Projet annuel
Modélisation de l'algorithme LLL
par CFG

(LOISEL Lucas 21711075, GUILLOUET Arthur 21707423)

Sous le tutorat de M. Ali Akhavi,

Mme. Brigitte Vallée & M. Julien Clement

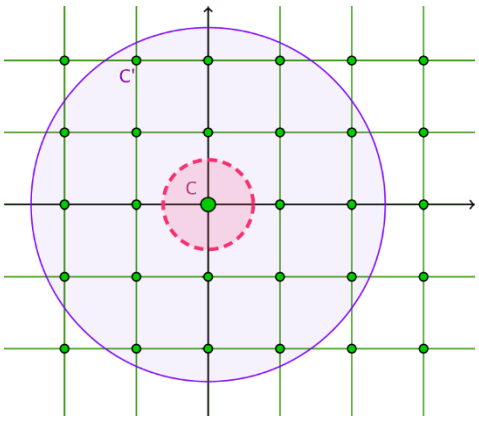
Tables des matières

Introduction

En 1982, A.Lenstra, H.Lenstra et L. Lovász propose un algorithme de réduction de base s'exécutant en temps polynomial (PTIME) de complexité O(d5nlog3B) . L'algorithme LLL portant les initiales de ses 3 créateurs joue un rôle important dans plusieurs domaines notamment dans celui de la cryptographie via son implication dans la réduction des réseaux euclidiens. Il a été utilisé pour factoriser des polynômes. Cependant, nous pouvons le retrouver en cryptographie à clé publique par exemple avec le chiffrement RSA nommé par les initiales de ses trois inventeurs qui est très utilisé dans le commerce électronique. De plus, l'algorithme LLL a rendu inefficaces les cryptosystèmes basés sur le problème du sac à dos. Cet algorithme est très efficace et a été utilisé pour résoudre un grand nombre de problèmes mais reste néanmoins mal compris. Il repose sur une réduction d'une base de réseau obtenue en transformant un réseau quelconque en un réseau possédant des vecteurs assez courts et presque orthogonaux (base réseau LLL-réduite). Pour modéliser de manière simplifiée l'algorithme LLL, il a été proposé récemment d'utiliser les tas de sable. Ainsi, cette méthode simplifiée a permis de retrouver les principales caractéristiques de l'algorithme étudié. Nous nous appuierons sur cette modélisation pour étudier cet algorithme. Dans ce rapport, nous approcherons et étudierons l'algorithme à travers 3 parties. Ainsi, nous allons d'abord aborder un premier lieu les réseaux euclidiens et leurs fonctionnements. Ensuite, nous allons poursuivre sur l'étude de la réduction de base et nous allons faire un lien explicite avec l'algorithme de A.Lenstra, H. Lenstra et L.Lovasz. Et pour finir, nous allons évoquer la modélisation de l'algorithme LLL par CFG qui nous permettra d'éclaircir et de faciliter la compréhension de ce fameux algorithme à travers cette exemple concret.

Le terme réseau étymologiquement “retiolus” fait son apparition au XIIe siècle. D’abord employé pour les domaines du textile, de la biologie, du génie militaire, de l’économie ou encore de l'économie, il fait son apparition en mathématiques, que plus tard. Actuellement, l’étude des réseaux recouvre différents domaines des mathématiques, tels que, l'algèbre linéaire, la géométrie des nombres ou encore dans l’étude des cristaux avec la cristallographie (réseau de Bravais).

Nous allons nous intéresser durant ce projet, aux problèmes arithmétiques (théorie des nombres) et plus précisément à la géométrie des nombres. Originellement fondée à la fin du XIXe siècle par Minkowski, elle a pour contexte initial l’observation élémentaire des formes quadratiques introduites dans les travaux de plusieurs mathématiciens tels que le Joseph-Louis Lagrange ou encore Carl Friedrich Gauss (algorithme d’élimination de Gauss-Jordan trouve la meilleure base pour dimension 2).



Observations élémentaire (Le disque violet contient des points du réseau pas le rose) Image tirée de Wikipédia la géométrie des nombres (par Cgolds)

Grossièrement, la réduction d’un réseau dans un espace euclidien consiste à transformer ce réseau en un réseau avec des vecteurs assez courts et assez orthogonaux afin d’obtenir une bonne base.

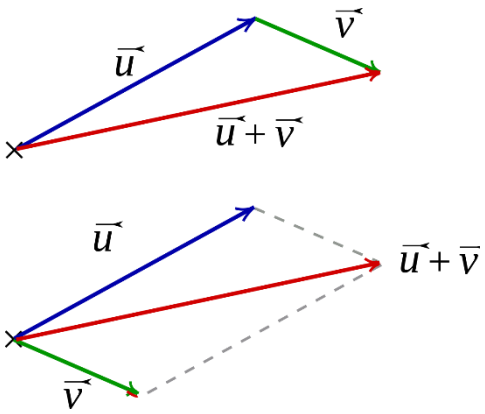
L’algorithme LLL, étudié dans ce projet, du nom de ses inventeurs Arjen K. Lenstra, Hendrik W. Lenstra et László Lovász, permet un développement notable de la géométrie des nombres d’un point de vue algorithmique. Il repose sur le principe de réduction de base vu précédemment et trouve de manière efficace une base réduite appelée base de réseau LLL-réduite.

1 Les réseaux euclidiens

1.1 Introduction aux réseaux euclidiens

Les réseaux euclidiens ont été considéré pour la première fois par Christian Huygens dans son livre Traité de la lumière publié en 1690.

Rappel : *Un espace euclidien est un espace vectoriel de dimension fini constitué de vecteurs où la norme représente la longueur et où la structure permet d'appliquer les opérations élémentaires dans le but de traiter des combinaisons linéaires.*



Somme vectoriel en géométrie euclidienne
Image tirée de Wikipédia Somme vectorielle (par Trassiorf)

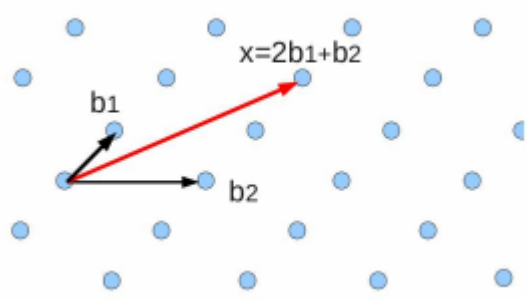
Un réseau d'espace euclidien est un ensemble de points se trouvant à intervalle régulier dans un espace vectoriel euclidien fini R^n. Autrement dit, pour reprendre la thèse de doctorat de Mariya Georgieva.

Définition d'un réseau tiré de la thèse de Mariya Georgieva: *Un réseau euclidien L est un sous-groupe additif discret non-vide de* $R^n (n \geq 0)$.

Un système B = (b1, b2, . . . , bn) ayant des vecteurs linéairement indépendants de R^n est une base du réseau si l'ensemble des éléments de B s'expriment comme une combinaison linéaire, à coefficients dans les entiers Z.

Autrement dit, si $L(B) := \{x \in R^n ; x := \sum_{i=1}^{d} x_i b_i , xi \in Z\}$

Nous pouvons illustrer nos propos à travers un exemple très simple et explicite avec un réseau bidimensionnel :



Réseau à 2 dimensions tirée du cours sur la modélisations de l'algorithme LLL et de ses entrées par des systèmes dynamiques de Loïck Lhote

Ainsi, nous pouvons voir sur cette image une illustration de ce que nous avons dit précédemment. Ici, le vecteur x est une combinaison linéaire des vecteurs du réseau avec pour coefficients entier 2 (pour le vecteur b1) et 1 (pour le vecteur b2).

Remarque : *Il existe une unique dimension n pour les toutes bases du réseau L.*

Pour conclure cette partie, il existe une infinité de bases du réseau L. On verra dans la partie 2, que l'algorithme LLL trouve une bonne base pour ce réseau qui n'est certainement pas la meilleure, mais qui reste néanmoins bonne en un temps acceptable (avec des vecteurs assez courts et assez orthogonaux).

2.2 Réduction de réseaux et orthogonalisation de Gram-Schmidt

Pour continuer, dans la suite logique de l'étude des réseaux, nous allons aborder la réduction de réseaux. Ainsi, comme nous avons pu le dire précédemment, un réseau possède une infinité de bases. Et certaines bases sont plus intéressantes que d'autres. Donc, l'objectif est de trouver une "bonne base" qui ne sera cependant pas la meilleure pour les réseaux à grande dimensions (possible d'obtenir la meilleure base pour les réseaux de petite dimension) mais qui reste relativement correcte. Pour cela, nous souhaitons obtenir une base avec des vecteurs assez courts et assez orthogonaux. L'algorithme de réduction de Lenstra, Lenstra et Lovász procède à une réduction de base d'un réseau (dans un temps de calcul relativement raisonnable) à travers plusieurs critères et procédés. Dans la suite de cette partie, nous allons donc voir

l'orthogonalisation de Gram-Schmidt que nous avons pu utiliser dans la modélisation de l'algorithme LLL par CFG.

Le procédé de Gram-Schmidt a pour but de construire une base orthonormée $B^* := (b^{*1}, b^{*2}, \ldots, b^{*n})$ du sous-espace qu'elle engendre, à partir d'une base $B = (b1, b2, \ldots, bn)$. Donc, même si la dimension reste inchangée, la base B* obtenue ne se trouve pas nécessaire dans le réseau initial.

Pseudo-code de l'algorithme de Gram-Schmidt :

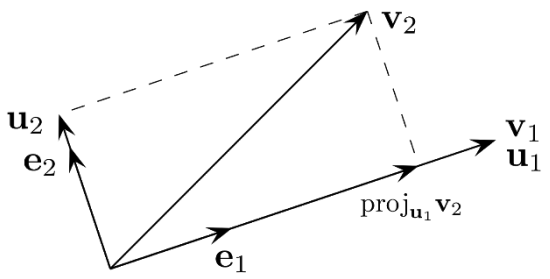
Algorithm 1: Algorithme de Gram-Schmidt

Enter: Une base $B = (b1, ..., bn)$;
Result: Une base $B^* = (b^*1, ..., b^*n)$ orthogonalisée de B;
$b^*1 := b1$;
for *i allant de 1 à n* **do**
  $b^*i := bi$;
  for *j allant de 1 à i* **do**
    $b^*i := b^*i - \frac{< b^*j | bi >}{||b^*j||^2} b^*j$;
  end
end

Pseudo-code de l'algorithme utilisé dans notre projet

Pour mieux comprendre le procédé étudié, nous pouvons nous appuyer sur le pseudo-code utilisé. En effet, nous pouvons voir que pour obtenir la matrice B* qui nous le rappelons et la base orthonormée construite à partir de B, nous calculons b*i étant en réalité la projection orthogonale de bi sur le sous-espace engendré par (b1, ..., bi-1). Pour illustrer nos propos, nous allons utiliser un exemple simple en décrivant les 2 premières étapes du procédé.

Par exemple :



Les premières étapes du procédé

Dans la première étape, nous pouvons voir que comme dans le pseudo code u1 := v1 $\Leftrightarrow$ b*1 := b1. Dans la deuxième étape, nous apercevons la projection orthogonale de v2 sur le vecteur u1 := v1 pour obtenir u2. Donc la projection de v2 sur u1 (dans l'illustration) est définie comme dans l'algorithme c'est-à-dire :

$$proj_{u_1}(v_2) = \frac{\langle u_1|v_2\rangle}{\langle u_1|u_1\rangle} * u_1 = \frac{\langle v_1|v_2\rangle}{\langle v_1|v_1\rangle} * v_1$$

Nous pouvons donc à partir de cette formule déterminer la matrice de passage M définit par $m_{i,j} = \frac{\langle b^*{}_j|b_i\rangle}{\langle b^*{}_j|b^*{}_j\rangle}$ qui nous l'expliquerons dans la seconde partie aura un rôle clé dans le fonctionnement de l'algorithme LLL.

1.3 Les réseaux en cryptographie.

La cryptographie est présente partout dans nos appareils numériques. Elle permet la sécurité de nos données (carte bancaire ou autres) à travers des mots de passe ou encore des codes pins. Les algorithmes cryptographiques asymétriques jouent le rôle de protecteur des données en utilisant des clés privées et publiques. Donc, sans la clé privée, il est impossible d'accéder aux données d'une personne en un temps raisonnable. La majorité des systèmes utilise deux techniques : le problème de la factorisation des entiers (chiffrement RSA) et le problème du logarithme discret. Comme la majorité des ordinateurs ne sont pas assez puissants pour les résoudre en un temps raisonnable, les systèmes semblent sécurisés. Des travaux, comme ceux de Miklos Ajtai, démontrent que les réseaux euclidiens pourraient servir de base solide en cryptographie. Logiquement, connaître la distance minimale entre 2 points dans un réseau euclidien est considéré comme un problème difficile. Ce problème n'est pas encore utilisé. Mais dans un futur proche, avec l'arrivée d'ordinateurs assez puissants, les ordinateurs quantiques pour résoudre en temps raisonnable le problème de la factorisation des entiers et le problème du logarithme discret, il pourrait trouver sa place afin de protéger ces nouveaux systèmes.

Nous pouvons aussi retrouver son utilité en cryptanalyse notamment dans l'attaque de certains algorithmes cryptographiques asymétriques comme le chiffrement RSA.

2 Réduction de base : Etude de l'algorithme LLL

Comme vu dans la partie précédente, certaines bases de réseaux sont plus aptes que d'autres bases pour travailler au sein d'un réseau euclidien car celles-ci sont notamment formées de vecteurs cours et permettent donc un meilleur parcours du réseau. Il existe de nombreux algorithmes qui procèdent à une réduction de base en un temps polynomial comme par exemple l'algorithme LLL, l'algorithme BKZ (Block,Korkine,Zolotorev) en 1987 ou encore l'algorithme RSR en 2002. Nous nous sommes intéressés, dans ce projet, par le premier d'entre eux, c'est-à-dire l'algorithme LLL. Si l'algorithme LLL est connu dans le domaine des réseaux c'est qu'il connaît, depuis une trentaine d'années, un succès dans la cryptologie. Il permet notamment la conception de cryptosystèmes (NTRU, ou Ajtai-Dwork) et de chiffrements homomorphes. Il est également utilisé dans la cryptanalyse de systèmes de chiffrement.

L'algorithme LLL, inventé en 1982 par H. Lenstra, A. Lenstra et L. Lovász, est un algorithme qui construit une base dite faiblement réduite (ou encore LLL-réduite) se rapprochant au maximum de la base de Gram Schmidt en un temps polynomial. Cet algorithme s'exécute en temps polynomiale ($O(d^5 n log^3 B)$ où B est la norme du plus grand vecteur de la base à réduire).

2.1 Principe de l'algorithme

Cet algorithme s'articule sous différentes phases, on reviendra précisément sur chacune d'entre elles plus tard.

La première phase est une phase d'initialisation. Elle permet de calculer la base b $*$, de construire la matrice de passage P et d'initialiser $l_i = |b_i^*|^2$ sous le procéder d'orthogonalisation de Gram-Schmidt vu précédemment. La base b $*$ ne sera qu'utile, dans l'algorithme, seulement pour construire la matrice de passage P qui sera essentielle dans chaque phase.

1. La seconde phase est une phase de *translation* des vecteurs b_i qui rend la base *propre* (On reviendra sur la propreté d'une base plus tard). Cette phase ne modifie pas la base b_i^*.

La troisième phase est une phase *d'échange* des vecteurs b_i et b_{i+1} qui ne respectent pas :

- La condition de s-Siegel sur la base b^*.
- La condition de t-Gauss sur le système B_i .

Cette phase modifie complétement b^*, l_i et la matrice P du fait de l'échange entre les vecteurs b_i et b_{i+1}.

2.2 Matrice de passage P et système B_i.

Revenons tout d'abord à la matrice de passage P et le système B_i. On rappelle que l'algorithme LLL prend en entrée une base $B = (b_1, \dots, b_d) \in R^{d \times n}$ qu'on cherche à réduire au maximum en fonction de la base orthogonalisé de Gram-Schmidt $B^* = (b_1{}^*, \dots, b_d{}^*)$. On obtient donc la matrice de passage :

$$P : B \to B^*$$

$$\begin{array}{c} \\ B_1 \\ B_2 \\ \vdots \\ B_i \\ B_{i+1} \\ \vdots \\ B_d \end{array} \begin{array}{c} \begin{array}{cccccccc} B_1^* & B_2^* & \cdots & B_i^* & B_{i+1}^* & \cdots & B_d^* \end{array} \\ \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ m_{2,1} & 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ m_{i,1} & m_{i,2} & \cdots & 1 & 0 & 0 & 0 \\ m_{i+1,1} & m_{i+1,2} & \dots & m_{i+1,i} & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ m_{d,1} & m_{d,2} & \dots & m_{d,i} & m_{d,i+1} & \cdots & 1 \end{pmatrix} \end{array}$$

Nous pouvons donc remarquer que les vecteurs u_i et v_i du système B_i se lisent sur la matrice P avec :

$$B_i = \begin{array}{c} \\ u_i \\ v_i \end{array} \begin{array}{c} \begin{array}{cc} B_i^* & B_{i+1}^* \end{array} \\ \begin{pmatrix} 1 & 0 \\ m_{i+1,i} & 1 \end{pmatrix} \end{array}$$

C'est sur cette boîte B_i que les phases de translation et d'échange s'effectuent. La phase de translation permet de réduire les coefficients de la matrice P. En effet si

$$|m_{i,j}| \geq \frac{1}{2} \; pour \; 1 \leq \mathrm{j} < \mathrm{i} \leq \mathrm{d},$$

Alors on procède à la mise en propreté de la base. De même si

$$|v_i|^2 \geq \frac{1}{t^2} |u_i|^2 \text{ où } t \; est \; le \; paramètre \; de \; l'algorithme \; LLL.$$

Alors on effectue l'échange entre les vecteurs b_i et b_{i+1} à l'aide de l'algorithme de Gauss.

La matrice de passage P joue donc un rôle essentiel dans le fonctionnement de l'algorithme LLL puisque c'est autour d'elle que les différentes phases se réalisent.

2.3 Propreté d'une base.

Un réseau engendré par une base $B = (b_1, \dots, b_d)$ n'admet pas nécessairement une base orthogonale mais elle possède une rapproche qui se rapproche au mieux de celle de Gram-Schmidt. La première idée venue est de minimiser les coefficients de la matrice de passage P, pour rapprocher B et $B*$, sans modifier $B*$ et le réseau engendré par B. On peut ainsi définir une base propre comme suit :

Définition : Une base $B = (b_1, \dots, b_d)$ *est dite propre si tous les coefficients de la matrice de passage P,* $|m_{i,j}| \leq \frac{1}{2}$ *pour* $1 \leq j < i \leq d$.

Il existe un algorithme qui peut rendre toutes bases d'un réseau en base propre. Cet algorithme a été introduit par Hermite. Voici son pseudo-code :

Algorithm 1 : B = $(b_1, \dots, b_n)$

$B^* \leftarrow GramSchmidt(B)$
for i de 2 à n **do**
  for j de i-1 à 1 **do**
    $b_i \leftarrow b_i - \lfloor m_{i,j} \rceil b_j$
    for k de 1 à j **do**
      $m_{i,k} \leftarrow m_{i,k} - \lfloor m_{i,j} \rceil m_{i,k}$
    end for
  end for
end for

Figure 1 Algorithme Propre

On peut donc affirmer que l'algorithme propre ne modifie ni b^* ,ni le réseau engendré par B. En effet l'algorithme ne fait que translater chaque vecteur b_j avec b_i où $j < i$. Il ne modifie seulement que les coefficients en les soustrayant avec leur entier le plus proche. Ainsi à chaque échange la matrice est mise à jour et chaque coefficient se situe dans l'intervalle [- ½, ½]. Il est à noter qu'à chaque translation les normes des vecteurs diminuent ce qui réduit considérablement le temps d'exécution de l'algorithme.

Cependant cet algorithme ne suffit pas à rendre une base presque orthogonale. En effet cette procède seulement à la réduction de coefficients de la matrice de passage mais ne permet pas de minimiser l'angle de b_i avec l'hyperplan H_{i-1} engendré par $(b_1, \dots, b_{i-1})$. Pour pouvoir minorer l'angle que forme b_i avec l'hyperplan H_{i-1} il est nécessaire de minorer la projection de b_i orthogonalement avec H_{i-1} où tout simplement la norme du vecteur b_i^*.

Cette minoration est l'objet de la condition de Siegel.

2.4 Conditions de réduction.

On rappelle qu'une réduction de réseau se définit comme suit. Le but d'une réduction de réseau est de former une base avec des vecteurs assez orthogonaux. Un réseau ne possède pas nécessairement une base orthogonale. Le procédé de Gram-Schmidt engendre bien une base b^* orthogonal du Q-espace engendré par b, mais celle-ci n'appartient pas, en général, au réseau $L(b)$.

Même si l'algorithme LLL utilise la réduction Lovasz, il n'en existe d'autres. La première est celle de Minkowski. Cette dernière cherche à réduire au maximum les longueurs des vecteurs. Les trois autres réductions, celles de Siegel, Lovâsz et Korkine-Zolotarev cherchent à réduire au maximum b^* parallèlement à la base b. Ce sont les réductions de Siegel et de Lovâsz qui nous intéressent le plus dans le cadre de l'algorithme LLL car ces deux réductions permettent de réduire une base b en un temps polynomial. Enfin il est démontré que la réduction au sens de Siegel englobe les autres réductions. En effet, toute base réduite en une des trois autres réductions, est aussi réduite au sens de Siegel.

2.4.1 Réduction de Siegel

Une base propre ne garantit pas qu'elle soit orthogonale. Une base propre signifie seulement que la projection sur H_i de b_{i+1} est petite. Il faut en plus minorer l'angle θ_{i+1} entre b_{i+1} et H_i . Pour cela, il nous faut réduire la projection orthogonale entre H_i et b_{i+1} , c'est-à-dire ${b_{i+1}}^*$. C'est ce à quoi sert la réduction de Siegel qui se définit comme suit :

$$|{b_{i+1}}^*| \geq \frac{1}{s} |{b_i}^*| \text{ , pour } 1 \leq i \leq n-1$$

Ou encore,

$$l_{i+1} \geq \frac{1}{s} l_i$$

Ou bien,

$$\frac{l_{i+1}}{l_i} \geq \frac{1}{s} \text{ où } \frac{l_{i+1}}{l_i} = r_i$$

Ce rapport $\frac{l_{i+1}}{l_i}$ est la pièce centrale dans la modélisation de l'algorithme LLL, sur laquelle on reviendra dans la prochaine partie.

Une base est dite s-Siegel réduite si elle répond aux conditions suivantes :

- Elle est propre.
- Elle vérifie la condition de Siegel présentée au-dessus.

On choisit habituellement une valeur $s \geq 1$ pour faciliter les calculs. La valeur s la plus communément choisie est $s = \frac{2}{\sqrt{3}}$.

2.4.2 Réduction au sens de Lovâsz

On rappelle qu'une base réduite au sens de Siegel est également réduite au sens de Lovâsz. En effet, une base au sens de Lovâsz est réduite si elle remplit les conditions suivantes :

- Elle est propre (Comme Siegel)
- Elle vérifie la condition suivante :

$$\|v_i\|^2 \geq \frac{1}{t} \ \|u_i\| \text{ où t } > 1$$

Ou bien,

$$\|b_i{}^*\|^2 \geq \left(t - \ m_{i,j}{}^2 \right) \left(\|b_{i-1}{}^*\|^2 \right)$$

Une base propre est dite t-réduite si elle vérifie la condition de Lovâsz avec le paramètre t. Les deux paramètres s et t, respectivement de la réduction de Siegel et de la réduction au sens de Lovâsz sont liés par la relation $s = \sqrt{\frac{4t^2}{(4-t^2}}$.

L'idée de Lovâsz était de généraliser l'algorithme de Gauss qui s'applique sur des réseaux de dimensions 2 et de le généraliser sur des réseaux plus grands, en appliquant l'algorithme de Gauss sur les boites B_i locales :

$$B_i = \begin{array}{c} \\ u_i \\ v_i \end{array} \begin{array}{c} \begin{array}{cc} B_i^* & B_{i+1}^* \end{array} \\ \begin{pmatrix} 1 & 0 \\ m_{i+1,i} & 1 \end{pmatrix} \end{array}$$

L'algorithme LLL permet de construire une base réduite au sens de Lovâsz avec un paramètre $t > 1$ qui fonctionne en un temps polynomial.

2.5 Etapes de Translation et d'échanges

Translation :

Comme dit précédemment l'étape de translation permet de réduire les normes des vecteurs de B^* et de rendre propre la base. On rappelle est dite propre si pour tous les coefficients de la matrice de passage P, $|m_{i,j}| \leq \frac{1}{2}$ *pour* $1 \leq \text{j} < \text{i} \leq \text{d}$.

Echange :

La seconde étape de l'algorithme est l'étape des échanges. Elle permet de rendre la base plus orthogonale. L'algorithme prend un paramètre $s > 1$, si la condition suivante de Siegel n'est pas respectée :

$$|b_{i+1}{}^*| \geq \frac{1}{s} \ |b_i{}^*| \text{ où s} > 1$$

Alors l'algorithme procède à l'échange de b_i et b_{i+1}.

On rappelle que $l_i = |b_i^*|^2$ et que $r_i = \frac{l_{i+1}}{l_i}$ et que $v_i = \{m_{i+1,i}\}$. Après les échanges des deux vecteurs b_i et b_{i+1} les nouveaux rapports de Siegel sont donc :

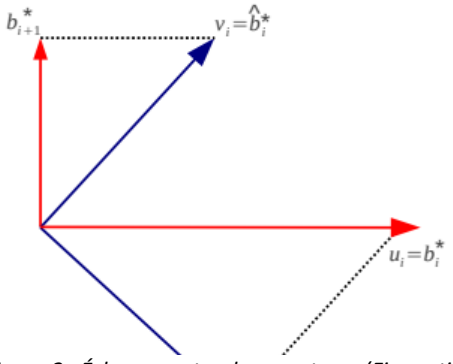


Figure 2 : Échange entre deux vecteurs (Figure tirée de la thèse de Mariya Georgieva)

$$\begin{cases} \hat{r}_{i-1} = p.\hat{r}_{i-1} \\ \hat{r}_i = \frac{1}{p^2}.\hat{r}_i \\ \hat{r}_{i+1} = p.\hat{r}_{i+1} \end{cases}$$

Avec un facteur de décroissance p qui vérifie $p^2 = {r_i}^2 + {v_i}^2 \leq 1$. Cette étape d'échange est importante dans l'algorithme de réduction car c'est elle qui permet d'augmenter la valeur des rapports de Siegel. Ce sont ces rapports de Siegel et l'étude du décrément qui jouent un rôle important dans l'algorithme LLL car c'est eux qu'on cherche à augmenter pour obtenir une base plus propre.

Retour sur l'algorithme :

Finalement nous avons pu remarquer que l'algorithme LLL parcours les rapports de Siegel, si un rapport ne vérifie pas la condition de Siegel alors on procède à l'échange des vecteurs de la base. Finalement, l'indice n parcourt les bases locales dans $[1..d-1]$. Cet indice correspond à la plus grande base $(b_1, b_2, .., b_n)$ qui est t-réduit au sens de Siegel.

Pseudo code de l'algorithme :

Algorithm 1: Algorithme LLL

Enter: Une base B = ($b_1, ..., b_n$) à réduire et un paramètre t;
Result: Une base B = ($b_1, ..., b_n$) réduite;
$B^* := Gram - Schmidt(B)$;
$P := Matricedepassage$;
; n:= 1;
while $n <= n - 1$ **do**
  for j *from* $k - 1$ *to* 0 **do**
    if $|P[k][j]| > 1/2$ **then**
      $b_k := b_k * \lfloor P[k][j] \rceil * b_j$;
      $B^* := Gram - Schmidt(B)$;
      $P := Matricedepassage$;
    end
  end
  if $\langle B_k^*, B_k^* \rangle \geq (t - P[k][j]^2).(\langle B_{k-1}^*, B_{k-1}^* \rangle)$ **then**
    $n := n + 1$;
  end
  else
    Echanger b_k et b_{k-1};
    $B^* := Gram - Schmidt(B)$;
    $P := Matricedepassage$;
    $n := max(n - 1, 1)$;
  end
end
Return B: Base réduite ;

Figure 3 : Pseudo code de l'algorithme LLL

L'algorithme LLL permet donc de réduire une *base* $B = (b_1, \dots, b_d)$ en une base réduite en un temps polynomial : $O(d^5 n log^3 B)$. Celui-ci a connu un grand succès dans la cryptologie dans lequel il permet d'effectuer de la cryptoanalyse (RSA) ou encore de résoudre des problèmes complexes desquels il était impossible de les résoudre. On pense notamment au problème de la programmation linéaire en nombre entier, au problème de calcul de racines n-ièmes modulo m ou, enfin, à factorisation de nombres entiers de Schnorr. Nous ne rentrerons pas en détail de ces problèmes, cela dépasse le cadre du projet.

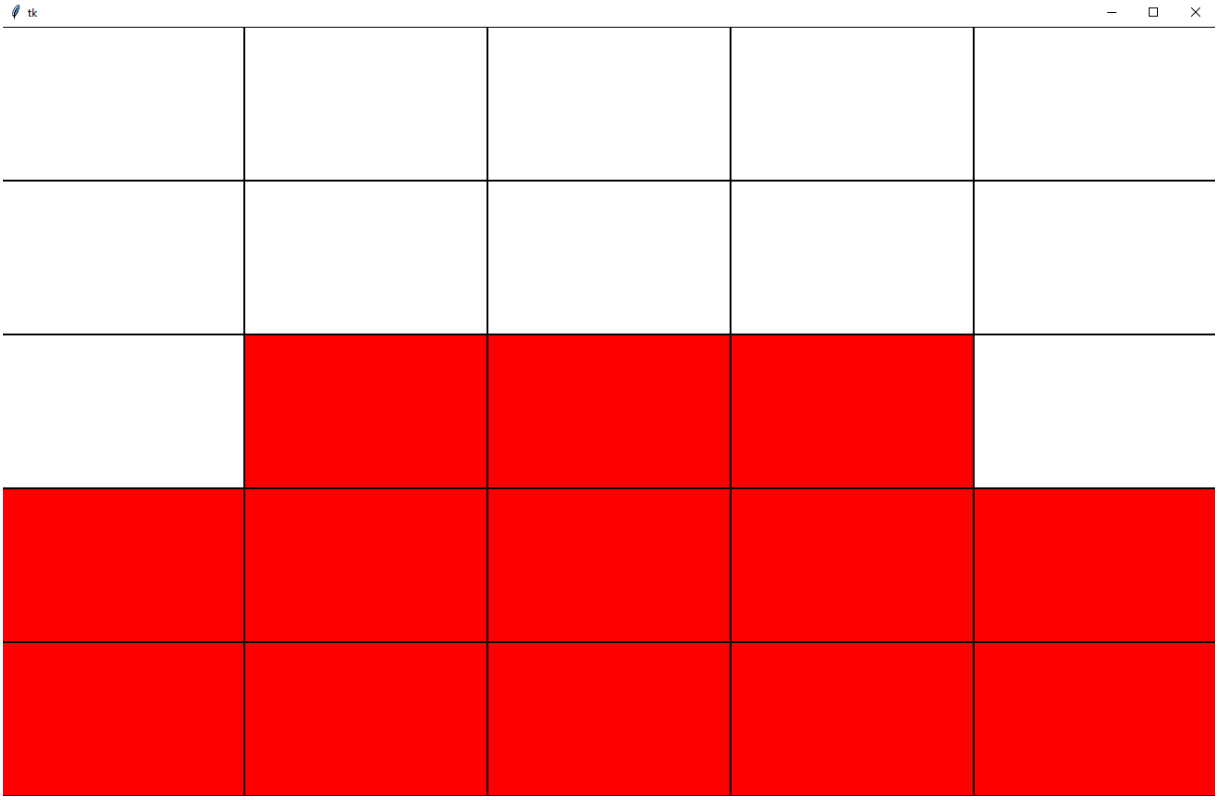
3 Modélisation de l'algorithme LLL par CFG

3.1 Description de la modélisation

Les deux modèles manipulent des piles de sables. Le modèle du tas de sable et celui de cfg fonctionnent sur le même principe, ils évoluent à chaque étape en distribuant une certaine quantité de sable aux piles voisines pendant le déroulement de la modélisation. Cependant, comme nous allons le voir dans la suite, ils ne suivent pas le même fonctionnement (les mêmes règles). Les piles de sable sont en réalité représentées par des nombres réels, que l'on note : q_i pour les tas de sable et c_i pour les cfg. $Q_d(q, H, h)$, représente le modèle des tas de sable et $C_d(c, H, h)$ celui des cfg. H et h sont deux réels positifs qui on le verra, ne possèdes pas les mêmes rôles dans les modèles.

3.1.1 Les tas de sables

Pour mieux comprendre, le fonctionnement des tas de sables expliquer clairement dans la thèse de doctorat de Mariya Georgieva, nous avons en premier lieu représenté très simplement le modèle graphiquement avec des entiers. Ce qui nous a permis de visualiser le fonctionnement du modèle des tas de sables avec un algorithme relativement naïf fonctionnent seulement avec des entiers.



Capture de notre modélisation graphique de l'algorithme tas de sable.

Afin d'éviter d'être barbant, nous n'allons pas expliquer l'algorithme à travers le code de l'algorithme mais plutôt à partir du fonctionnement présenté dans la thèse. La configuration initiale des piles $q = (q_1, \dots, q_d) \in R^d$.

La fonction de transition à chaque état est définie par :

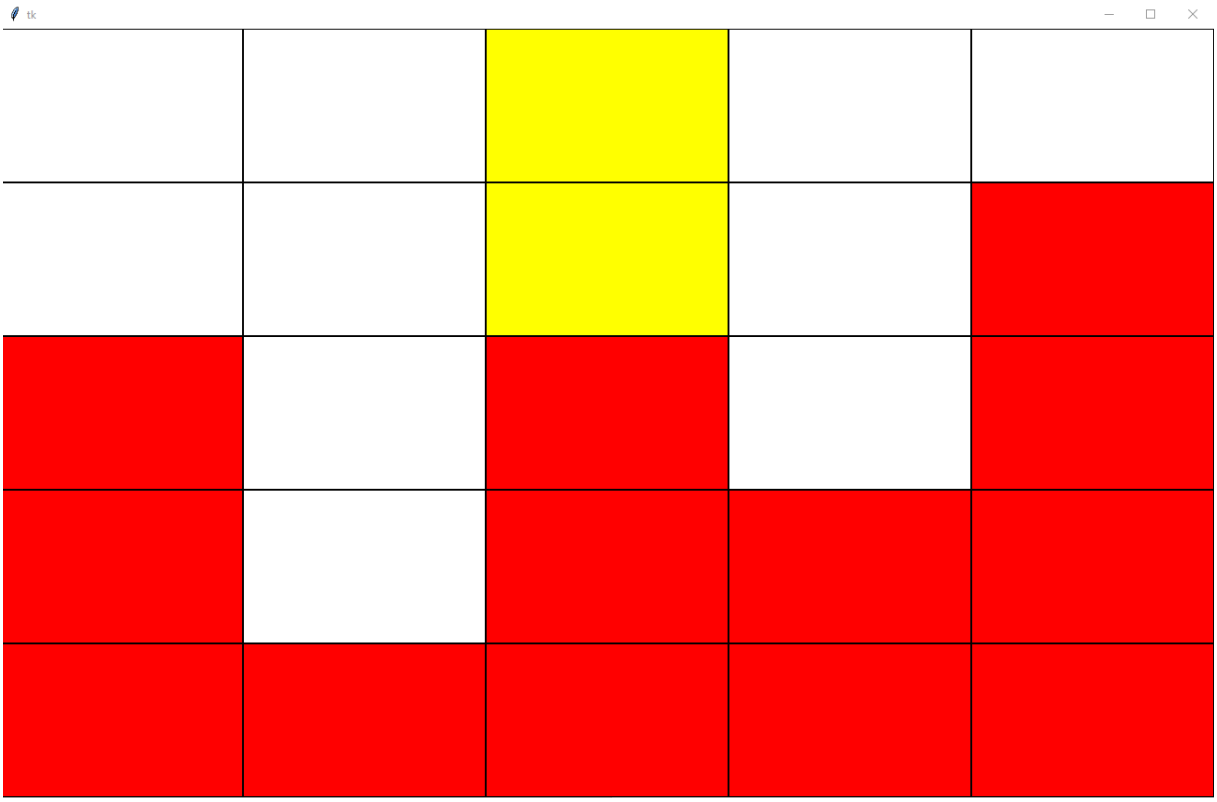
$$\check{q_j} = \begin{cases} q_j - h & si\ j = i\ et\ q_i - q_{i+1} > H, \\ q_j + h & si\ j = i+1\ et\ q_i - q_{i+1} > H, \\ q_j & sinon. \end{cases}$$

Fonction de transition présente dans la thèse

Cette formule définit clairement le fonctionnement de l'algorithme naïf utilisé dans la compréhension du fonctionnement des tas de sables. En effet, si la différence de hauteur entre q_i et $q_{i+1} >$ H et $j = i$. Alors on considère que la pile j est trop haut (elle ne respecte pas la condition de différence maximal H), nous retirons la hauteur h à la pile j. Sur le même principe, si la différence de hauteur entre q_i et $q_{i+1} >$ H et $j = i + 1$. On considère que la pile j manque de sable, nous ajoutons alors une quantité de sable h à la pile j. Sinon la pile reste inchangée. L'algorithme s'arrête quand aucune pile n'est plus grande que H d'une pile voisine.

3.1.2 Chip firing game

Pour les mêmes raisons, nous avons représenté très simplement le modèle graphiquement avec des entiers. Ce qui nous a permis de visualiser le fonctionnement du modèle des cfg avec un algorithme naïf fonctionnent aussi seulement avec des entiers.



Capture de notre modélisation graphique de l'algorithme naïf cfg en cours d'exécution.

De manière analogue, nous allons expliquer le fonctionnement de l'algorithme. La configuration initiale des piles $c = (c_1, \dots, c_{d-1}) \in R^{d-1}$.

La fonction de transition à chaque état est définie par :

$$\check{c}_j = \begin{cases} c_j - 2h & si\ j = i\ et\ c_i > H, \\ c_j + h & si\ j \in \{i-1, i+1\}\ et\ c_i > H, \\ c_j & sinon. \end{cases}$$

Fonction de transition présente dans la thèse

Dans ce cas, si $j = i$ et $c_i > H$, la pile ne respecte pas la contrainte de la hauteur maximale d'une pile. Alors on retire 2 fois la quantité de sable h à la pile j pour la distribuée la quantité h à au deux piles voisines $c_{i-1}\ et\ c_{i+1}$. Dans le cas ou i = 1 ou i = d – 1 on se retrouve en bordure. Donc une partie h du sable va disparaitre à gauche ou a droite tout dépend dans quel cas nous sommes. Donc, à la différence des tas de sable, l'évolution du système engendre la diminution de la taille des piles. L'algorithme s'arrête quand toutes les piles sont inferieur ou égale à H.

3.2 Modélisation de l'algorithme LLL à l'aide du CFG

Nous rappelons qu'un Chip Firing Game (cfg) est un modèle $C_d(c, H, h)$ qui modélise les évolutions des configurations en fonction d'une configuration initiale $c = (c_1, \dots, c_{d-1}) \in \mathbb{R}^{d-1}$ sous l'action d'une famille de fonctions de transition. Une fonction de transition $g\_i : c \to \hat{c}$ est définie comme suit :

$$\widehat{c_j} = \begin{cases} c_j - 2h \text{ si } j = i \text{ et } c_j > H, \\ c_j + h \text{ si } j \in \{i-1, i+1\} \text{ et } c_i > H, \\ c_j \text{ sinon} \end{cases}$$

Il est aisé de remarquer que le Chip Firing Game s'adapte extrêmement bien au cas de l'algorithme LLL. En effet, dans le cadre de l'algorithme LLL ce sont les rapports de Siegel qui jouent un rôle fondamental dans la progression de l'algorithme car on cherche les maximiser au maximum. Ce sont donc ces rapports qu'on va chercher à modéliser dans le cadre de notre projet. En effet, la modification du rapport $r_i = \frac{l_{i+1}}{l_i}$ entraîne également la modification des rapports r_{i-1} et r_{i+1} avec un facteur de décrémentation p. Les nouvelles valeurs des rapports sont donc :

$$\begin{cases} \hat{r}_{i-1} = p.\hat{r}_{i-1} \\ \hat{r}_i = \dfrac{1}{p^2}.\hat{r}_i \\ \hat{r}_{i+1} = p.\hat{r}_{i+1} \end{cases}$$

Ces nouveaux rapports ressemblent beaucoup aux fonctions de transitions du Chip Firing Game. Cependant la transition entre les rapports est multiplicative et donc difficile à représenter graphiquement. On va chercher à rendre cette transition additive comme dans le CFG.

3.2.1 Point de vue additif

Pour chercher à modéliser les transitions de l'algorithme LLL on va donc utiliser la fonction logarithmique. Ainsi la base B à réduire est représentée graphiquement par les logarithmes des rapports de Siegel. La base B est alors représentée par $c := (c_1, \dots, c_{d-1})$ $avec$ $1 \leq i < d$.

$$\text{Avec } c_i = -\log_s ri = -\log_s \frac{l_{i+1}}{l_i}$$

Avec s le paramètre de Siegel qui est choisi dans l'intervalle $]S_0, +\infty]$ avec $S_0 = {}^{2}/_{\sqrt{3}}$ qui est la valeur habituellement choisie. On rappelle que la condition d'arrêt d'un point de vue multiplicative est $r_i \geq 1/s$. La condition d'arrêt de Siegel d'un point de vue additif est donc $c_i \leq 1$.

L'étape d'échange entre les vecteurs b_i et b_{i+1} d'un point de vue additif modélisé par le CFG devient donc :

$$\begin{cases} \hat{c}_{i-1} = \hat{c}_{i-1} + \alpha \\ \hat{c}_i = \hat{c}_i - 2\alpha \\ \hat{c}_{i+1} = \hat{c}_{i+1} + \alpha \end{cases}$$

Où α est le facteur de décrémentation de l'algorithme LLL qui est égal à :

$$\alpha = -\frac{1}{2}\log_s(s^{-2c_i} \times {m_{i+1,i}}^2)$$

3.2.2 Modèles de modélisations.

Il existe plusieurs choix pour modéliser la progression de l'algorithme LLL. Le modèle, qu'on appelle M1, est un modèle dans lequel le décrément α est fixé au début de l'algorithme et ne change pas. Le modèle M2 est un modèle dans lequel α_i se calcule en fonction de c_i et v_i, ce dernier est fixé dans l'intervalle $\left]-\frac{1}{2},\frac{1}{2}\right]$. Dans le modèle M3, α_i est calculé en fonction de c_i et v_i, qui varie dans $\left]-\frac{1}{2},\frac{1}{2}\right]$ cette fois. Les modèles M4 et M5 sont ceux qui modélisent véritablement l'algorithme LLL. Dans le modèle M4, celui qu'on modélisera dans le projet, l'algorithme fonctionne à partir de la condition de Siegel ($c_i \leq 1$). Le Modèle M5 est encore plus réaliste car dans celui-ci la condition d'arrêt c_i est calculé en fonction de la pile.

Il existe donc plusieurs moyens de modéliser l'algorithme LLL. En effet, on peut par exemple choisir i, l'indice de la pile c_i sur lequel l'algorithme LLL travaille, aléatoirement ou en fonction de la taille de la pile. Dans notre cas, i est choisi de manière classique. On peut, de plus, choisir le mode de calcul de α. En effet, il peut être fixé dès le début, comme c'est le cas dans le modèle M1, ou sinon calculé en fonction de la pile, comme c'est le cas dans le modèle M4, qu'on représente dans le projet.

3.3 Fonctionnement de l'application

Notre application fonctionne à l'aide de 4 classe distincts (LLL, modelisationCFG, modelisationOFLLL et parserData) que nous allons expliquer dans cette partie.

3.3.1 Parseur de données

Pour commencer par le plus simple, nous avons d'abord un parser de données : parserData.py qui nous permet de lire des bases se trouvant dans des fichiers. Ce fichier est composé en deux définitions : parser et parser2 ; Etant donné que les fichiers ne respectent pas la même syntaxe.

```
from fractions import Fraction
import numpy as np
from numpy import long


def parser(file_name):...

def parser2(file_name):...
```

Fichier parserData tronqué contenant les 2 defs

Les deux fichiers retournent respectivement une arraylist contenant la base lue en entrée.

3.3.2 ModélisationCFG

Ensuite, la classe modélisationCFG fait office de jonction entre la modélisation et l'algorithme LLL. Nous l'avons mise en place pour facilité la compréhension de notre code (en l'épurant). Cette classe contient donc une simple initialisation de la partie visuel en appelant la classe modélisationofLLL que nous verrons plus tard, une methode update pour mettre à jour notre modélisation avec en paramètre le alpha courant, la liste, i (la pile modifié) et un boolean . Et une méthode permettant d'initier la liste des hauteurs des piles.

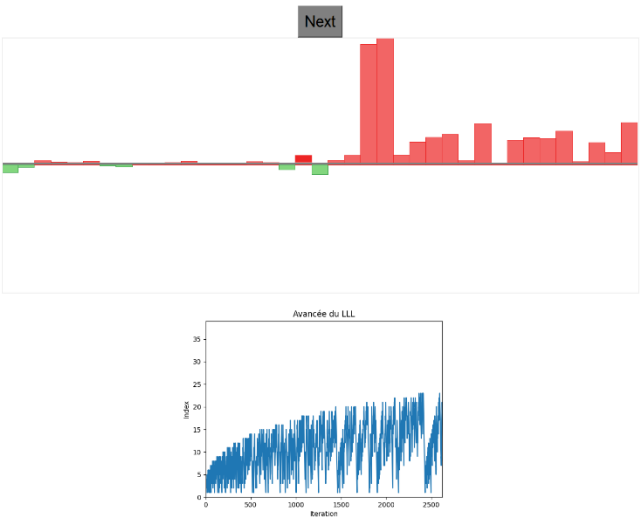
```
class ModelisationCFG:

    def __init__(self,ci_list):
        self.ci_list = ci_list
        self.application = modelisationOfLLL(2000,800,1)
        self.application.creation_frame(self.ci_list)
        self.iterator = 0
        self.list_i =[0]

    def setCiList(self,list):
        self.ci_list =list

    def update(self,i,alpha,continu):
        self.iterator += 1
        self.list_i.append(i+1)
        self.application.update_frame(self.ci_list,i,self.list_i,self.iterator,alpha,continu)
```

Classe modélisationCFG

3.3.3 ModélisationLLL

Pour continuer, on a notre classe modélisationofLLL qui s'occupe de l'affichage à chaque étape de l'avancée de notre algorithme.

Illustration de l'algorithme en fonctionnement :



Illustation de l'algorithme lll en fonctionnement

Comme nous pouvons le voir dans cette illustration, l'affichage graphique est composé deux parties, une représentant les rapports de Siegel avec un CFG (à travers des piles). Nous pouvons remarquer que la pile en cours de modification est en rouge foncé sur l'image (en fonction de la hauteur de alpha). Et l'autre graphique, représente l'avancée de l'indice i dans la base à réduire dans l'algorithme LLL.

Pour ce faire, nous avons dans cette classe, une methode new_canvas qui initialise l'affichage du premier graphique en fonction de la hauteur de chaque pile. Une méthode newplot qui initialise le deuxième graphique. Une méthode creationframe qui fait le lien entre les 2. Et de manière analogue, une méthode updatemodelisation, updateplot et updateframe qui s'occupe de maintenir à jour l'avance de la modélisation en fonction de la hauteur de chaque pile et du alpha courant. Nous avons d'autre petite méthode annexe qui permettent de gérer le confort visuel de la fenêtre et interactif avec l'utilisateur.

```
def ratio(self):...
#Création d'une figure
def new_plot(self):...
#Création d'un canvas comportant chaque pile
def new_canvas(self, heights_list):...
#Creation frame
def creation_frame(self, heights_list):...
#Mise à jour du canvas
def update_modelisation(self, new_heights_list, position,alpha,continu):...
# Mise à jour du plot
def update_plot(self, list_i, iterator):...

# Update frame
def update_frame(self, new_heights_list, position, list_i, iterator,alpha,continu):...
```

Méthodes utilisées dans la classe modelisationOfLLL

Pour finir, nous avons la classe LLL. Cette classe s'occupe du fonctionnement de l'algorithme. Nous avons à l'intérieur de cette dernière 8 méthode. La méthode dot permet tout simplement d'effectuer le produit scalaire entre 2 vecteurs. La méthode m est utilisé dans la projection d'un vecteur. La méthode projection comme son nom l'indique renvoie la projection scalaire d'un vecteur sur un autre. La méthode Gram-Schmidt permet d'orthogonaliser une base (voir partie 1). La méthode matrix quant à elle retourne le coefficient P[i,j] de la matrice de passage paramètre. La méthode listCiConstruction construit la liste des c_i pour la modélisation graphique. La méthode alpha calcule le coefficient α pour chaque pile à chaque itération. Pour finir, la méthode LLL_algorithm représente le fonctionnement de l'algorithme.

```
class LLL:
    def __init__(self,basis,param,base_log):...

    def dot(self,u,v) -> long:...

    def m(self,u,v):...

    def projection(self,u,v) -> long:...


    def gram_schmidt(self):...


    def matrix(self,i,j):...


    def listCiConstruction(self):
        for i in range(len(self.orthogonalize_basis)-1):...


    def constructionAlpha(self,index):...

    def lll_algorithm(self):...
```

Méthodes utilisées dans la classe LLL

Pour conclure, ce rapport, nous avons avancé différents points clé de notre projet.

D'abord, nous avons étudié les réseaux à travers 3 parties, une concernant le fonctionnement des réseaux euclidiens, une autre exposant la réduction de réseau et la procédure de Gram-Schmidt et une dernière abordant des exemples d'utilisation des réseaux en cryptographie et en cryptanalyse. Ensuite, nous avons expliqué le fonctionnement de l'algorithme de A.Lenstra, H.Lenstra et L. Lovász. Pour finir, nous avons présenté la modélisation de l'algorithme LLL par CFG que nous avons pu coder durant ce projet afin de mieux assimiler le fonctionnement de l'algorithme à travers un exemple concret.

Nous avons vraiment apprécié ce projet. En effet, nous avons pu approfondir nos connaissances sur les réseaux et surtout voir une infime partie de l'utilité de ce domaine à travers l'algorithme LLL. De plus, nous avons pu voir des exemples d'applications de l'algorithme LLL (qui peut paraître complexe à première vue) avec la modélisation par CFG et son utilité dans la cryptanalyse de certains algorithmes. Néanmoins, nous n'avons pas pu faire tout ce que nous voulions.

Effectivement, il serait intéressant plus tard d'ajouter pour la modélisation de nouveaux graphiques comme nous pouvons le voir dans la thèse de Mariya Georgieva. En ce qui concerne l'algorithme, nous aurions aussi aimé adopter plusieurs stratégies différentes : une stratégie gloutonne ou aléatoire.

Bibliographie

Principale source d'information : Thèse de doctorat par Mariya Georgieva, sous la direction de Brigitte Vallée

Autres sources d'information : Page Web de l'algorithme LLL

Page Web de Gram-Schmidt

Page Web d'un réseau et réduction de réseau

Cours PDF (ENSICAEN) de M. LHOTE : Modélisation de l'algorithme LLL et de ses entrées par des systèmes dynamiques