, 1984. Plenum Press.

[2] E. F. Brickell. Solving low density knapsacks. In *Advances in Cryptology, Proceedings of Crypto'84*, pages 342–358, Berlin, 1985. Springer-Verlag.

[3] E. F. Brickell. The cryptanalysis of knapsacks cryptosystems. In R. D. Ringeiseen and F. S. Roberts, editors, *Applications of Discrete Mathematics*, pages 3–23, 1988.

[4] E. F. Brickell and A. M. Odlyzko. Cryptanalysis: a survey of recent results. *Proc. IEEE*, 76:578–593, 1988.

[5] Y. M. Chee, A. Joux, and J. Stern. The cryptanalysis of a new public-key cryptosystem based on modular knapsacks. In J. Feigenbaum, editor, *Advances in Cryptology, Proceedings of Crypto'91*, volume 576 of *Lecture Notes in Computer Science*, pages 204–212, New York, 1991. Springer-Verlag.

[6] B. Chor and R. Rivest. A knapsack-type public key cryptosystem based on arithmetic in finite fields. *IEEE trans. Information Theory*, IT-34:901–909, 1988.

[7] M. J. Coster, B. A. LaMacchia, A. M. Odlyzko, and C.-P. Schnorr. An improved low-density subset sum algorithm. In D. W. Davies, editor, *Advances in Cryptology, Proceedings of Eurocrypt'91*, volume 547 of *Lecture Notes in Computer Science*, pages 54–67, New York, 1991. Springer-Verlag.

[8] Y. Desmedt. What happened with knapsacks cryptographic schemes? In J. K. Skwirzynsky, editor, *Performance Limits in Communication, Theory and Practice*, pages 113–134. Kluwer, Boston, 1988.

[9] P. Flajolet and B. Vallée. The lattice reduction algorithm of gauss: An average case analysis. In *31st FOCS*, pages 830–846, Los Alamitos, 1990. IEEE.

[10] A. M. Frieze. On the lagarias-odlyzko algorithm for the subset sum problems. *SIAM J. Comput.*, 15(2):536–539, 1986.

[11] M. L. Furst and R. Kannan. Succinct certificates for almost all subset sum problems. *SIAM J. Comput.*, 18:550–558, 1989.

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

[13] C.F. Gauss. Disquisitiones arithmeticae. Leipzig, 1801.

[14] A. Grothendieck. Esquisse d'un programme. Notes non publiées.

[15] J. Håstad, B. Just, J. C. Lagarias, and C.-P. Schnorr. Polynomial time algorithms for finding integer relations among real numbers. *SIAM J. Comput.*, 18(5):859–881, 1989.

[16] C. Hermite. Extraits de lettres de m. hermite à m. jacobi sur différents objets de la théorie des nombres, deuxième lettre. *J. Reine Angew. Math*, 40:279–290, 1850.

[17] A. Joux and J. Stern. Cryptanalysis of another knapsack cryptosystem. In *Advances in Cryptology, Proceedings of AsiaCrypt'91*, Lecture Notes in Computer Science, New York, 1991. Springer-Verlag.

[18] A. Joux and J. Stern. Improving the critical density of the lagarias-odlyzko attack against subset sum problems. In L. Budach, editor, *Proceedings of Fundamentals of Computation Theory 91*, volume 529 of *Lecture Notes in Computer Science*, pages 258–264, New York, 1991. Springer-Verlag.

[19] R. Kannan. Improved algorithms for integer programming and related lattice problems. In *Proc. 15th Symp. Theory of Comp.*, pages 193–206, 1983.

[20] A. Korkine and G. Zolotarev. Sur les formes quadratiques. *Math. Ann.*, 6:336–389, 1873.

[21] J. C. Lagarias and A. M. Odlyzko. Solving low-density subset sum problems. *J. Assoc. Comp. Mach.*, 32(1):229–246, 1985.

[22] L. Lagrange. *Recherches d'arithmétique*, pages 265–312. Nouv. Mém. Acad., Berlin, 1773.

[23] B. A. LaMacchia. Basis reduction algorithms and subset sum problems. Master's thesis, Dept. of Elect. Eng. and Comp. Sci., Massachusetts Institute of Technology, Cambridge, MA, 1991.

[24] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:515–534, 1982.

[25] J. E. Mazo and A. M. Odlyzko. Lattice points in high-dimensional spheres. *Monatsh. Math*, 110:47–61, 1990.

[26] R. C. Merkle and M. E. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Trans. Inform. Theory*, IT–30:594–601, 1984.

[27] H. Minkowski. *Geometrie der Zahlen*. Teubner, Leipzig, 1910.

[28] V. Niemi. A new trapdoor in knapsacks. In I. B. Damgård, editor, *Advances in Cryptology, Proceedings of Eurocrypt'90*, volume 473 of *Lecture Notes in Computer Science*, pages 405–411, New York, 1991. Springer-Verlag.

[29] A. M. Odlyzko. The rise and fall of knapsack cryptosystems. In C. Pomerance, editor, *Cryptology and Computational Number Theory*, volume 42 of *Proc. Symp. Appl. Math.*, pages 75–88, Providence, 1990. Amer. Math. Soc.

[30] A. Paz and C.-P. Schnorr. Approximating integer lattices by lattices with cyclic factor groups. In *Automata, Languages and Programming:* 14^{th} *ICALP*, volume 267 of *Lecture Notes in Computer Science*, pages 386–393, New York, 1987. Springer-Verlag.

[31] S. Radziszowski and D. Kreher. Solving subset sum problems with the l^3 algorithm. *J. Combin. Math. Combin. Comput.*, 3:49–63, 1988.

[32] A. H. Sameh and D. J. Kuck. On stable parallel linear system solvers. *J. Assoc. Comp Mach.*, 25(1):81–91, 1978.

[33] C.-P. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical Computer Science*, 53:201–224, 1987.

[34] C.-P. Schnorr. A more efficient algorithm for lattice basis reduction. *J. Algorithms*, 9:47–62, 1988.

[35] C.-P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. In L. Budach, editor, *Proceedings of Fundamentals of Computation Theory 91*, volume 529 of *Lecture Notes in Computer Science*, pages 68–85, New York, 1991. Springer-Verlag.

[36] M. Seysen. Simultaneous reduction of a lattice basis and its reciprocal basis. *Combinatorica*. to appear.

[37] G. B. Shabat and V. A. Voevodsky. Drawing curves over number fields. In *Papers in honour of A. Grothendieck. The Grothendieck Festschrift*, pages 199–229. Birkhauser, 1990.

[38] A. Shamir. A polynomial time algorithm for breaking the basic merkle-hellman cryptosystem. *IEEE Trans. Inform. Theory*, IT–30:699–704, 1984.

[39] B. Vallée. Gauss' algorithm revisited. *J. Algorithms*, 12:556–572, 1991.

[40] P. van Emde Boas. Another np-complete partition problem and the complexity of computing short vectors in a lattice. Technical Report 81–04, Dept. of Mathematics, Univ. of Amsterdam, 1981.

[41] G. Villard. Parallel lattice basis reduction. In Paul S. Wang, editor, *ISSAC '92*, pages 269–277, New York, 1992. ACM Press. Proceedings, July 27–29, Berkeley.

[42] J. H. Wilkinson. The algebraic eigenvalue problem. In *Monographs on Numerical Analysis*. Clarendon Press, Oxford, 1965.

Table des matières

I Résultats théoriques **3**

1 Introduction à la réduction de réseau **5**
1.1 Historique 5
1.2 Quelques Définitions 6
1.3 L'algorithme de Gauss 9
1.4 L'orthogonalisation de Gram Schmidt 13
1.5 L'algorithme LLL 15
1.6 La réduction au sens de Korkine-Zolotarev 19

2 Les problèmes de sac-à-dos **23**
2.1 Introduction 23
2.2 Description de l'algorithme de Lagarias-Odlyzko 24
2.3 Amélioration de la densité critique 27
2.4 Une autre amélioration de la densité critique 29
2.5 Influence de la proportion de 0 dans e 31

3 La parallélisation de LLL **37**
3.1 L'approche de Villard 39
3.2 L'orthogonalisation de Givens 39
3.3 Retour à la réduction parallèle 42
3.4 L'algorithme 43
3.5 Preuve de l'algorithme 45

II Programmation et applications **49**

4 Réalisation d'un programme de réduction de réseau **51**
4.1 Mode d'emploi du programme 51

5 Expérimentation, réseaux aléatoires, sac-à-dos aléatoires **53**
5.1 Les réseaux aléatoires 53
5.2 Les sacs-à-dos 55

6 Calculs de dessins d'enfant **65**
6.1 Dessins d'enfants, Introduction 65
6.2 Méthode Numérique 68
6.3 Performances, perspectives et remarques. 69

7 Exemple de cryptanalyse par LLL **71**
7.1 Introduction 71
7.2 Le cryptosystème 71
7.3 Attaque sur les messages 72
7.4 Attaque sur les clefs 74