UNIVERSITAT DE
BARCELONA

Treball final del grau de Matemàtiques
Facultat de Matemàtiques i Informàtica

LLL. ALGORITME DE REDUCCIÓ DE BASES DE XARXES

Pere-Lluís Huguet Cabot

Director: Dr. Artur Travesa Grau
Departament de Matemàtiques i Informàtica
Barcelona, juny 2017

Abstract

The algorithm LLL is a strong tool for reducing lattice bases in polinomical time introduced by Arjen Lenstra, Hendrik Lenstra and László Lovász in 1982. We will study it's implementation, as well as proof it's polinomical time behaviour. Finally, we will show it's use in factorizing factorizing polynomials with rational coefficients and some computational examples.

Resum

L'algoritme LLL és una eina potent per a reduir bases de xarxes en temps polinòmic introduït per Arjen Lenstra, Hendrik Lenstra i László Lovász el 1982. Estudiarem la seva implementació, així com provarem els seu caràcter polinòmic. Finalment, veurem el seu ús per a factoritzar polinomis de coeficients racionals així com algun exemple computacional.

Agraïments

Vull agrair a tots els companys que m'han ajudat a seguir tots aquests anys, els de la universitat i els de fora. Agrair també al Dr. Artur Travesa pel seguiment d'aquest treball i l'empenta per a que sigui el que avui tenen al davant. Per últim, a la meva família, sense la qual seria impossible ser aquí.

Índex

1 Introducció **1**

2 Conclusions **1**

3 Preliminars **2**

4 Xarxes **3**

4.1 Mínims successius 5

4.2 Problemes computacionals 7

4.3 Reducció de xarxes 9

4.3.1 Xarxes de dues dimensions 9

5 L'Algoritme LLL **11**

5.1 Algoritme 11

5.1.1 Fase de reducció 11

5.1.2 Fase de d'intercanvi 13

5.1.3 Anàlisi de la sortida 13

5.2 Temps de computació 13

5.2.1 Iteracions 14

5.2.2 Mida dels vectors 15

5.3 Implicacions 17

6 Anàlisi computacional **19**

6.1 Temps 19

6.2 Mida 20

7 Aplicació a altres algoritmes de reducció de bases de xarxes **22**

8 Factorització de polinomis de coeficients enters **24**

8.1 Descomposició d'un polinomi en factors isorreductibles 24

8.2 Descomposició dels polinomis isorreductibles 25

8.3 Aixecament de Hensel 27

8.4 Algoritme de factorització de polinomis de coeficients enters 29

8.4.1 LLL per a factorització 30

1 Introducció

L'algoritme LLL és un dels algoritmes més rellevants dels últims anys. Introduït per Arjen Lenstra, Hendrik Lenstra i László Lovász el 1982, d'aquí el seu nom, per les inicials de qui el va introduir. Pensat inicialment per a factoritzar polinomis de coeficients enters, la seva eficiència per a reduir bases de xarxes ha portat a que sigui utilitzat molt més enllà.

El seu temps de computació polinòmic suposava la primera implentació de temps polinòmic per a factoritzar polinomis de coeficients enters. Alhora, el seu caràcter determinista permet ser utilitzat per a resoldre problemes que abans es consideraven difícils.

En aquest treball veurem en què consisteix l'algoritme LLL així com la seva principal aplicació per a factoritzar polinomis de coeficients enters. Demostrarem el seu caràcter polinòmic i analitzarem la seva sortida. També veurem quines implicacions té l'algoritme, així com la seva utilització en altres casos.

En acabat, farem una breu anàlisi computacional de la seva utilització, iterant el seu ús sobre diferents casos amb python i SageMath.

2 Conclusions

El vist durant tot aquest treball ens mostra el potencial de l'algoritme LLL, així com la complexitat que pot arribar a suposar quelcom aparentment simple com a buscar el vector més curt d'una xarxa, i les limitacions computacionals que mostra. Si l'algoritme LLL és el millor algoritme de temps polinómic per a fer-ho, ens dóna una bona fita de fins a on podem arribar, així com un punt de partida per a noves implementacions.

Alhora veiem que tot i que la seva aplicació directa consisteix en la reducció de xarxes, existeixen força més implementacions. Hem vist com utilitzar l'algoritme per a la factorització de polinomis de coeficients enters, una de les aplicacions més conegudes i utilitzades. També hem vist el seu ús en altres algoritmes.

3 Preliminars

A tot el treball, i llevat que en algun punt t'indiqui quelcom contrari, treballarem a l'espai euclidià n-dimensional, $\mathbb{R}^n$, prenen com a base de l'espai la base canònica ortonormal, e_i, amb el producte escalar ordinari

$$\langle e_i, e_j \rangle = \delta_{i,j} = \begin{cases} 1, \text{ si } i = j, \\ 0, \text{ si } i \neq j, \end{cases}$$

i, per a $x, y \in \mathbb{R}^n$,

$$\langle x, y \rangle = (x_1, \ldots, x_n)(y_1, \ldots, y_n) = x_1 y_1 + \cdots + x_n y_n.$$

Donats uns vectors qualssevol $b_1, \ldots, b_j \in \mathbb{R}^n$, indicarem per $\langle b_1, \ldots, b_j \rangle_{\mathbb{R}}$ el subespai generat per $b_1, \ldots, b_j$.

3.1 (Procés d'ortogonalització de Gram-Schmidt)**.** Sigui $B := \{b_1, \ldots, b_n\}$ una $\mathbb{R}$-base de $\mathbb{R}^n$. Els vectors $\tilde{b}_1, \ldots, \tilde{b}_n \in \mathbb{R}^n$ definits recursivament per

$$\tilde{b}_i = b_i - \sum_{j=1}^{i-1} \mu_{i,j} \tilde{b}_j, \qquad \mu_{i,j} = \frac{\langle b_i, \tilde{b}_j \rangle}{\langle \tilde{b}_j, \tilde{b}_j \rangle},$$

formen una base ortogonalitzada $\tilde{B} = \{\tilde{b}_1, \ldots, \tilde{b}_n\}$ de $\mathbb{R}^n$, que s'anomena ortogonalització de Gram-Schmidt de B. Si a més a més normalitzem la base ortogonal, obtenim $\tilde{B}' = \{\frac{\tilde{b}_1}{\|\tilde{b}_1\|}, \ldots, \frac{\tilde{b}_n}{\|\tilde{b}_n\|}\}$, base ortonormal.

El coeficient $\mu_{i,j}$ és la projecció de b_i en $\tilde{b}_j$ dividit per la norma de $\tilde{b}_j$.

4 Xarxes

En aquesta secció introduïrem les eines necessàries sobre xarxes per a la resta del treball. Per començar, hem de veure què és una xarxa.

Definició 4.1 (Xarxa a $\mathbb{R}^n$)**.** Una xarxa de $\mathbb{R}^n$ és un subgrup abelià lliure de $\mathbb{R}^n$ i de dimensió n. Si $\{b_1, \ldots, b_n\}$ n'és una base, la podem escriure com

$$\mathcal{L}(b_1, \ldots, b_n) := \bigoplus_{i=1}^{n} \mathbb{Z}b_i = \left\{ \sum_{i=1}^{n} \alpha_i b_i \,|\, \alpha_i \in \mathbb{Z} \right\},$$

indistintament. Recíprocament, si $\{b_1, \ldots, b_n\}$ és una base de $\mathbb{R}^n$, llavors $\mathcal{L}(b_1, \ldots, b_n)$ és una xarxa.

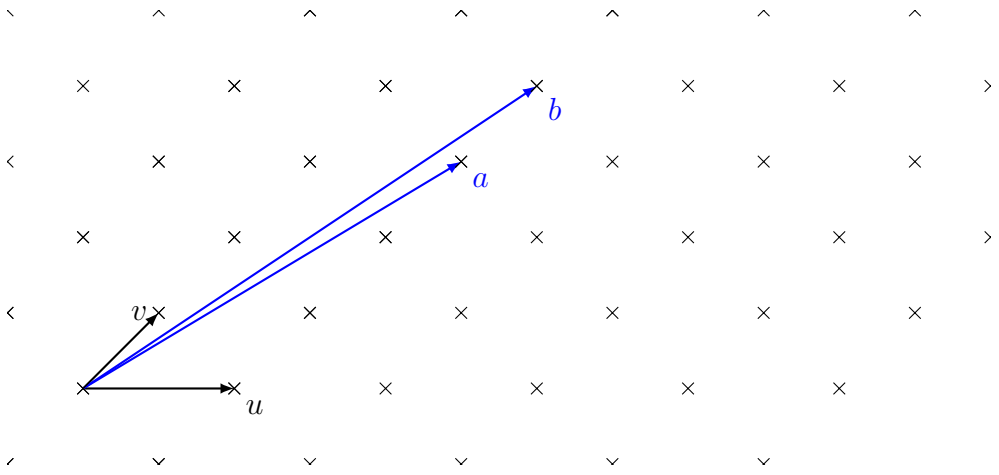

Figura 1: Xarxa a $\mathbb{R}^2$, que anomenem $\mathcal{L}_1$, amb dues bases $B_1 = (u, v), B_2 = (a, b)$

A la Figura 1 tenim dues bases diferents que generen la mateixa xarxa. Això permet classificar les bases segons diferents paràmetres. Per exemple, si ens fixem en la figura, podem veure que els vectors u, v són més curts que a, b, i ens interessarà treballar amb bases al més curtes possible. Si tenim dues bases de la mateixa xarxa, sempre podrem obtenir una a partir de l'altra, però trobar la base adequada no sempre serà fàcil. A la figura, $4a - 3b = u$ i $b - a = v$. Trobar aquesta base "petita" és un objectiu de la reducció de xarxes. Més endavant precisarem què vol dir "petita".

Per saber si un conjunt de vectors és una base, farem servir el concepte de cel·la i veurem certes propietats relacionades.

Definició 4.2 (Cel·la dels vectors $x_1, \ldots, x_n$)**.**

$$\mathcal{C}(x_1, \ldots, x_n) = \{ \sum_{i=1}^{n} \alpha_i x_i \mid \alpha \in \mathbb{R}, 0 \leq \alpha_i < 1 \}$$

Definició 4.3. Si els vectors formen una base de $\mathcal{L}(B)$, la cel·la és un paral·lelepípede de dimensió n; s'anomena paral·lelepípede fonamental,

$$\mathcal{P}(b_1, \ldots, b_n) = \{ \sum_{i=1}^{n} \alpha_i b_i \mid \alpha \in \mathbb{R}, 0 \leq \alpha_i < 1 \}$$

Proposició 4.1. *Prenem $\mathcal{L}$ com a xarxa de rang n i $b_1, \ldots, b_n \in \mathcal{L}$, n vectors linealment independents. Llavors $b_1, \ldots, b_n$ formen una base de $\mathcal{L}$ si i només si $\mathcal{P}(b_1, \ldots, b_n) \cap \mathcal{L} = \{0\}$.*

Demostració. Suposem primer que $b_1, \ldots, b_n$ formen una base de $\mathcal{L}$. Per definició, la xarxa és un conjunt format per els vectors de coeficients enters, i $\mathcal{P}(b_1, \ldots, b_n)$ pels vectors amb coeficients a $[0, 1)$, per tant $\mathcal{P}(b_1, \ldots, b_n) \cap \mathcal{L} = \{0\}$.

Si suposem ara que $\mathcal{P}(b_1, \ldots, b_n) \cap \mathcal{L} = \{0\}$, podem escriure qualsevol vector $x \in \mathcal{L}$ com a combinació lineal dels vectors $b_1, \ldots, b_n$, $x = \sum_{i=1}^{n} y_i b_i$ amb $y_i \in \mathbb{R}$. Per la definició de xarxa, $x' = \sum_{i=1}^{n} (y_i - \lfloor y_i \rfloor) b_i \in \mathcal{L}$. Però $(y_i - \lfloor y_i \rfloor) \in [0, 1)$ i per tant la nostra hipòtesis implica que $x' = 0$, i per tant tots els y_i són enters i x és una combinació lineal de coeficients enters de $b_1, \ldots, b_n$. □

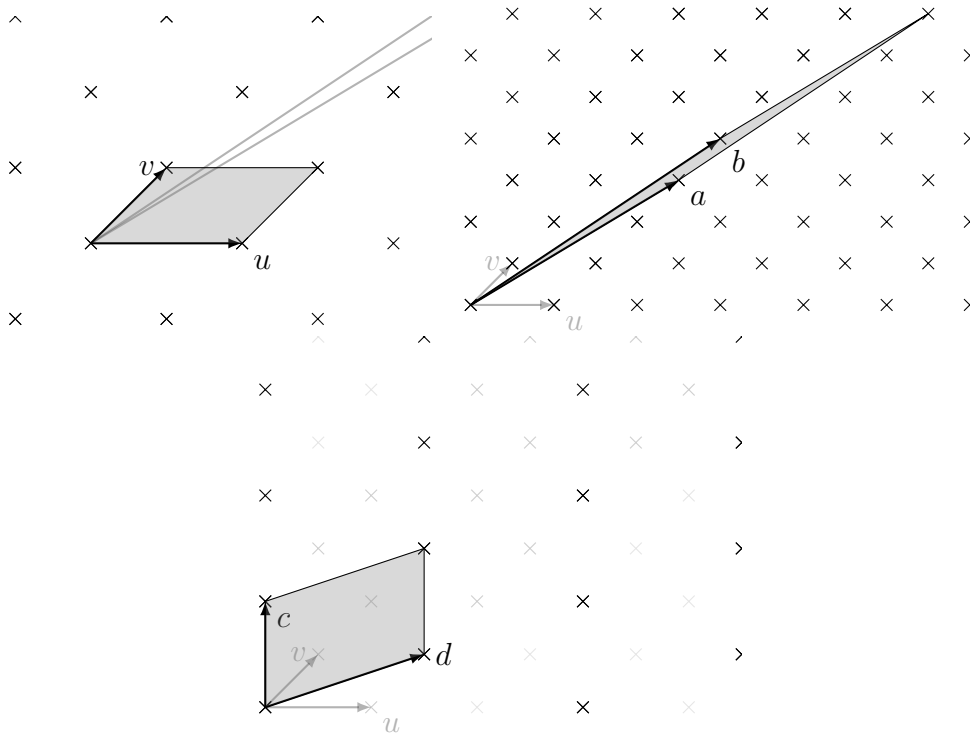

Figura 2: Les dues cel·les generades per les bases de la Figura 1 (a sobre) i una cel·la per als vectors c i d que no formen una base (a sota)

Com podem veure a la figura 2 les cel·les de les imatges superiors no contenen cap punt de la xarxa, i els seus vectors formen bases, en canvi per la cel·la a la imatge inferior, a la cel·la hi ha dos punts de la xarxa $\mathcal{L}$, i els seus vectors són base d'una xarxa diferent.

Definició 4.4 (Determinant d'una xarxa)**.**

$$det(\mathcal{L}) = \prod \left\| \tilde{b}_i \right\| = \sqrt{det(B^T B)}.$$

Si dues bases $B_1, B_2 \in \mathbf{M}(n, \mathbb{R})$ generen la mateixa xarxa, és a dir que són equivalents, existeix $U \in GL(n, \mathbb{Z})$, tal que $B_1 = UB_2$.

Proposició 4.2. *El determinant d'una xarxa no depèn de la seva base.*

Demostració. Si tenim dues bases B_1, B_2 de la mateixa xarxa, existeix $U \in GL(n, \mathbb{Z})$ tal que $B_1 = UB_2$. Per tant, per la definició de determinant d'una xarxa, $det(\mathcal{L}(B_1) = |det(B_1)| = |det(U)det(B_2)| = |det(B_2)|$. □

4.1 Mínims successius

Un altre paràmetre important a les xarxes és la mida (euclidiana) del vector no nul més curt. Estarà expressat per λ_1 i en el cas de la xarxa de l'exemple anterior, seria el vector v. Això ens porta a definir el concepte de mínims successius.

Definició 4.5 (Mínims successius)**.** Si tenim una xarxa $\mathcal{L}$ de $\mathbb{R}^n$, per a $1 \leq i \leq n$, definim i-èsim mínim successiu com

$$\lambda_i(\mathcal{L}) = inf\{r \mid dim(\langle \mathcal{L} \cap \overline{B}(0, r)\rangle_{\mathbb{R}}) \geq i\},$$

on $\overline{B}(0, r) = \{x \in \mathbb{R}^n | \|x\| \leq r\}$ és la bola tancada de radi $r \in \mathbb{R}$ i centrada al 0.

La definició ens diu doncs que els mínims successius són els radis de les boles més petites que contenen com a mínim i vectors linealment independents de $\mathcal{L}$. Amb això podem enunciar el teorema següent que ens permet donar una cota inferior del vector més curt de la xarxa.

Proposició 4.3. *Si tenim B, base d'una xarxa de dimensió n, i $\tilde{B}$ la seva ortogonalització de Gram-Schmidt, aleshores,*

$$\lambda_1(\mathcal{L}(B)) \geq \min_{i=1,\ldots,n} \|\tilde{b}_i\| > 0.$$

Demostració. Tenim $x \in \mathbb{Z}^n$ un vector d'enters, hem de veure que $\|Bx\| \geq \min \|\tilde{b}_i\|$. Prenem el valor $j \in \{1, \ldots, n\}$ més gran tal que $x_j \neq 0$. Llavors,

$$|\langle Bx, \|\tilde{b}_j\| \rangle| = |\sum_{i=1}^{j} \langle x_i b_i, \tilde{b_j}\rangle| = |x_j| \langle \tilde{b_j}, \tilde{b_j}\rangle = |x_j| \|\tilde{b_j}\|^2$$

on hem utilitzat que $\forall i < j$, $\langle b_i, \tilde{b}_j\rangle = 0$ i que $\langle b_j, \tilde{b_j}\rangle = \langle \tilde{b_j}, \tilde{b_j}\rangle$. També tenim que $|\langle Bx, \tilde{b_j}\rangle| \leq \|Bx\| \cdot \|\tilde{b}_j\|$ i per tant:

$$\|Bx\| \geq |x_j| \|\tilde{b_j}\| \geq \|\tilde{b}_j\| \geq min \|\tilde{b}_i\|.$$

□

Un cop tenim una fita inferior en buscarem una de superior. És de destacar la feina de Minkowski al respecte, que també establí alguns resultats importants que són utilitzats i ben coneguts al camp de la teoria de nombres als teoremes de finitud com la finitut del grup de classes d'ideals. No obstant, el conegut com a teorema de Minkowski del cos convex, no deixa de ser un col·lorari del menys conegut teorema de Blichfeldt, que enunciem a continuació.

Teorema 4.4 (Teorema de Blichfeldt). *Si S és un conjunt mesurable tal que $vol(S) > det(\mathcal{L})$, existeixen dos punts diferents $z_1, z_2 \in S$ tals que $z_1 - z_2 \in \mathcal{L}$.*

Demostració. Podem cobrir el conjunt S per un nombre suficient de còpies del paral·lelepípede fonamental. Si considerem la intersecció d'aquestes amb S i les transladem a l'origen, com que $vol(S) > det(\mathcal{L})$, existirà un solapament entre alguna de les translacions, d'on podem obtenir un punt z, que respectivament serà $z_1 = z + x_1$ i $z_2 = z + x_2$ a S, amb $x_1, x_2 \in \mathcal{L}$ punts de la xarxa d'on provenen les interseccions, i la diferència entre ambdós $z_1 - z_2 = x_1 - x_2 \in \mathcal{L}$. A la figura 3 es veu clarament. □

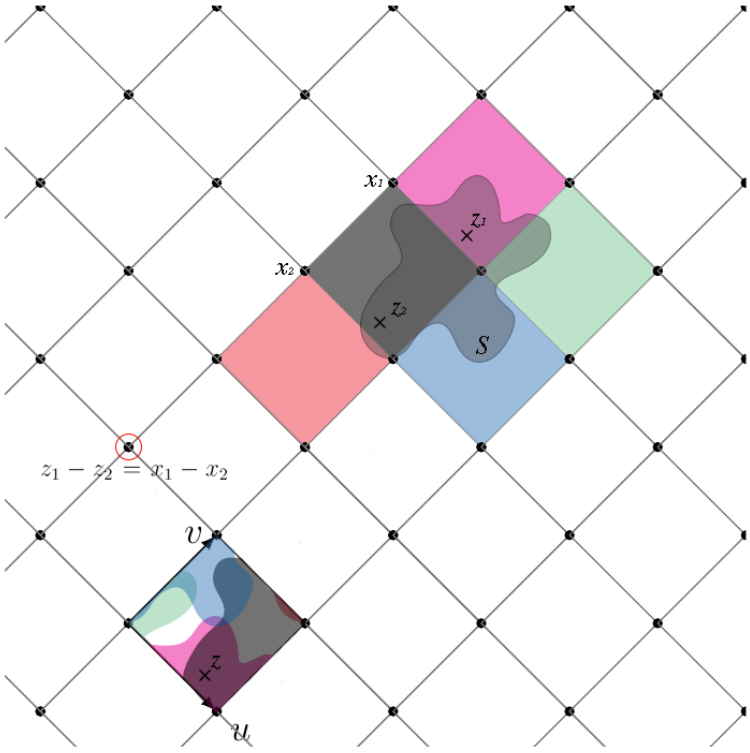

Figura 3: Teorema de Blichfeldt

Teorema 4.5 (Teorema de Minkowski del cos convex). *Prenem $\mathcal{L}$ xarxa de rang n. Llavors per a qualsevol conjunt S convex amb simetria central, si $vol(S) > 2^n det(\mathcal{L})$, S conté un punt no nul de la xarxa.*

Demostració. Definim $\tilde{S} = \frac{1}{2}S = \{x \,|\, 2x \in S\}$. Llavors, $vol(\tilde{S}) = 2^{-n} vol(S) > det(\mathcal{L})$. Pel teorema de Blichfeldt, existeixen punts $z_1, z_2 \in \tilde{S}$ tals que $z_1 - z_2 \in \mathcal{L}$

és un punt no nul de la xarxa. Per definició, $2z_1, 2z_2 \in S$ i com que S té simetria central, també $-2z_2 \in S$. Finalment, per ser S convex, $\frac{2z_1-2z_2}{2} = z_1 - z_2$ pertany a S. □

Teorema 4.6 (Primer teorema de Minkowski)**.** *Per a qualsevol xarxa $\mathcal{L}$ de rang n,*

$$\lambda_1(\mathcal{L}) \leq \sqrt{n}(det\mathcal{L})^{1/n}.$$

Demostració. Per definició, la bola oberta $B(0, \lambda_1(\mathcal{L}))$ no conté cap vector no nul. Pel teorema anterior i com que tenim que $vol(B(0,r)) \geq (\frac{2r}{\sqrt{n}})^n$,

$$(\frac{2\lambda_1(\mathcal{L})}{\sqrt{n}})^n \leq vol(B(0, \lambda_1(\mathcal{L}))) \leq 2^n det(\mathcal{L}),$$

d'on obtenim que la desigualtat és $\lambda_1(\mathcal{L}) \leq \sqrt{n}(det\mathcal{L})^{1/n}$. □

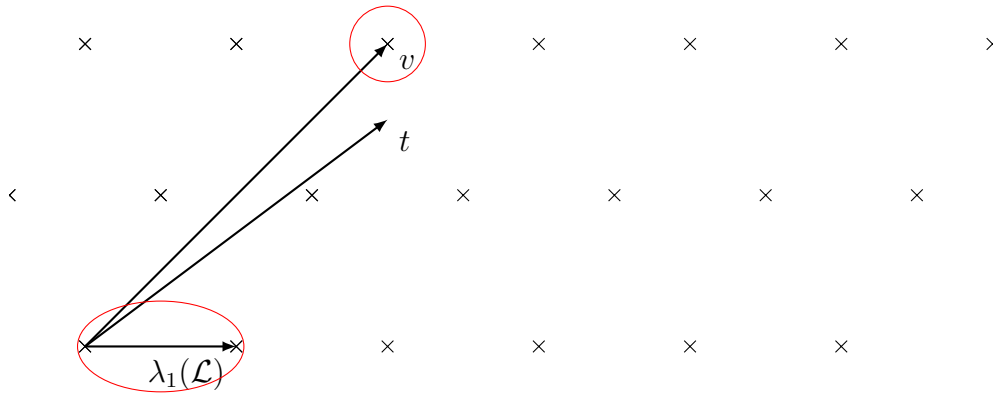

Figura 4: Representació del problemes SVP i CVP

4.2 Problemes computacionals

Donada una xarxa $\mathcal{L}$ els vectors estan fitats inferiorment per la mida dels vectors resultants de l'ortogonalització de Gram-Schmidt i superiorment a partir del primer teorema de Minkowski, però no tenim forma de trobar els mínims successius, i en concret el vector més curt λ_1. Això ens porta a enunciar diversos problemes computacionals amb moltes implicacions, ja que es conjectura que són de complexitat no polinòmica.

Problema 4.1. Un dels problemes més coneguts és el conegut com a problema del vector més curt (shortest vector problem) i denotat usualment per les seves inicials en anglès, SVP. Hi ha tres variants d'aquest problema. Consisteixen en:

Cerca SVP:	Donada una base B d'una xarxa, trobar $v \in \mathcal{L}(B)$ tal que $\|v\| = \lambda_1(\mathcal{L}(B))$.

Optimització SVP: Donada la base B d'una xarxa trobar $\lambda_1(\mathcal{L}(B))$.

Decisional SVP: Donades la base d'una xarxa B i un nombre racional $r \in \mathbb{Q}$, determinar si $\lambda_1(\mathcal{L}(B)) \leq r$ o no.

Com a problema computacional, caldrà treballar amb quantitats finites de bits; això fa que, d'entrada, i perquè els problemes són obviament equivalents per homotècies, puguem prendre els vectors de les bases B com a vectors de coordenades enteres expressables amb una quantitat finita (i fitada) de bits. La raó per treballar amb bases a $\mathbb{Z}^{n\times n}$ és que en considerar-los problemes computacionals, ens interessarà que siguin expressables en un nombre finit de bits.

Els tres problemes són equivalents, ja que es pot provar que cadascun és tan difícil com qualsevol dels altres. A continuació introduirem les variants *aproximades* d'aquests problemes.

Problema 4.2 (SVP_γ)**.**

Cerca SVP_γ: Donada la base d'una xarxa B trobar $v \in \mathcal{L}(B)$ tal que $v \neq 0$ i $\|v\| \leq \gamma \cdot \lambda_1(\mathcal{L}(B))$.

Optimització SVP_γ: Donada la base B d'una xarxa trobar $d\mathbb{R}$ tal que $d \leq \lambda_1(\mathcal{L}(B)) \leq \gamma \cdot d$.

Promesa SVP_γ: Donada una parella (B,r) on B és la base d'una xarxa i $r \in \mathbb{Q}$ un nombre racional, les instàncies positives són $\lambda_1(\mathcal{L}(B)) \leq r$ i les instàncies negatives $\lambda_1(\mathcal{L}(B)) > \gamma \cdot r$. L'objectiu és veure de quines instàncies prové l'entrada. Al contrari que els problemes de decisió, la unió dels conjunts de les dues instàncies no és el conjunt de totes les entrades possibles.

Aquest últim també s'anomena $GapSVP_\gamma$.

En aquest cas però, malgrat que tenim que el problema Promesa no és més difícil que el problema Optimització i aquest no ho és més que els problemes Cerca o Promesa, no està demostrat que el problema Cerca no sigui més difícil que el problema Optimització. De nou, imposarem les mateixes condicions com a problemes computacionals prenent bases formades per vectors de coordenades enteres expressables en una quantitat fitada de bits.

De forma semblant podem enunciar tres problemes més on l'objectiu és trobar el vector de la xarxa més proper a un punt donat de l'espai. Les variants aproximades s'enuncien com:

Problema 4.3. Un altre dels problemes més coneguts és el problema del vector més proper (closest vector problem) i denotat usualment per les seves inicials en anglès, CVP. Hi ha tres variants aproximades d'aquest problema. Consisteixen en:

Cerca CVP_γ: Donada la base d'una xarxa $B \in \mathbb{Z}^{n\times n}$ i el vector $t \in \mathbb{Z}^n$ trobar $v \in \mathcal{L}(B)$ tal que $\|v - t\| \leq \gamma \cdot dist(t, \mathcal{L}(B))$.

Optimització CVP_γ: Donada la base d'una xarxa $B \in \mathbb{Z}^{n\times n}$ i el vector $t \in \mathbb{Z}^n$ trobar $d \in \mathbb{R}$ tal que $d \leq dist(t, \mathcal{L}(B)) \leq \gamma \cdot d$.

Promesa CVP_γ: Donada la tríada (B,t,r) on $B \in \mathbb{Z}^{n\times n}$ és la base d'una xarxa, $r \in \mathbb{Q}$ un nombre racional i $t \in \mathbb{Z}^n$; les instancies positives són $dist(t, \mathcal{L}(B)) \leq r$ i les instàncies negatives $dist(t, \mathcal{L}(B)) > \gamma \cdot r$. L'objectiu és veure de quines instàncies prové l'entrada.

La majoria d'aplicacions criptogràfiques relacionades amb xarxes giren al voltant d'aquestes variants aproximades, ja que són les versions més bàsiques i amb menys estructura necessària, i de les que se n'ha demostrat més implicacions relacionades amb la seva complexitat així com reduccions entre els escenaris de pitjor-cas i cas-mitjana (*worst-case and average-case*).

4.3 Reducció de xarxes

L'objectiu de la reducció de xarxes és, donada una base d'una xarxa, obtenir-ne una amb vectors més curts i quasi ortogonals. La base més reduïda possible d'una xarxa de dimensió n seria la formada pels n primers vectors linealment independents corresponents als mínims succesius. Això s'anomena una base de Minkowski, però no sempre ha d'existir, ja que no necessàriament generen una base de la xarxa. Si imposem que els vectors siguin els vectors mínims que formen una base de la xarxa, obtenim l'anomenada base òptima. És a dir, una base òptima és aquella formada pels vectors linenalment independets més curts que formen una base de la base. Si la base de Minkowski existeix, és sempre una base òptima.

Obtenir una base mínima no és trivial. Malgrat que l'existència de diversos algoritmes per a reduïr bases de xarxes, no es coneix cap algoritme en temps polinomial per a obtenir bases de xarxes mínimes de dimensió $n > 2$.

4.3.1 Xarxes de dues dimensions

Per a xarxes de dimensió 2, existeix un algoritme atribuït a Gauss que aprofita la idea que hi ha darrere l'algoritme d'Euclides per a trobar el maxim comú divisor.

L'algoritme és com segueix:

Entrada: Una base d'una xarxa formada pels vectors x, y a $\mathbb{R}^2$, amb $\|x\| \leq \|y\|$.
Sortida: La base mínima de la xarxa generada pels vectors x, y.

Començament
  Fixem $b_1 \leftarrow x$ i $b_2 \leftarrow y$;
  Establim *acabat* com a $fals$;
  mentre *acabat sigui* $fals$ **fes**
    Fixa $m \leftarrow \left\lceil \frac{\langle b_2, b_1 \rangle}{\langle b_1, b_1 \rangle} \right\rfloor$;
    Fixa $b_2 \leftarrow b_2 - mb_1$;
    si $\|b_1\| \leq \|b_2\|$ **llavors**
      Estableix *acabat* com a *cert*;
    en cas contrari
      Intercanvia $b_1 \leftrightarrow b_2$;
    fi
  fi
  Retorna b_1 i b_2
Final

Algorithm 1: Algoritme de $Gaus(x, y)$

Si prenem l'exemple de la figura 1, podem aplicar l'algoritme sobre B_2, (a, b)

Exemple (Gaus(a,b))

  $b_1 = a = (5, 3)$ *i* $b_2 = b = (6, 4)$*;*
  acabat $= fals$*;*
  $m = \left\lceil \frac{\langle b_2, b_1 \rangle}{\langle b_1, b_1 \rangle} \right\rfloor = \left\lceil \frac{42}{34} \right\rfloor = 1$*;*
  $b_2 = (6, 4) - (5, 3) = (1, 1)$*;*
  $\|b_1\| = \sqrt{34} > \sqrt{2} = \|b_2\|$*;*
  $b_1 \leftrightarrow b_2$*,* $b_1 = (1, 1)$ *i* $b_2 = (5, 3)$*;*
  acabat segueix sent fals ;
  $m = \left\lceil \frac{\langle b_2, b_1 \rangle}{\langle b_1, b_1 \rangle} \right\rfloor = \left\lceil \frac{8}{2} \right\rfloor = 4$*;*
  $b_2 = (5, 3) - 4(1, 1) = (1, -1)$*;*
  $\|b_1\| = \sqrt{2} = \sqrt{2} = \|b_2\|$*;*
  acabat $=$ *cert;*
  retornem $b_1 = (1, 1)$ *i* $b_2 = (1, -1)$
fi

Com podem veure, $b_1 = u$ de la base B_1 però $v - u = b_2 \neq v$, això implica que (u, v), tot i ser una base reduïda de (a, b), no és la base mínima.

En aquesta línea, l'algoritme LLL permet trobar sempre una base reduïda, però no assegura que aquesta sigui una base mínima.

5 L'Algoritme LLL

5.1 Algoritme

En aquesta secció introduirem l'algoritme LLL, descrit per primer cop per A. K. Lenstra, H. W. Lenstra Jr. i L. Lovász a l'article [1]. Inicialment es va concebre com un algoritme per a factoritzar polinomis de coeficients enters tot i que el seu ús s'ha estès més enllà.

Definició 5.1 (Base δ-LLL reduïda)**.** Donat $\delta \in \mathbb{R}$, $\frac{1}{4} < \delta < 1$, una base $B = \{b_1, \ldots, b_n\}$ de $\mathbb{R}^n$ s'anomena una base δ-LLL, reduïda si per als seus elements se satisfà que

1. $\forall 1 \leq j < i \leq n,\ \mu_{i,j} \leq \frac{1}{2}$;
2. $\forall 1 \leq i \leq n,\ \delta \|\tilde{b}_i\|^2 \leq \|\mu_{i+1,i}\tilde{b}_i + \tilde{b}_{i+1}\|^2$.

Hem de veure tot seguit l'algoritme LLL i analitzar-ne els seus passos:

Entrada: La base B d'una xarxa, $b_1, \ldots, b_n \in \mathbb{Z}$ i una $\frac{1}{4} < \delta < 1$.
Sortida: Una base δ-LLL reduïda de $\mathcal{L}(B)$.

```
1  Començament
2  |  Computa la base Gram-Schmidt de L(B)
3  |  Pas 1
4  |  |  Des de i = 2 fins a n fes
5  |  |  |  Des de j = i − 1 fins a 1 fes
6  |  |  |  |  c_{i,j} ← ⌈μ_{i,j}⌋;
7  |  |  |  |  b_i ← b_i − c_{i,j} b_j ;
8  |  |  |  fi
9  |  |  fi
10 |  Pas 2
11 |  |  si ∃i t.q δ‖b̃_i‖² > ‖μ_{i+1,i} b̃_i + b̃_{i+1}‖² llavors
12 |  |  |  b_i ↔ b_{i+1};
13 |  |  |  torna al començament
14 Final
```

Algorithm 2: Algoritme LLL

Com podem veure, es tracta d'un algoritme relativament senzill, en el sentit que consta de només 14 línies, amb tres funcions recursives. El Pas 1 és coneix com a *fase de reducció*, ja que es reassigna un nou vector aparentment més petit a la base d'Entrada. El Pas 2 s'anomena sovint *Intercanvi*, ja que si es compleix la desigualtat de Algoritme, es fa un intercanvi entre els dos vectors. Anem a veure a continuació amb més detall cada pas de l'algoritme.

5.1.1 Fase de reducció

Malgrat el nom, no necessàriament el vector b_i es reduirà a cada iteració però servirà perquè la condició 1 de 5.1 es compleixi. Comencem per veure que aquesta reducció

no afectarà el càlcul de la base de Gram-Schmidt . Fixem-nos que les operacions seran del tipus $b_i \leftarrow b_i - Cb_j$, on $C \in \mathbb{Z}$. Per tant només sostrau un cert nombre de vegades la columna j a i, operació que no afectarà el càlcul de la base de Gram-Schmidt, ja que només és una combinació lineal entre les columnes. Si expressem B a la base $\tilde{B}'$ obtinguda en normalitzar els vectors de Gram-Schmidt, obtindrem una matriu triangular superior. Aquesta descomposició de B a partir de normalitzar els vectors de Gram-Schmidt s'anomena descomposició $B = QR$, on R és la matriu triangular superior. Si ens fixem en l'algoritme, la reducció itera en els elements d'aquesta matriu, amb $2 \leq i \leq n$ i $j < i$.

És important veure que les j van de grans a petites, ja que ens permetrà entendre el següent. Abans de començar, B expresat a la base ortonormal serà

$$\begin{bmatrix} \|\tilde{b}_1\| & \mu_{12}\|\tilde{b}_1\| & \mu_{13}\|\tilde{b}_1\| & \dots & & & \dots \\ 0 & \|\tilde{b}_2\| & \mu_{23}\|\tilde{b}_2\| & \dots & & & \dots \\ \vdots & \vdots & \|\tilde{b}_3\| & \dots & \mu_{i3}\|\tilde{b}_3\| & & \dots \\ 0 & & \ddots & \ddots & \vdots & & \\ 0 & & & \dots & \mu_{ii-1}\|\tilde{b}_{i-1}\| & & \dots \\ 0 & & & \dots & \|\tilde{b}_i\| & & \dots \\ 0 & & & \dots & 0 & \|\tilde{b}_{i+1}\| & \dots \\ \vdots & & & \vdots & \vdots & \vdots & \ddots \end{bmatrix},$$

que coincideix amb R a la descomposició QR.

A la iteració i-èsima, per a $j = 2$, el canvi a la matriu serà:

$$\begin{bmatrix} \|\tilde{b}_1\| & \leq \frac{1}{2}\|\tilde{b}_1\| & \leq \frac{1}{2}\|\tilde{b}_1\| & \dots & ? & ? & \dots \\ 0 & \|\tilde{b}_2\| & \leq \frac{1}{2}\|\tilde{b}_2\| & \dots & ? & ? & \dots \\ \vdots & \vdots & \|\tilde{b}_3\| & \dots & \leq \frac{1}{2}\|\tilde{b}_3\| & ? & \dots \\ 0 & & \ddots & \ddots & \vdots & & \\ 0 & & & \dots & \leq \frac{1}{2}\|\tilde{b}_{i-1}\| & ? & \dots \\ 0 & & & \dots & \|\tilde{b}_i\| & ? & \dots \\ 0 & & & \dots & 0 & \|\tilde{b}_{i+1}\| & \dots \\ \vdots & & & \vdots & \vdots & \vdots & \ddots \end{bmatrix}.$$

Anem a veure què passa amb l'element en el lloc (i, j) de la matriu anterior. Si operem com diu a l'algoritme, només afectarà els elements per sobre de j. En aquesta iteració en concret, sostraiem la columna 2 un nombre enter de vegades a la columna i, perquè el segon element de la columna i-èsima compleixi $\frac{1}{2}\left\|\tilde{b}_2\right\|$; que és la condició que veiem als elements per sobre la diagonal on tots són de la forma $\leq \frac{1}{2}\|\tilde{b}_k\|$, amb k el nombre de la fila. És a dir, en aquesta etapa ens assegurem que les projeccions de b_i sobre $\|\tilde{b}_j\| \, \forall j < i$ són com a molt $\leq \frac{1}{2}\|\tilde{b}_j\|$. Veurem que això és cert a l'anàlisi de la sortida.

5.1.2 Fase de d'intercanvi

En aquesta fase de l'algoritme ens trobarem que sí que canvia la base de Gram-Schmidt, i que el canvi és per a garantir que la segona condició de les bases δ-LLL Reduïdes es compleixi a la sortida del nostre algoritme. El *Intercanvi* ens diu que si no es compleix aquesta condició, es canviïn de lloc els elements b_i i b_{i+1}, que afectarà la base i el seu càlcul de Gram-Schmidt, en ser un canvi en l'ordre de les columnes i no una combinació lineal. Això suposarà que al final d'aquest s'hagi de tornar a calcular. A més, aquest pas *Intercanvi* només deixa d'actuar quan es compleix la condició esmentada, fet que suposa que, si l'algoritme acaba, aquesta condició es complirà.

Això ens porta a dues preguntes. La primera és; quan l'algoritme acaba, es compleixen les dues condicions perquè la base sigui δ-LLL reduïda? i la segona, de vital importància; acaba mai l'algoritme? Respondrem a aquestes preguntes a continuació.

5.1.3 Anàlisi de la sortida

Aquí volem veure si l'algoritme acaba, què n'obtenim. Per això, enunciem el següent resultat:

Lema 5.1. *Si l'algoritme LLL acaba, la seva sortida és una base δ-LLL reduïda de la xarxa generada per la base d'entrada $b_1, \ldots, b_n$.*

Demostració. Hem de veure que la sortida de l'algoritme compleix les dues condicions per a ser δ-LLL reduïda i a més que aquest genera la xarxa $\mathcal{L}(B)$. Podem garantir que serà una base de $\mathcal{L}(B)$, ja que els canvis sobre la base només consisteixen en combinacions lineals enteres invertibles entre columnes a la fase de reducció i en un intercanvi de columnes al pas *Intercanvi*, canvis que no afecten la xarxa generada per la base $b_1, \ldots, b_n$ en acabar l'algoritme. Hem vist que es complirà la segona condició si l'algoritme acaba a 5.1.2. Per tant només ens queda veure que complirà la primera condició, $\forall 1 \leq j < i \leq n, \quad \mu_{i,j} \leq \frac{1}{2}$.

Per veure-ho, ens hem de fixar en la fase de reducció. Com hem esmentat a 5.1.1, la base de Gram-Schmidt no canvia. Si a més prenem $i > j$ i considerem la iteració j-èsima del bucle intern a **5** i la iteració i-èssima a **4**, veiem que:

$$|\mu_{i,j}| = \left| \frac{\langle b_i - c_{i,j} \cdot b_j, \tilde{b}_j \rangle}{\langle \tilde{b}_j, \tilde{b}_j \rangle} \right| = \left| \frac{\langle b_i, \tilde{b}_j \rangle}{\langle \tilde{b}_j, \tilde{b}_j \rangle} - \lceil \frac{\langle b_i, \tilde{b}_j \rangle}{\langle \tilde{b}_j, \tilde{b}_j \rangle} \rfloor \cdot \frac{\langle b_j, \tilde{b}_j \rangle}{\langle \tilde{b}_j, \tilde{b}_j \rangle} \right| \leq \frac{1}{2},$$

on la primera igualtat és donada per la fase de reducció i la segona és certa gràcies al fet que $\langle b_j, \tilde{b}_j \rangle = \langle \tilde{b}_j, \tilde{b}_j \rangle$. □

5.2 Temps de computació

Quan parlem de temps de computació d'un algoritme, i en concret de si és polinòmic, no ens referim només a què les operacions que hi prenen part siguin considerades

polinòmiques, sinó també que els valors que apareguin siguin representables en un nombre polinòmic de bits.

Per exemple, si pensem en un algoritme que consisteixi a elevar quadràticament n vegades l'entrada, la funció d'exponenciar el valor requerirà n operacions aritmètiques, però el valor creix ràpidament en $2^{O(n)}$.

Per tant abans de començar és important considerar que la primera fita estarà establerta per la mida de l'entrada. Així doncs haurem de veure que el nombre d'operacions i valors són polinòmics en la mida de la base d'entrada. Aquest valor serà de la mida $\mathcal{M} = \max\{\text{n}, \log(\max_i \|b_i\|)\}$.

5.2.1 Iteracions

Hem de veure que l'algoritme acaba i per tant, que el seu nombre d'iteracions és finit. Per a fer-ho assignarem un nombre natural a la base, de forma que aquest disminueixi amb cada iteració. Sabem que el quadrat del determinant d'una matriu d'enters serà un nombre natural. El següent resultat ens ajudarà a assignar un nombre enter a la nostra xarxa.

Lema 5.2. *Si la base B d'una xarxa és entera, $det(\mathcal{L}(B))^2 \in \mathbb{Z}$.*

Gràcies a això associem el següent nombre a la xarxa:

$$\mathcal{D}_n = \prod_{k=1}^{n} det(\mathcal{L}(b_1, \ldots, b_k))^2 \in \mathbb{Z}.$$

Volem veure que $\mathcal{D} = \mathcal{D}_n$ es redueix almenys en un factor δ en cada iteració. Ja que el pas 3 de l'algoritme no depèn dels elements $\tilde{b}_i$, i com que $\mathcal{D}$ es pot expressar com a funció d'aquests elements (4.4), $\mathcal{D}$ no canviarà en aquest pas de l'algoritme. Tot seguit hem de veure com afectarà el pas 10 a $\mathcal{D}$. Si considerem el canvi entre $\tilde{b}_i$ i $\tilde{b}_{i+1}$, anomenant $\mathcal{D}'$ l'enter associat a la nova base, veiem que només canvia en cas que $k = i$, ja que per a $i < k$ només canvien vectors dins del determinant i aquest conserva el seu valor i per $k < i$ els dos queden fora i no hi ha cap canvi. Per tant $\mathcal{D}'$ només canvia si l'últim terme de $\mathcal{L}(b_1, \ldots, b_i)$ s'intercanvia pel següent. Anem a veure com ho fa.

$$\begin{aligned}
\frac{\mathcal{D}'}{\mathcal{D}} &= \frac{det\mathcal{L}(b_1, \ldots, b_{i-1}, b_{i+1})}{det\mathcal{L}(b_1, \ldots, b_i)} \\
&= \frac{(\prod_{j<i} \left\|\tilde{b}_j\right\|^2) \cdot \left\|\mu_{i+1,i}\tilde{b}_i + \tilde{b}_{i+1}\right\|^2}{\prod_{j\leq i} \left\|\tilde{b}_j\right\|^2} \\
&= \frac{\left\|\mu_{i+1,i}\tilde{b}_i + \tilde{b}_{i+1}\right\|^2}{\left\|\tilde{b}_i\right\|^2} < \delta.
\end{aligned}$$

Això implica que, per a $\mathcal{D}^{(k)}$, el valor final de $\mathcal{D}$ després de k iteracions és,

$$\mathcal{D}' \leq \delta\mathcal{D} \Rightarrow \mathcal{D}^{(k)} \leq \delta^n\mathcal{D} \Rightarrow k \leq \log_{\frac{1}{\delta}} \mathcal{D}.$$

Per tant, a cada iteració $\mathcal{D}'$ es redueix un factor δ. Tenint en compte que $\mathcal{D}$ és un enter positiu i $\mathcal{D} \geq 1$, que $\left(\frac{1}{\delta}\right)^n \leq \mathcal{D}$ i que el valor inicial de $\mathcal{D}$ està acotat per $(\max_i \|b_i\|)^{n(n+1)}$ i

$$k \leq \log_{\frac{1}{\delta}} \mathcal{D} = \frac{\log \mathcal{D}}{\log \frac{1}{\delta}} \leq \frac{1}{\log \frac{1}{\delta}} \cdot n(n+1) \log \left(\max_i \|b_i\|\right),$$

que, per a $\delta < 1$, serà d'ordre polinòmic.

Ja que hem acotat el nombre d'iteracions necessàries, ens queda veure que a cada iteració treballem també en temps polinòmic. Per a fer-ho hem de veure que tenim un nombre polinòmic d'operacions aritmètiques representables a cada una. Hem de demostrar doncs que en aquestes operacions, els valors que hi prenen part poden ser representats per un nombre polinòmic de bits.

5.2.2 Mida dels vectors

A continuació demostrarem que els nous valors que prenen els b_i i el càlcul dels $\tilde{b}_i$ a cada iteració prenen valors polinòmics.

Lema 5.3. *Els vectors $\tilde{b}_1, \ldots, \tilde{b}_n$ poden ser calculats en temps polinòmic fitat per $\mathcal{M}$. A més, per a $1 \leq i \leq n$ tenim que $\mathcal{D}\tilde{b}_i \in \mathbb{Z}^n$ i que $\|\tilde{b}_i\| \leq \mathcal{D}$.*

Demostració. Anem a veure que podem generar els $\tilde{b}_1, \ldots, \tilde{b}_n$ del procès d'ortogonalització de Gram-Schmidt. Segons el càlcul definit a 3.1 tenim que $\tilde{b}_i - b_i \in \langle b_1, \ldots, b_{i-1} \rangle_{\mathbb{R}}$ per tant podem escriure $\tilde{b}_i - b_i = \sum_{j=1}^{i-1} a_j b_j$, amb $a_1, \ldots, a_{i-1} \in \mathbb{R}$ tals que $\tilde{b}_i$ sigui ortogonal a cada $b_1, \ldots, b_{i-1}$. Si prenem $1 \leq k \leq i-1$, $\langle \tilde{b}_i, b_k \rangle = 0$ es pot escriure com:

$$\langle \tilde{b}_i, b_k \rangle = \langle b_i + \sum_{j=1}^{i-1} a_j b_j, b_k \rangle = \langle b_i, b_k \rangle + a_1 \langle b_1, b_k \rangle + \cdots + a_{i-1} \langle b_{i-1}, b_k \rangle = 0$$

que per a tot k ens genera el següent sistema de $i-1$ equacions i variables:

$$\begin{aligned} a_1\langle b_1, b_1 \rangle + a_2 \langle b_2, b_1 \rangle + \cdots + a_{i-1}\langle b_{i-1}, b_1 \rangle &= -\langle b_i, b_1 \rangle \\ a_1\langle b_1, b_2 \rangle + a_2 \langle b_2, b_2 \rangle + \cdots + a_{i-1}\langle b_{i-1}, b_2 \rangle &= -\langle b_i, b_2 \rangle \\ &\vdots \\ a_1\langle b_1, b_{i-1} \rangle + a_2 \langle b_2, b_{i-1} \rangle \cdots + a_{i-1}\langle b_{i-1}, b_{i-1} \rangle &= -\langle b_i, b_{i-1} \rangle \end{aligned}$$

Un sistema com aquest és resoluble en temps polinòmic i queda demostrada la primera part de la proposició. Per la segona part, si fem servir la regla de Cramer [2]:

$$a_j = \frac{det(\text{matriu d'enters})}{det \begin{pmatrix} \langle b_1, b_1 \rangle & \ldots & \langle b_{i-1}, b_1 \rangle \\ \vdots & \ddots & \vdots \\ \langle b_1, b_{i-1} \rangle & \ldots & \langle b_{i-1}, b_{i-1} \rangle \end{pmatrix}} = \frac{enter}{detB_{i-1}^T B_{i-1}} = \frac{enter}{det(\mathcal{L}(b_1, \ldots, b_{i-1}))^2};$$

per tant, $\tilde{b}_i = b_i + \sum_{j=1}^{i-1} a_j b_j$, on ara sabem que els a_j són nombres racionals amb denominador det $\mathcal{L}(b_1, \ldots, b_{i-1})^2$. Això implica que $\mathcal{D}_i \tilde{b}_i$ i en particular $\mathcal{D}\tilde{b}_i$ són vectors enters. Per la definició de $\mathcal{D}_i$,

$$\mathcal{D}_i = (\prod_{j=1}^{i-1} \|\tilde{b}_j\|)^2 \cdot \|\tilde{b}_i\|^2$$

i, per tant,

$$\sqrt{\|\tilde{b}_i\|} = \frac{\sqrt{\mathcal{D}_i}}{\prod_{j=1}^{i-1} \|\tilde{b}_j\|} \leq \sqrt{\mathcal{D}_i} \prod_{j=1}^{i-1} \mathcal{D}_j \leq \mathcal{D},$$

la primera desigualtat degut a què $\|\tilde{b}_j\| \geq \frac{1}{\mathcal{D}_j}$. □

Lema 5.4. *Els vectors b_i que apareixen a cada iteració poden ser representats amb poly($\mathcal{M}$) bits.*

Demostració. Hem de veure que després de la fase de reducció la llargada dels vectors b_i no ha crescut en excés. Per a cada $1 \leq i \leq n$,

$$\|b_i\|^2 = \|\tilde{b}_i\|^2 + \sum_{j=1}^{i-1} \mu_{i,j}^2 \|\tilde{b}_j\|^2 \leq \mathcal{D}^2 + \frac{n}{4}\mathcal{D}^2 \leq n\mathcal{D}^2.$$

La primera igualtat és pel fet que $\tilde{b_1}, \ldots, \tilde{b_n}$ són ortogonals. La primera desigualtat es basa en allò que s'ha demostrat en el lema anterior i gràcies al fet que $|\mu_{i,j}| \leq \frac{1}{2}$.

La fita aconseguida implica que cada coordenada de b_1 conté com a màxim un enter de mida $\sqrt{n}\mathcal{D}$. Per a un vector d'enters això implica que es pot representar amb $log(\sqrt{n}\mathcal{D})$ bits. Totes les b_i es mantenen com a vectors enters durant tota la fase, ja que ho són al principi i només canvien afegint-hi enters, podem assegurar per tant que en acabar la fase es poden representar amb poly($\mathcal{M}$) bits.

Per acabar, hem de demostrar que *durant* la fase de reducció, els vectors també es poden expressar en aquests termes. Si ens fixem en el bucle de la fase, si considerem el vector b_i,

$$|c_{i,j}| = \left| \lceil \frac{\langle b_i, \tilde{b_j} \rangle}{\langle \tilde{b_j}, \tilde{b_j} \rangle} \rfloor \right| \leq \frac{\|b_i\| \|\tilde{b}_j\|}{\|\tilde{b}_j\|^2} + 1 \leq \frac{\|b_i\|}{1/\mathcal{D}} + 1 \leq 2\mathcal{D}\|b_i\|;$$

la primera desigualtat s'obté aplicant la desigualtat de Cauchy-Schwartz i la segona a partir del lema anterior. Amb això podem veure que:

$$\|b_i - c_{i,j} b_j\| \leq \|b_i\| + |c_{i,j}| \|b_j\| \leq (1 + 2\mathcal{D}\|b_j\|)\|b_i\| \leq (1 + 2\mathcal{D}\sqrt{n}\mathcal{D})\|b_i\| \leq (4n\mathcal{D})^2 \|b_i\|.$$

La primera desigualtat és la triangular, la segona ve donada per la fita trobada per a $|c_{i,j}|$ i la següent per la fita de $\|b_j\|$ després de la fase de reducció. Efectivament, si estem a la iteració de b_i, tots els $j < i$, ja han estat reduïts, i no tornaran a ser modificats, per això podem fer servir la fita. Per tant, hem vist que després de com a màxim n iteracions del bucle, la norma de b_i ha augmentat com a molt $(4n\mathcal{D})^{2n}$, que és representable amb poly($\mathcal{M}$) bits. □

Per tant, hem vist que els vectors són representables en un nombre polinòmic en $\mathcal{M}$ de bits i a més que el nombre d'iteracions a l'algoritme és també polinòmic. Podem concloure que el temps de computació de l'algoritme és polinòmic en la mida de l'entrada.

5.3 Implicacions

Proposició 5.5. *Prenem $b_1, \ldots, b_n$ com la sortida de l'algoritme LLL, és a dir, una base δ-LLL reduïda de la xarxa $\mathcal{L}$ a $\mathbb{R}^n$. Llavors $\|b_1\| \leq (\frac{2}{\sqrt{4\delta-1}})^{n-1}\lambda_1(\mathcal{L})$.*

Demostració. Ja que per a tota base $b_1, \ldots, b_n$, tenim que $\lambda_1(\mathcal{L}) \geq \min_i \|\tilde{b}_i\|$, obtenim que

$$\|\tilde{b}_n\|^2 \geq (\delta - \frac{1}{4}\|\tilde{b}_{n-1}\|^2 \geq \cdots \geq (\delta - \frac{1}{4})^{n-1}\|\tilde{b}_1\|^2 = (\delta - \frac{1}{4})^{n-1}\|\tilde{b}_1\|^2,$$

amb l'última igualtat ja que, per definició, $\tilde{b_1} = b_1$. Llavors, per a qualsevol i,

$$\|\tilde{b}_1\| \leq \left(\delta - \frac{1}{4}\right)^{-(i-1)/2} \|\tilde{b}_i\| \leq \left(\delta - \frac{1}{4}\right)^{-(n-1)/2} \|\tilde{b}_i\|.$$

Per tant,

$$\|b_1\| \leq \left(\delta - \frac{1}{4}\right)^{-(n-1)/2} \min_i \|\tilde{b}_i\| \leq \left(\delta - \frac{1}{4}\right)^{-(n-1)/2} \cdot \lambda_1(\mathcal{L})$$

□

Proposició 5.6. *Si $b_1, \ldots, b_n$ és una base δ-reduïda de la xarxa $\mathcal{L}$ a $\mathbb{R}^n$, i $b'_1, \ldots, b'_n \in \mathcal{L}$ són n vectors linealment independents de la xarxa, llavors per a $1 \leq j \leq n$ tenim que*

$$\|b_j\| \leq \beta^{(n-1)/2} max\{\|b'_1\|, \ldots, \|b'_n\|\},$$

on $\beta := \frac{4}{4\delta-1} > \frac{4}{3}$ es coneix com a paràmetre auxiliar.

Demostració. Comencem escrivint b'_j com a combinació lineal dels $b_1, \ldots, b_n$,

$$b'_j = \sum_{i=1}^{n} \alpha_{i,j} b_i \quad \alpha_{i,j} \in \mathbb{Z}.$$

Prenem i' com el valor més gran de i tal que $\alpha_{i,j} \neq 0$. Podem expressar i' com $i'(j)$, ja que depèn de j. Segons la definició per l'ortogonalització de Gram Schmidt podem escriure:

$$b'_j = \sum_{i=1}^{i'(j)} \alpha_{i,j} b_i = \sum_{i=1}^{i'(j)} \alpha_{i,j} \sum_{k=1}^{i} \mu_{i,k} \tilde{b}_k = \sum_{i=1}^{i'(j)} \sum_{k=1}^{i} \alpha_{i,j}\mu_{i,k} \tilde{b}_k.$$

Si considerem el terme $\tilde{b}_{i'(j)}$ podem observar que $\alpha_{i,i'(j)} \neq 0$ i $\mu_{i'(j),i'(j)} = 1$; obtenim que

$$\left\|b'_j\right\|^2 \geq \left\|\tilde{b}_{i'(j)}\right\|^2; \tag{5.1}$$

ja que $\{b'_1, \ldots, b'_n\}$ és un conjunt desordenat, podem assumir que,

$$i'(1) \leq i'(2) \leq \cdots \leq i'(n).$$

Podem assumir també que $j \leq i'(j)$, ja que si tinguéssim $i'(j) < j$, podríem expressar $b'_1, b'_2, \ldots, b'_j$ com a combinacions lineals de $b_1, b_2, \ldots, b_{i'(j)}$, i com que $i'(j) < j$, aquests últims contradiuen la independència vectorial assumida l'inici. Per tant, $j \leq i'(j)$ i utilitzant la proposició anterior i l'equació 5.1,

$$\|b_j\|^2 \leq \beta^{i'(j)-1}\left\|\tilde{b}_{i'(j)}\right\|^2 \leq \beta^{n-1}\left\|\tilde{b}_{i'(j)}\right\|^2 \leq \beta^{n-1}\left\|b'_j\right\|^2 \leq \beta^{n-1} max\{\|b'_1\|^2, \ldots, \|b'_n\|^2\},$$

i, per tant, com enunciava la proposició,

$$\|b_j\| \leq \beta^{(n-1)/2} max\{\|b'_1\|, \ldots, \|b'_n\|\}.$$

□

Corol·lari 5.7. *El vectors d'una base δ-LLL reduïda $b_1, \ldots, b_n$ de la xarxa $\mathcal{L}$ satisfan:*

$$2^{1-i} \leq \frac{\|b_i\|^2}{\lambda_i^2(\mathcal{L})} \leq 2^{n-1}.$$

6 Anàlisi computacional

Per a realitzar una anàlisi computacionals de l'algoritme LLL s'ha utilitzat la distribució de Sage junt a Python. Els tests s'han realitzat amb un *MacOS Sierra* amb processador *2,3 GHz Intel Core i5* i memòria *4 GB 1333 MHz DDR3*.

6.1 Temps

En aquesta prova, s'han generat 100 matrius aleatòries d'enters petits, entre -9999 i 9999, per a cada dimensió i, amb $i \in (2, 10, 18, \ldots, 194, 202)$. S'ha aplicat l'algoritme LLL a les 100 matrius, obtenint els temps en mitjana, màxim i mínim. Un cop obtinguts els resultats, s'ha aplicat el logaritme sobre els temps per obtenir els gràfics següents:

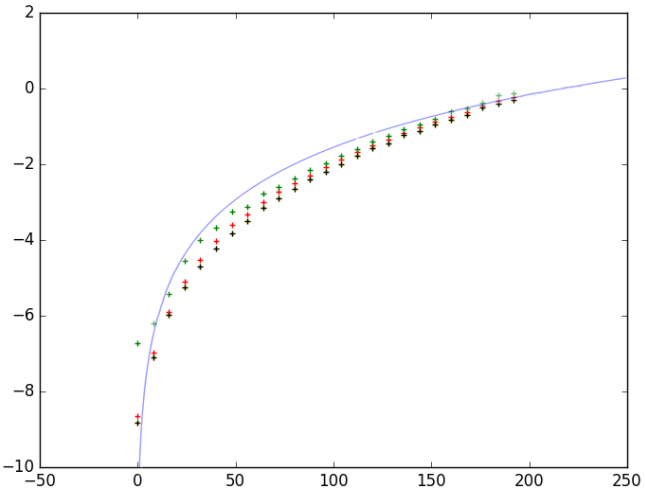

Figura 5: Temps de computació màxim (verd), en mitjana (vermell), mínim (negre) i una regressió logarítmica (blau)

Com podem veure, el temps augmenta considerablement amb el creixement de i, però no té un comportament lineal, i per tant resulta una comprobació *empírica* del temps polinòmic de l'algoritme LLL. Notar també que el pitjor dels casos no és gaire pitjor que el temps promig, degut al caràcter determinista de l'algoritme. La regressió s'ha fet utilitzant una funció $y = A + Bln(x)$, i s'ha obtingut $A = -10.8$ i $B = 2$, fet que indica que és quadràtic.

6.2 Mida

Aquí hem analitzat la mida dels vectors de sortida de l'algoritme, comparant-los amb els d'entrada. Per a una base A formada per enters al voltant de 24 dígits de dimensió 100, hem obtingut el que es mostra a la figura 6

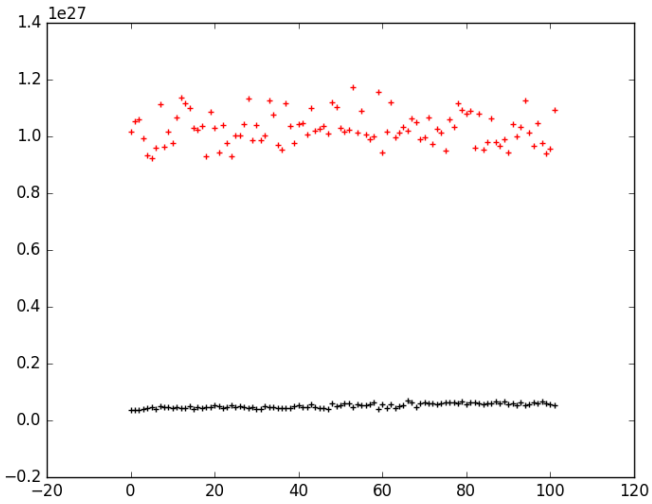

Figura 6: Normes dels vectors de la base original (vermell) i de la reduïda (negre)

Com podem veure, els vectors s'han reduït força. És de destacar el fet que el primer vector de la reducció, tot i ser més curt, no necessàriament és el més curt, ja que sempre ens movem al voltant de cotes, i l'algoritme acostuma a comportar-se millor de l'esperat a la pràctica. En el nostre cas, per a la base de sortida B, tenim que $\|b_1\| = 3.747 \times 10^{25}$, i en canvi $min_i(\|b_i\|) = 3.697 \times 10^{25}$. Per a fer-nos una idea de la reducció, la mitjana de les normes de la base d'entrada A és 1.027×10^{27} i la de sortida 5.215×10^{25}, amb $max_i(\|a_i\|) = 1.175 \times 10^{27}$ i $max_i(\|b_i\|) = 6.943 \times 10^{25}$. Per tant les normes s'han reduït de l'ordre d'un factor 20.

A la següent gràfica hem repetit el procès anterior per bases de dimensió 30, i representem la mitjana de les normes, així com els valors mínim i màxim de cada iteració.

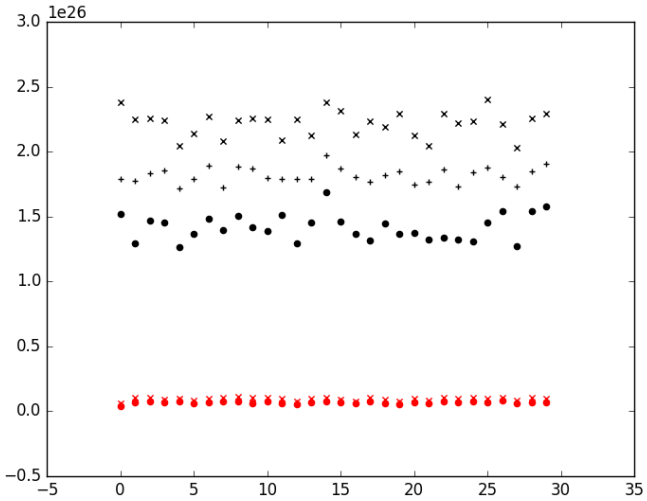

Figura 7: Mitjanes de les normes dels vectors de les bases original (vermell) i de les reduïdes (negre). Les creus + representen les mitjanes, les x els valors màxims i les rodones els mínims.

Com podem veure, per molt que els valors d'entrada puguin variar més, un cop reduïdes, les bases prenen valors similars.

7 Aplicació a altres algoritmes de reducció de bases de xarxes

L'algoritme LLL també s'utilitza com a pas intermedi en altres algoritmes de reducció de bases de xarxes. Com hem vist, una base òptima, que pot ser Minkowski reduida, consisteix en obtenir els vectors més curts que formen una base de la xarxa. Existeixen algoritmes per obtenir-la, però no són gaire millors que una cerca exhaustiva d'aquests, sobretot en augmentar la dimensió de la xarxa i, per tant, de temps exponencial, i no són pràctics. Els esmentem sense entrar en detalls

Definició 7.1 (Base reduïda)**.** Una base d'una xarxa $\mathcal{L}$ s'anomena reduïda si per la seva descompició QR, els vectors r_i de R compleixen:

$$|r_{i,j}| \leq \frac{1}{2}|r_{i,i}|.$$

Corol·lari 7.1. *Una base reduïda d'una xarxa compleix la primera condició d'una base δ-LLL reduïda definida a 1.*

És interessant en aquest context redefinir la condició de base δ-LLL reduïda.

Definició 7.2 (Base δ-LLL reduïda)**.** Una base d'una xarxa $\mathcal{L}$ s'anomena δ-LLL reduïda si la seva matriu R de la descomposició QR és reduïda i a més compleix que:

$$r_{i,i}^2 + r_{i-1,i}^2 \geq \delta r_{i-1,i-1}^2,$$

que equival a la condició 2 de la definició de Base δ-LLL reduïda a la pàgina11.

Definició 7.3 (Base HKZ reduïda)**.** Una base d'una xarxa $\mathcal{L}$ s'anomena HKZ reduïda si la seva matriu R de la descomposició QR és reduïda i per a cada submatriu de dimensió $(n-i+1)\times(n-i+1)$ de la base, per a $1 \leq i \leq n$, la seva primera columna és el vector més curt de la xarxa generada per la submatriu. També es pot definir com una base on el primer vector és el més curt de la xarxa i totes les sub-bases que genera la base HKZ reduïda, són també HKZ reduïdes.

Una base HKZ és força propera al concepte de base mínima, i obtenir-ne una serà gairebé igual de difícil. Existeixen dos algoritmes importants per obtenir bases HKZ reduïdes. Un és de Kannan [3], amb una millora per part de Helfrich i Schnorr, amb una complexitat de $n^{\frac{n}{2e}(1+o(1))}$ com a cota superior sobre n dimensió de la base. L'altre algoritme és de Ajtai, Kumar i Sivakumar [4], amb una complexitat probable de $2^{5,9\cdot n}$ però amb alguns desavantatges respecte el primer, com el fet de ser probabilístic.

Definició 7.4 (Base k-BKZ reduïda)**.** Una base d'una xarxa $\mathcal{L}$ s'anomena BKZ reduïda si la seva matriu R de la descomposició QR és reduïda i les submatrius de dimensió $k\times k$ són HKZ reduïdes.

Una base 2-BKZ reduïda equival pràcticament a una de δ-LLL reduïda i una n-BKZ és exactament igual que una HKZ reduïda. Es pot entendre per tant la

reducció k-BKZ com una generalització entre les altres dues. Per aconseguir una base k-BKZ reduïda existeix un algoritme de Schnorr. Tot i la seva practicitat, no està garantit que funcioni de forma polinòmica, amb els temps compromesos pel tamany dels blocs, quan més grans siguin, major temps de computació. Sovint s'utilitza l'algorime, forçant-ne la parada quan es creu que haurà assolit una base suficientment petita segons la utilitat pràctica.

L'algoritme de Schnorr per a la reducció BKZ depèn directament de l'algoritme LLL. En fa crides recursives alternades amb un algoritme ENUM que busca i insereix un vector b_{new}, que és el més curt possible generat per $b_i, \ldots, b_{i+k-1}$. L'algoritme LLL es crida tot seguit per a eliminar la dependència lineal. L'algoritme s'atura quan el procès esmentat no genera ja cap canvi i no existeix cap fita relativa a quan succeeix. A l'article [5] s'explica amb detall al apartat 6. Essencialment, és una combinació de l'algoritme LLL sobre dimensió n amb un algoritme de resolució del SVP 4.1 sobre dimensió k, de temps exponencial.

A la pràctica, tant l'algoritme LLL com el BKZ, acostumen a comportar-se millor que les seves prediccions teòriques, malgrat que no es coneix una raó d'aquest fet.

8 Factorització de polinomis de coeficients enters

Una de les aplicacions més conegudes de l'algoritme LLL és per a factorització de polinomis de coeficients enters. De fet, va ser la primera que es va donar. Mostrarem aquí com l'algoritme s'utilitza en aquest context, però per entendre-ho, haurem de parlar abans d'alguns algoritmes i teoremes necessaris per descriure el procediment.

Sigui $\mathbb{F}$ un cos finit, considerem l'anell de polinomis $\mathbb{F}[x]$, tal que els elements de l'anell prenen la següent forma:

$$a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 \quad n \geq 0,$$

amb $a_n, \ldots, a_0 \in \mathbb{F}$ i $a_n \neq 0$, on n és el grau del polinomi, $n = deg(f)$ i $a_n = l(f) \in \mathbb{F}$ el primer coeficient. Prenem $f, g \in \mathbb{F}[x]$. Si un polinomi no és lliure de quadrats, és a dir, $mcd(f, f') \neq 1$, substituim el polinomi pel seu $rad(f)$.

Així com l'algoritme d'Euclides ens dóna el màxim comú divisor entre f i g, anomenarem $XEuclid(f, g)$ a l'algoritme d'Euclides estès, que ens donarà com a sortida el $mcd(f, g) = h$ i dos polinomis $s, t \in \mathbb{F}[x]$ pels quals $sf + tg = h$. L'algoritme utilitza un procediment anàleg a Euclides, però assignant un valor a $s_{i+1} \leftarrow s_{i-1} - q_i s_i$, i $t_i \leftarrow s_{i-1} - q_i s_i$ a cada iteració utilitzant $r_{i?1} = q_i r_i + r_{i+1}$, tal que $s_i f + t_i g = r_i$ fins que $r_i = 0$ i ens queda $s_{i-1} = s, t_{i-1} = t$.

Considerem ara l'anell de polinomis $\mathbb{F}_q[x]$ amb $q = p^n$ amb p primer i n enter positiu. En aquest context tindrem el següent teorema:

Teorema 8.1. *Per a $i \geq 1$ i $q = p^n$, a $\mathbb{F}_q[x]$ se satisà la igualtat*

$$x^{q^i} - x = \prod_{d|i} \prod_{\substack{deg(g)=d \\ f\, monic\ irreductible}} f$$

Observació 8.1. Notem que el cas particular $i = 1$ és

$$x^q - x = \prod_{a \in \mathbb{F}_q} (x - a).$$

Amb això podem veure a continuació un mètode per a descompondre un polinomi lliure de quadrats en el producte dels seus elements de diferents graus.

8.1 Descomposició d'un polinomi en factors isorreductibles

Proposem la definició següent.

Definició 8.1. Un polinomi és isoreductible de grau d si és producte de factors irreductibles del mateix grau.

Un polinomi és isoreductible de grau d quan els seus factors irreductibles tenen el mateix grau, d. Si tenim un polinomi f mònic a $\mathbb{F}_q[x]$, la seva descomposició en

els factors h_d isoreductibles consisteix en la descomposició

$$f = \prod_{d=1}^{\delta} h_d,$$

on δ és el grau més gran dels factors irreductibles de f i cada h_d és el producte

$$h_d = \prod_{j=1}^{l_d} h_{dj}$$

amb h_{dj} els elements irreductibles de f amb grau d i l_d el nombre de factors irreductibles amb aquest grau. Per tant, definim la seqüència com:

Definició 8.2 (Descomposició en factors isoreductibles d'un polinomi)**.**

$$ddg(f) = [h_1, h_2, \ldots, h_\delta]$$

Per aconseguir obtenir aquesta descomposició només hem d'aplicar el resultat 8.1. Com que aquest ens diu la descomposició del polinomi $x^{q^i} - x$, si anem iterant el càlcul, $h_i = mcd(f_{i-1}, x^{q^i} - x)$ per $1 \leq i \leq \delta$, amb $f_i = \frac{f_{i-1}}{h_i}$, amb $f_0 = f$, acabem obtenint tots els h_i corresponents a $ddg(f)$. Ara que hem vist com és de fàcil obtenir una descomposició en factors isorreductibles, anem a veure com obtindrem una descomposició dels polinomis isorreductibles, que serà força més complicat.

8.2 Descomposició dels polinomis isorreductibles

Definició 8.3 (Descomposició d'un polinomi issoreductible)**.**

$$dmg(h_d) = [h_{d1}, \ldots, h_{dl_d}]$$

Per aconseguir aquesta descomposició s'utilitza un mètode probabilístic, basat en l'algoritme de Cantor i Zassenhaus, que gràcies al teorema xinès del residu, sabem que $\mathbb{F}_q[x]/(h_d) \cong \mathbb{F}_q[x]/(h_{d1}) \times \cdots \times \mathbb{F}_q[x]/(h_{dl_d})$, que ens diu que trobar factors irreductibles, és equivalent a trobar les arrels de 1 i -1 a $\prod \mathbb{F}_q[x]/(h_d)$, que és el que intentarà l'algoritme:

Entrada: Un polinomi mònic lliure de quadrats $h \in \mathbb{F}_q[x]$ de grau dl, producte de l factors de grau d

Sortida: Si $l \geq 2$, retorna, amb una probabilitat $\geq \frac{1}{2}$, un factor g de h; en cas que no trobi g retorna 0

Començament
  si $gr(h) = 1$ **llavors**
    $g \leftarrow 0$;
  en cas contrari
    Genera $g_1 \in \mathbb{F}_q[x]$ pseudoaleatori amb $gr(g_1) < prd(h)$;
    $g_2 \leftarrow mcd(g_1, h)$;
    si $g_2 \neq 1$;
     llavors
      $g \leftarrow g_2$;
    en cas contrari
      Estableix $e \leftarrow (q^d - 1)/2$;
      Estableix $g_3 \leftarrow \bar{g_1^e}$ on la barra denota el residu sobre mod h;
      Estableix $g_4 \leftarrow mcd(g_3 - 1, h)$;
      si $0 < gr(g_4) < gr(h)$ **llavors**
        estableix $g \leftarrow g_4$
      en cas contrari
        $g \leftarrow 0$
      fi
    fi
  fi
  Retorna g
Final

Algorithm 3: Algoritme de $IntentParticio(h)$

Aquest algoritme per tant intenta partir el polinomi en dos factors. Si l'utilitzem recursivament, podem obtenir un factor g de h amb una probabilitat superior a $\geq 1 - 2^s$ amb s un paràmetre de parada. A l'algoritme que crida $IntentParticio(h)$ fins a s l'anomenem $Particio(h, s)$. Utilitzant això podem generar un algoritme que actuï recursivament sobre els factors que obtenim de $Particio(h, s)$ per a obtenir

tots els factors de la descomposició del mateix grau del polinomi:

Entrada: Un polinomi mònic lliure de quadrats $h \in \mathbb{F}_q[x]$ de grau dl, producte de l factors de grau d i un paràmetre de parada $s \geq 1$
Sortida: La descomposició del mateix grau del polinomi en una llista de factors

Començament
  Genera $g_1 \leftarrow Particio(h, s)$;
  si $g_1 = 0$;
  llavors
    Afegeix h a la llista de factors;
  en cas contrari
    Crida de forma recursiva $DMG(g_1, s)$;
    Crida de forma recursiva $DMG(h/g_1, s)$;
  fi
  Retorna la llista de factors.
Final

Algorithm 4: Algoritme $DMG(h, s)$

8.3 Aixecament de Hensel

Amb els algoritmes $ddg(f)$ i $DMG(h, s)$ podem obtenir la descomposició completa dels factors de f, però no hem d'oblidar que estem a $\mathbb{F}_q[x]$. El nostre objectiu és la factorització de polinomis de coeficients enters, i per tornar de $\mathbb{F}_q[x]$ a $\mathbb{Z}[x]$ farem servir el que s'anomena aixecament de Hensel.

El lema de Hensel ens garanteix que si tenim un polinomi amb una arrel simple mòdul p amb p primer, aquesta arrel correspon a una arrel única del mateix polinomi mòdul qualsevol potència de p, i que podem obtenir-les executant el que s'anomena aixecament de la solució mòdul succesives potències de p. Podem utilitzar això en el nostre cas per aixecar els factors que trobem després d'aplicar l'algoritme que combini $ddg(f)$ i $DMG(h, s)$, que anomenarem $Factoritzacio(f, s)$. Enunciem formalment la versió del lema de Hensel que farem servir.

Lema 8.2 (Lema de Hensel)**.** *Prenem p un nombre primer. Prenem $f, g_1, h_1, s_1, t_1 \in \mathbb{Z}[x]$ tal que h_1 és mònic i*

$$f \equiv g_1 h_1 \pmod{p}, \qquad gr(f) = gr(g_1) + gr(h_1),$$
$$s_1 g_1 + t_1 h_1 \equiv 1 \pmod{p}, \qquad gr(s_1) < gr(h_1), \quad gr(t_1) < gr(g_1).$$

Llavors per a qualsevol $n \geq 1$, existeixen $g_n, h_n, s_n, t_n \in \mathbb{Z}[x]$ tal que h_n és mònic i

$$f \equiv g_n h_n \pmod{p^{2^n}}, \qquad g_n \equiv g_{n-1} \pmod{p^{2^{n-1}}}, \quad gr(g_n) = gr(g_1),$$
$$h_n \equiv h_{n-1} \pmod{p^{2^{n-1}}}, \quad gr(h_n) = gr(h_1),$$
$$s_n g_n + t_n h_n \equiv 1 \pmod{p^{2^n}}, \qquad s_n \equiv s_{n-1} \pmod{p^{2^{n-1}}}, \quad gr(s_n) < gr(h_n),$$
$$s_n \equiv s_{n-1} \pmod{p^{2^{n-1}}}, \quad gr(t_n) < gr(g_n).$$

Amb això podem escriure l'algoritme d'aixecament de Hensel, on m serà una potència de p.

Entrada: Un mòdul m$\in \mathbb{Z} \geq 2$, els polinomis $f, h_1, g_1, s_1, t_1 \in \mathbb{Z}[x]$ en les condicions del Lema 8.2

Sortida: $g_2, h_2, s_2, t_2 \in \mathbb{Z}[x]$ tal com queden definits al Lema 8.2

1 **Començament**

2 Pren $e \leftarrow f - g_1 h_1 \pmod{m^2}$;

3 Computa $q, r \in Z[x]$ tals que

$$s_1 e \equiv q h_1 + r \pmod{m^2}, \quad gr(r) < gr(h_1)$$

4 Estableix $g_2 \leftarrow g_1 + t_1 e + q g_1 \pmod{m^2}$;

5 Estableix $h_2 \leftarrow h_1 + r \pmod{m^2}$;

6 Estableix $e^* \leftarrow s_1 g_2 + t_1 h_2 - 1 \pmod{m^2}$;

7 Computa $q^*, r^* \in \mathbb{Z}[x]$ tals que

$$s_1 e^* \equiv q^* h_2 + r^* \pmod{m^2}, \quad gr(r^*) < gr(h_2)$$

8 Estableix $s_2 \leftarrow s_2 + r^* \pmod{m^2}$;

9 Estableix $t_2 \leftarrow t_1 + t_1 e^* + q_2^* \pmod{m^2}$;

10 Retorna g_2, h_2, s_2, t_2.

11 **Final**

Algorithm 5: Algoritme $Hensel(m, f, h_1, g_1, s_1, t_1)$

Amb els algoritmes introduïts som capaços de generar-ne un per factoritzar polinomis amb coeficients enters i veure com la reducció de xarxes a través de l'algoritme LLL ens pot ajudar.

8.4 Algoritme de factorització de polinomis de coeficients enters

Entrada: Un polinomi lliure de quadrats $f \in \mathbb{Z}[x]$
Sortida: Els factors irreductibles de f, $f_1, \ldots, f_r \in \mathbb{Z}[x]$

1 **Començament**
2 Fixa $n \leftarrow gr(f)$;
3 **si** $n = 1$ **llavors**
4 resultat $\leftarrow [f]$
5 **en cas contrari**
6 Fixa $C \leftarrow (n+1)^{2n}|f|_\infty^{2n-1}$, fixa $r \leftarrow \lceil 2\log C \rceil$;
7 Fixa $B \leftarrow (n+1)^{1/2}2^n|f|_\infty|l(f)|$.;
8 Troba un primer $p < 2rlnr$ tal que $p \nmid l(f)$ i $p \nmid disc(f)$.;
9 Fixa $k \leftarrow \lceil \log_p(2B+1) \rceil$.;
10 Crida Factorització per trobar factors irreductibles $h_1, \ldots, h_r \in \mathbb{Z}[x]$ tals que $f \pmod p = l(f)h_1 \ldots h_r \pmod p$.;
11 Crida Hensel sobre els polinomis h_i de forma repetida per tal de trobar els polinomis $g_1, \ldots, g_r \in \mathbb{Z}[x]$ per als quals $f \equiv l(f)g_1 \ldots g_r \pmod{p^k}$ i $g_i \equiv h_i \pmod p$ per a tot i;
12 Fixa $T \leftarrow \{1, 2, \ldots, r\}$, $F \leftarrow f$, $s \leftarrow 1$, $acabat = 0$;
13 **mentre** $2s \leq |T|$ **fes**
14 Fixem $llista \leftarrow \{s - \text{elements subset de } T\}$;
15 Fixa $j \leftarrow 0$;
16 **mentre** $j < |llista|$ *i* $acabat = 0$ **fes**
17 Fixa $j \leftarrow j+1$ i $S \leftarrow llista[j]$ Troba $G, H \in \mathbb{Z}[x]$ tals que $|G|_\infty, |H|_\infty \leq p^k/2$ i

$$G \equiv l(F)\prod_{i \in S} g_i \pmod p^k, \quad H \equiv l(F)\prod_{i \in T \setminus S} g_i \pmod p^k$$

si $|G|_1|H|_1 \leq B$ **llavors**
18 Afegeix G al resultat Fixa $T \leftarrow T \setminus S$, fixa $F \leftarrow prim(H)$, $acabat = 1$.
19 Fixa $s \leftarrow s+1$
20 **fi**
21 **fi**
22 **fi**
23 **Final**

Algorithm 6: Algoritme $\mathbb{Z}Factor(f)$

Els valors B, C ens serveixen com a fites. C ens dóna un màxim del discriminant. B s'anomena fita de Mignotte:

Definició 8.4 (Fita de Mignotte)**.** Si $f, g, h \in \mathbb{Z}[x]$ compleixen $f = gh$, llavors

$$|g|_1|h|_1 \leq (n+1)^{1/2}2^n|f|_\infty|l(f)| = B, \quad n = gr(f).$$

Els passos $13 - 21$ consisteixen en, un cop obtinguda la factorització mòdul

p^k, trobar els factors a $\mathbb{Z}[x]$. Com que alguns factors irreductibles de $\mathbb{Z}[x]$ poden reduir en més factors a $\mathbb{F}_q[x]$, haurem d'anar buscant exhaustivament entre tots els subconjunts de factors trobats per reconstruir els factors a $\mathbb{Z}[x]$ a través de S. Gràcies al fet que hi ha $2^r - 1$ subconjunts no buits de 1,2,..., r, en el pitjor dels casos això pot portar a un nombre exponencial d'iteracions del bucle a 13, en concret a una exponencial de $n = gr(f)$.

En aquest context, l'algoritme LLL ens permet realitzar aquesta cerca en temps polinòmic.

8.4.1 LLL per a factorització

Per començar, hem de canviar els valors per B i k a 6 i 7:

$$B \leftarrow (n+1)^{1/2} 2^n |f|_\infty, \quad k \leftarrow \lceil \log_p(2^{n^2/2} B^{2n}) \rceil,$$

que implica

$$p^k \leq 2^{n^2/2} B^{2n} = 2^{n^2/2}((n+1)^{1/2} 2^n |f|_\infty)^{2n} = 2^{n^2/2}(n+1)^n 2^{2n^2} |f|_\infty^{2n};$$

aquests canvis en serviran més endavant per garantir una sèrie de propietats.

El nostre objectiu serà substituir la cerca dels factors de $\mathbb{Z}[x]$ dels passos $13-21$ per un mètode de temps polinòmic. Per això farem servir els següents resultats.

Lema 8.3. *Tenim $f, g \in \mathbb{Z}[x]$ amb $gr(f) + gr(g) \geq 1$ llavors existeixen $s, t \in \mathbb{Z}[x]$ tals que $sf + tg = res(f, g)$ amb $gr(s) < gr(g)$ i $gr(t) < gr(f)$. [6] [pg.153]*

Lema 8.4. *Tenim $f, g \in \mathbb{Z}[x]$ amb $n = gr(f)$ i $m = gr(g)$; llavors,*

$$|Res(f, g)| \geq |f|_2^m |g|_2^n \geq (n+1)^{m/2}(m+1)^{n/2} |f|_\infty^m |g|_\infty^n.$$

[6] [pg.156]

Aquí, $Res(f, g)$ és el resultant de f i g, que es defineix com el determinant de la matriu de Sylvester, $S(f, g)$. El resultant de f i el seu derivat, $Res(f, f')$ és el discriminant.

Veiem ara l'últim resultat necessari, i sens dubte el resultat central, per a la cerca amb LLL.

Lema 8.5. *Tenim $f, g \in \mathbb{Z}[x]$ amb $n = gr(f) > 0$ i $m = gr(g) > 0$. Suposem que $\exists u \in \mathbb{Z}[x]$ mònic i no constant, i que $f \equiv uv_1 \pmod m$ i que $g \equiv uv_2 \pmod m$ per a $v_1, v_2 \in Z[x]$ i $m > |f|_2^k |g|_2^n$. Llavors $mcd(f, g) \in \mathbb{Z}[x]$ no és constant.*

Demostració. Ho veurem per contradicció, provant primer que $mcd(f, d) \in \mathbb{Q}[x]$ no és constant. Suposem doncs que $mcd(f, d) = 1 \in \mathbb{Q}[x]$. Pel lema 8.4 existeixen $s, t \in \mathbb{Z}[x]$ tal que $sf + tg = Res(f, g)$ i per tant $sf + tg \equiv Res(f, g) \pmod m$. Com que $f \equiv uv_1 \pmod m$ i $g \equiv uv_2 \pmod m$, obtenim:

$$Res(f, g) = u(sv_1 + tv_2) \pmod m$$

i per tant u divideix a $Res(f,g) \pmod m$. Però u és mònic i no constant i $Res(f,g) \in \mathbb{Z}$ tal que $Res(f,g) \in \mathbb{Z}$, per tant $mcd(f,g) \equiv 0 \pmod m$. El lema 8.4 implica que:

$$m > |f|_2^k |g|_2^n \geq |Res(f,g)|,$$

i per tant $Res(f,g) = 0$. Però això implica que $mcd(f,g) \neq 1$, que contradiu la hipòtesi inicial. Per tant $mcd(f,g) \in \mathbb{Q}[x]$ no és constant i per tant tampoc ho és $mcd(f,g) \in \mathbb{Z}[x]$. □

Anem a fer ús d'aquests lemes per a definir una alternativa de temps polinòmic. Si hem seguit tots els passos de l'algoritme fins al pas 12, tindrem una sèrie de factors de $f \pmod p^k$, per tant, en el context del lema anterior, $m = p^k$. De la mateixa forma, $u \in \mathbb{Z}[x]$ és un d'aquests factors, $f \equiv uv \pmod m$, per a un $v \in \mathbb{Z}[x]$. El nostre objectiu serà per tant buscar un factor g com el del lema, complint la condició següent:

$$m > |f|_2^{gr(g)} |g|_2^{gr(f)} \leftarrow |g|_2^n < m|f|_2^{-gr(g)},$$

ja que si aquesta condició es dóna, podem afirmar que $mcd(f,g) \in \mathbb{Z}[x]$ no és constant i obtenim un factor de $f \in \mathbb{Z}[x]$.

Si plantegem els coeficients possibles del polinomi g com un vector d'incògnites, podem plantejar la seva cerca en termes d'una xarxa. Suposem que $j \in \{d+1, \ldots, n\}$, on $d = gr(u) < n$. Representarem un polinomi g de grau $gr(g) < j$ com a $(g_{j-1}, \ldots, g_1, g_0) \in \mathbb{Z}^j$ on $g = g_{j_1} x^{j-1} + \cdots + g_1 x + g_0$. Per a trobar-lo haurem de generar la base de la xarxa $L \subseteq \mathbb{Z}^j$ a partir de u com a continuació:

$$\{u, xu, \ldots, x^{j-d-1}u\} \cup \{m, mx, \ldots, mx^{d-1}\},$$

formada per j vectors linealment independents, ja que $gr(u) = d$. Els elements $g' \in L$ prendran la forma següent:

$$\sum_{i=0}^{j-d-1} a_i x^i u + \sum_{i=0}^{d-1} b_i m x^i = \left(\sum_{i=0}^{j-d-1} a_i x^i \right) u + m \left(\sum_{i=0}^{d-1} b_i x^i \right) = au + mb$$

on $a_i, b_i \in \mathbb{Z}$ i $a = \sum_{i=0}^{j-d-1} a_i$, $b = \sum_{i=0}^{d-1} b_i$. Amb això, $g' = au + mb$ i $g \equiv au \pmod m$ és u és factor de g' mòdul $m = p^k$. Això implica que si $g' \in L$, llavors $gr(g) < j$ i $u|g \pmod m$. Volem demostrar també el sentit oposat.

Suposem que $g' \in \mathbb{Z}[x]$, $gr(g) < j$ i $u|g \pmod m$. Com que u és un divisor mòdul m de g podem escriure $g' = a_1 u + mb_1$ per a $a_1, b_1 \in \mathbb{Z}[x]$. Per ser u mònic, podem escriure $b_1 = a_2 u + b_2$ amb $a_2, b_2 \in \mathbb{Z}[x]$ i $gr(b_2) < gr(u)$, que obtenim en dividir b_1 per u. Tenim, per tant, que

$$g' = a_1 u + m(a_2 u + b_2) = (a_1 + a_2 m)u + mb_2.$$

Amb això hem expressat g' com $g' = au + mb$ per $a = a_1 + a_2 m$ i $b = b_2$, on $gr(a_1) \leq gr(g) - gr(u) < j - d$ i $gr(a_2) \leq gr(b_1) - gr(u) < j - d$ i per tant $gr(a) < j - d$ i $g' \in L$. Hem vist doncs que

$$g' \in L \Leftrightarrow gr(g') < j,\ u|g' \pmod m$$

Ara que tenim els elements de g' que compleixen les condicions necessàries del lema, si aconseguim a més trobar un element que satisfaci

$$|g|_2^n < m|f|_2^{-gr(g)}$$

tindrem que g és un factor no trivial de f. Si utilitzem l'algoritme LLL amb $\delta = \frac{3}{4}$, sabem que $|g_1|_2 \leq 2^{(j-1)/2}|g'|_2$, amb g_1 el primer vector de la base reduïda i g' qualsevol element de L. Per ser $dim(L) = j$,

$$|g_1|_2 \leq 2^{n/2}|g'|_2$$

Com que g' pot ser qualsevol element de L, prenem g un factor irreductible de f divisible per u mòdul m. Amb això, aplicant la fita de Mignotte del lema 8.4:

$$|g|_\infty \leq |g|_2 \leq (n+1)^{1/2}2^n A, \quad A = max(|f|_\infty, |g|_\infty);$$

Per tant:

$$|g_1|_2 \leq 2^{n/2}B, \quad B = (n+1)^{1/2}2^n A$$

Que implica

$$|g_1|_2^{j-1}|g|_2^{deg(g_1} < (2^{n/2}B)^n B^n = 2^{n^2/2}(n+1)^{1/2}2^{2n^2}A^{2n} \leq p^k = m$$

Gràcies al valor de k que hem escollit amb anterioritat. Aplicant aquí el lema 8.5 sabem que $mcd(g, g_1)$ no és constant, i per tant g_1 ens donarà un factor no trivial de f. Si ara prenem $h = mcd(g_1, f) \neq 0$ sabem que $g|h$ i que el grau és més petit que j. Això ens dona un factor de f, que pot ser g o $h = gk$ amb $deg(k) < j - deg(h)$. En cas que no sigui g, només haurem de tornar a aplicar l'algoritme sobre h per trobar g. Haurem d'aplicar aquest procediment a tots els u que haguem trobat, i aplicar l'algoritme recursivament sobre els factors h_u no irreductibles que trobem.

Referències

[1] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982. URL: http://dx.doi.org/10.1007/BF01457454, doi:10.1007/BF01457454.

[2] Gabriel Cramer. Introduction à l'Analyse des lignes Courbes algébriques. *Geneve*, page 4 Math.p. 91, 1750.

[3] Ravi Kannan. Improved algorithms for integer programming and related lattice problems. pages 193–206, 1983. URL: http://doi.acm.org/10.1145/800061.808749, doi:10.1145/800061.808749.

[4] Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. pages 601–610, 2001. URL: http://doi.acm.org/10.1145/380752.380857, doi:10.1145/380752.380857.

[5] C P Schnorr and M Euchner. Lattice Basis Reduction: Improved Practical Algorithms and Solving Subset Sum Problems. pages 1–27, 1993.

[6] Joachim Von Zur Gathen and Jürgen Gerhard. Modern computer algebra. 2013.

[7] Prof J Buchmann. *PQCrypto Proceedings 2016*. 2010. arXiv:arXiv:1011.1669v3, doi:10.1007/978-3-642-38616-9.

[8] Eduardo Morais Ricardo Dahab. Lattice-based cryptography. 2016.

[9] Daniele Micciancio and San Diego. Shortest Vector Problem (1982 ; Lenstra , Lenstra , Lovasz). *Magma*, 1:1–5, 1982.

[10] Jonathan Kelner. Brief Review of Gram-Schmidt and Gauss's Algorithm. *Lecture*, pages 1–7, 2009. URL: https://ocw.mit.edu/courses/mathematics/18-409-topics-in-theoretical-computer-science-an-algorithmists-toolkit-fall-2009/lecture-notes/MIT18_409F09_scribe19.pdf.

[11] Murray R. Bremner. *Lattice Basis Reduction: An Introduction to the LLL Algorithm and Its Applications*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 2011.

Índex alfabètic

A. K. Lenstra, H. W. Lenstra Jr. i L. Lovász, 11
Algoritme LLL, 10, 11, 17, 24, 28, 30, 32
  Base δ-LLL Reduïda, 22
  Base δ-LLL reduïda, 11

Blichfeldt, 6

Minkowski, 6, 7

ortogonalització de Gram-Schmidt, 2, 5, 7, 11–13, 15

Problema del Vector més Curt, 7
  Cerca SVP_γ, 8
  Decisional, 8
  Optimització, 8
  Optimització SVP_γ, 8
  Promesa SVP_γ, 8
Problema del vector més curt
  Cerca, 7
Problema del Vector més Proper, 8
  Optimització CVP_γ, 9
  Promesa CVP_γ, 9
  Cerca CVP_γ, 8

xarxa, 3
  base mínima, 10
  cel·la, 3
  mínims successius, 5
  vector més curt, 5