Algorithms for Isometries of Lattices

Malonguemfo Teagho Amandine

Supervised by: Dr. Aurel Page
Institut de Mathématiques de Bordeaux (IMB), France

14 June 2018

A thesis submitted in Partial Fulfillement of the Requirements for the Masters Degree of Science at the University of Bordeaux

Contents

Abstract

In this essay, we study, describe and implement in Sage 8.1 algorithms to compute short vectors, automorphisms and isometries of lattices. Concerning the short vectors of lattices, we present a method of computation and we study the complexity analysis. Then, we compute of the group of automorphisms of lattices. For this purpose, we give the naive algorithm, the algorithm using the fingerprint and a computation time measurement of these algorithms. Finally, we present three methods for computing isometries between lattices followed by a runtime of these methods on an example.

Chapter 1

Introduction

A lattice is an abelian group $\mathcal{L}$ isomorphic to $\mathbb{Z}^n$ and given together with a positive definite bilinear form $\Phi : \mathcal{L} \times \mathcal{L} \to \mathbb{Z}$. Lattices have been used in Mathematics for many years. This date back to the 18th century when Mathematicians such as Gauss and Lagrange used lattices in number theory. Lagrange considered lattice basis reductions in dimension two in [6] and Gauss studied what is known as the Gauss' Circle in [7]. After that, many contribution on the field were made namely in 1910, Minkowski greatly advanced the study of lattices in his "Geometry of numbers" [8], after which other texts followed in this field. Then early in 1980s, Lenstra, Lenstra and Lovasz discovered their famous LLL basis-reduction algorithm [9] to reduce lattice bases. The applications of this method include factoring integer polynomials and breaking several cryptosystems. Lattices have many applications in computer science and mathematics, including the solution of integer programming problems, the design of secure cryptographic systems and others. Many computational problems and algorithms highly use the enumeration of short vectors of lattices. In 1985, the Mathematicians Finke and Pohst published a method [4] for computing all lattice elements whose norm was less than a given constant. In 1997, Plesken and Souvignier published an algorithm [1] for computing the group of automorphisms of Euclideans lattices using these short vectors. The main point of this project is to study this algorithm and make a slightly modified version to find isometries between lattices. Being able to test whether two lattices are isometric or not allows to construct the genus of lattices that are useful in the study of modular forms, arithmetic groups and others. This essay is organised as follows: in chapter 2, we will give a background information on lattices; The third chapter deals with the description of an algorithm to compute short vectors of lattices, the study of the complexity analysis of these algorithms and a computation time of these algorithms on an example; The enumeration of these short vectors is used in chapter 4 and 5; Chapter 4 covers algorithms for computing the group of automorphisms of a given lattice; It also presents a computation time measurement of these algorithms; Chapter 5 illustrates how to adapt the method from chapter 4 to compute isometries between two lattices; It also covers the main algorithm for computing isometries of lattices using automorphisms of the first lattice and only one isometry between the lattices involved; We end with a general conclusion. We describe each algorithm by assembling several simpler sub-algorithms. We write all code in Sage 8.1.

Conventions Let M be an arbitrary element of some set. In the whole work:

- We denote by M^T the transpose of M and by M^{-1} its inverse M;
- $\langle u, v \rangle$ denotes the inner product between two vectors u and v;

- We denote by either $|C|$ or $card(C)$ the cardinal of an arbitrary set C;
- We use only symmetric bilinear forms.

Chapter 2

Background

The term 'lattice' is used in many contexts and has many definitions. One may consider a lattice as a regular arrangement of points on an n-dimensional grid. We might also describe it as a discrete subgroup of $\mathbb{R}^n$ with a basis. This section presents an overview not only on lattice, but also on the key words that will be used later on. We start with some definitions and examples. To study the elements of lattices, we consider the following definitions.

Definition 1 *A **lattice** $\mathcal{L}$ is the set of all integer linear combinations of a basis of an $\mathbb{R}$-vector space. Letting $B = (b_1, \cdots, b_n)$ be a matrix (with column the b_i) of linearly independent vectors in $\mathbb{Z}^m$ with integer coefficients, the lattice $\mathcal{L}$ generated by B is expressed as:*

$$\mathcal{L} = \{Bx : x \in \mathbb{Z}^n\} = \Big\{ \sum_{i=1}^{n} x_i b_i : x_i \in \mathbb{Z} \Big\}$$

If $\mathcal{L}$ is a lattice generated by B, we have the following:

*a) The **matrix** B is called a basis matrix for the lattice $\mathcal{L}$.*

*b) The **rank** of the lattice $\mathcal{L}$ is the number of basis vectors generating $\mathcal{L}$; it is the integer n in this case.*

c) If $n = m$, then $\mathcal{L}$ is called a full rank lattice.

Definition 2 *Let $\mathcal{L} = \{Bx : x \in \mathbb{Z}^n\}$ be a lattice (with B defined as above). An element v of $\mathcal{L}$ is a vector of the form $v = Bx$. Here:*

- *v is called the **embedded vector**,*
- *x is called the **coordinate vector** (as it is the coordinates of v in the basis vector B).*

Example 3 *Consider the lattice $\mathcal{L}$ generated by the basis $B = \begin{pmatrix} 0 & 1 & 0 \\ 2 & 0 & 1 \\ 0 & -1 & 1 \end{pmatrix}$. We have the following:*

- $rk(\mathcal{L}) = 3$;
- *The coordinate vectors of* $b_1 = (0,2,0)^T$, $b_2 = (1,0,-1)^T$ *and* $b_3 = (0,1,1)^T$ *are respectively* $(1,0,0)^T$, $(0,1,0)^T$, *and* $(0,0,1)^T$.
- $(1,1,1)^T$ *is coordinate vector of the embedded vector* $(1,3,0)^T$. *Indeed,*
$$\begin{pmatrix} 0 & 1 & 0 \\ 2 & 0 & 1 \\ 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \\ 0 \end{pmatrix}$$

2.1 Quadratic and bilinear forms

Definition 4 *A* ***bilinear form*** *on a lattice* $\mathcal{L}$ *is a map* $\Phi : \mathcal{L} \times \mathcal{L} \to \mathbb{Z}$ *that is linear in each variable. More precisely,* $\forall u, v, w \in \mathcal{L}$ *and for any scalar* $\alpha \in \mathbb{Z}$:

- $\Phi(u, v + w) = \Phi(u, v) + \Phi(u, w)$;
- $\Phi(u + v, w) = \Phi(u, w) + \Phi(v, w)$;
- $\Phi(\alpha u, v) = \alpha\Phi(u, v)$;
- $\Phi(u, \alpha v) = \alpha\Phi(u, v)$.

A bilinear form is ***symmetric*** *if* $\Phi(x, y) = \Phi(y, x) \quad \forall x, y \in \mathcal{L}$.

Definition 5
- Φ *is said to be* ***positive definite*** *if* $\Phi(x, x) > 0 \quad \forall x \neq 0$;
- *Throughout this work, for* $(\mathcal{L}, \Phi)$ *a lattice equipped with a symmetric positive definite bilinear form* Φ, *with* $\mathcal{L} = \{Bx : x \in \mathbb{Z}^n, n \in \mathbb{N}\}$, *the bilinear form* Φ *will be defined as follows:*
For $u = Bx$, $v = By$, *two embedded vectors, with* x *and* y *the coordinate vectors of* u *and* v *respectively, we define:*
$$\Phi(x, y) = x^T B^T B y$$
Notice that, if Ψ *is the bilinear form (used for embedded vectors) associated to* $\mathcal{L}$, *then one has:*
$$\Psi(u, v) = \langle u, v \rangle = \langle (Bx)^T, (By) \rangle = x^T B^T B y = \Phi(x, y)$$
.

Definition 6 *A* ***quadratic form*** *is a map* $Q : x \mapsto \Phi(x, x)$, *where* Φ *is a bilinear form.*

- *A quadratic form* Q *is said to be* ***positive definite*** *if* $Q(x) > 0 \quad \forall x \neq 0$;
- *A lattice* $\mathcal{L}$ *equipped with a quadratic form* Q *is a said to be* ***positive definite*** *if* Q *is positive definite;*

- *For Q a quadratic form defined as above, one has: $\Phi(x,y) = \frac{1}{2}(Q(x+y) - Q(x) - Q(y))$.*

Example 7 *Using the lattice defined on the first example, for $x, y \in \mathbb{Z}^3$ arbitrary coordinate vectors, we have: $\Phi(x,y) = x^T B^T B y$, with $B = \begin{pmatrix} 0 & 1 & 0 \\ 2 & 0 & 1 \\ 0 & -1 & 1 \end{pmatrix}$.*

Definition 8 *In the context of lattices, the norm is simply the quadratic form of the lattice. In order words, the **norm** of an embedded vector $v = Bx$ is given by $\Psi(v,v) := \langle v, v \rangle$; Hence the norm of its coordinate vector x is $\Phi(x,x) = x^T B^T B x$.*

Example 9 *Consider the basis $B = \begin{pmatrix} 0 & 1 & 0 \\ 2 & 0 & 1 \\ 0 & -1 & 1 \end{pmatrix}$ defined as in the first example, then the norm of the embedded vector $(1,3,0)^T$ is 10 and the norm of its coordinate vector $x = (1,1,1)^T$ computed using the formula $\Phi(x,x) = x^T B^T B x$ equals 10.*

Remark 10
- *The definition of Φ, the norm of an embedded vector is the same as the norm of its associated coordinate vector;*
- *In general cases, the norm is the square root of the quadratic form.*

Definition 11 *Let $(\mathcal{L}, \Phi)$ be a lattice generated by a basis $B = (b_1, \cdots, b_n)$. The **Gram matrix** of $\mathcal{L}$ with respect to Φ is the matrix F with coefficients $F_{ij} = \Psi(b_i, b_j)$ for $1 \leq i, j \leq n$ (where Ψ is the bilinear form of $\mathcal{L}$ used for embedded vectors). In other words, $F = B^T B$.*
*The **determinant** of a lattice is the determinant of its Gram matrix.*

Example 12 *The Gram matrix of the lattice given with the basis $B = \begin{pmatrix} 0 & 1 & 0 \\ 2 & 0 & 1 \\ 0 & -1 & 1 \end{pmatrix}$, is:*

$$F = B^T B = \begin{pmatrix} 0 & 2 & 0 \\ 1 & 0 & -1 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 2 & 0 & 1 \\ 0 & -1 & 1 \end{pmatrix} = \begin{pmatrix} 4 & 0 & 2 \\ 0 & 2 & -1 \\ 2 & -1 & 2 \end{pmatrix}.$$

Remark 13 *Using the Gram matrix F of a lattice $(\mathcal{L}, \Phi)$ with basis vector B, we can generalise the expression of Φ by setting:*

$$\Phi(x,y) = x^T F y \text{ where } x,\ y \text{ are coordinate vectors.}$$

While working on the algorithm, we will use this definition of Φ.

2.2 Matrix Decomposition

In linear algebra, a matrix decomposition or matrix factorisation is a factorisation of a matrix into a product of matrices. There are many matrix decompositions. In this section, we present some methods of decompositions for matrices.

2.2.1 The Cholesky decomposition

The reference for this section is [3].

Definition 14 *The **Cholesky decomposition** of a real-valued symmetric positive definite matrix A is a decomposition of the form $A = LL^T$, where L is a lower triangular matrix with real and strictly positive diagonal entries, and L^T denotes the transpose of L. The matrix L is called the Cholesky factor of A.*

We consider the Cholesky decomposition of n by n matrix $A = (a_{ij})_{1\leq i,j\leq n}$. In terms of coordinates, if $L = (l_{ij})_{1\leq i,j\leq n}$ (with $l_{ij} = 0$ for $i < j$) then we have:

$$\begin{cases} l_{jj} = \left(a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2\right)^{\frac{1}{2}} \\ l_{ij} = \frac{1}{l_{jj}}\left(a_{ij} - \sum_{k=1}^{j-1} l_{ik}l_{jk}\right) \end{cases}$$

Remark 15

- *This decomposition is very useful to solve linear equation of the form $Ax = b$ when A is a symmetric and positive definite matrix; b is a column vector and x is the indeterminate vector. This is done by first writing A as $A = LL^T$, then solving the equation $Ly = b$ for y, finally solving the equation $L^Tx = y$ for x. Solving these equation will be easier as L and L^T are respectively lower and upper triangular matrices.*
- *The computation of the Cholesky factor of a matrix requires the use of the square roots as the Cholesky factor of a matrix may contain entries that are square roots. But this is not the case for the following decomposition.*

2.2.2 LDL^T-decomposition

A closely related variant of the Cholesky decomposition is the LDL^T-decomposition.

Definition 16 *The LDL^T**-decomposition** of a real-valued symmetric positive definite matrix A is a decomposition of the form $A = LDL^T$, where L is a lower unit triangular matrix (i.e triangular matrix with 1's on the diagonal entries) with real entries, L^T is the transpose of L, and D is a diagonal matrix.*

We consider the LDL^T-decomposition of n by n matrix $A = (a_{ij})_{1\leq i,j\leq n}$. In terms of coordinates, if $L = (l_{ij})_{1\leq i,j\leq n}$ (with $l_{ij} = 0$ for $i < j$ and $l_{ii} = 1$ for $1 \leq i \leq n$) and D has as diagonal entries $(d_{jj})_{1\leq j\leq n}$ then we have:

$$\begin{cases} d_{jj} = a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2 d_{kk} \\ l_{ij} = \frac{1}{d_{jj}}\left(a_{ij} - \sum_{k=1}^{j-1} l_{ik}l_{jk}d_{kk}\right) \end{cases}$$

Remark 17

- *If a matrix has a LDL^T-decomposition then it has also a Cholesky decomposition; But the converse is not always true.*

- *If it is efficiently implemented, the LDL^T-decomposition requires the same space and computational complexity to construct and to use, but avoids extracting square roots. For these reasons, one may prefer the LDL^T-decomposition.*

2.2.3 LLL-reduction

Gram-Schmidt orthogonalisation : Let $(b_1, \cdots, b_n)$ be linearly independent vectors. Their Gram-Schmidt orthogonalisation (GSO) $(b_1^*, \cdots, b_n^*)$ is the orthogonal family defined recursively as follows:

a) The vector b_i^* is the component of the vector b_i that is orthogonal to the linear span of the vectors $b_1, \cdots, b_{i-1}$;

b) $b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{ij} b_j^*$, where $\mu_{ij} = \frac{\langle b_i, b_j^* \rangle}{||b_j^*||^2}$;

c) For $i \leq n$, we let $\mu_{ii} = 1$.

Notice that the GSO family depends on the order of the vectors. If the b_i's are integer vectors, then the b_i^*'s and the μ_{ij}'s are rational.

Definition 18 *A basis $(b_1, \cdots, b_n)$ is **size reduced** if its GSO family satisfies*

$$|\mu_{ij}| \leq \frac{1}{2}, \quad \forall\ 1 \leq j \leq i < n.$$

Definition 19 *A basis $(b_1, \cdots, b_n)$ is **LLL-reduced** if it is size reduced and if its GSO satisfies the $(n-1)$ Lovasz conditions:*

$$\frac{3}{4}||b_{k-1}^*||^2 \leq ||b_k^* + \mu_{kk} b_{k-1}^*||^2.$$

Remark 20 *The LLL reduction algorithm provide an orthogonal basis whose vectors' norms are bounded. This algorithm is often used to reduce large basis to smaller basis. Such a basis reduces the computation such as sums and products of the column vectors of the matrix formed by that basis. See [5] for more explanations of the LLL-reduction.*

2.3 Automorphisms and isometries

What follows is an outline on the definitions of automorphisms and isometries of a lattice.

Definition 21 *An **automorphism** of a lattice $(\mathcal{L}, \Phi)$ is a $\mathbb{Z}$-linear bijection $\theta : \mathcal{L} \to \mathcal{L}$ satisfying:*

$$\Phi(x, y) = \Phi(\theta(x), \theta(y)) \quad \forall x, y \text{ coordinate vectors in } \mathcal{L}.$$

*The **automorphism group** of a lattice $(\mathcal{L}, \Phi)$ is the set $Aut(\mathcal{L})$ of all automorphisms defined on $\mathcal{L}$.*

Lemma 22 *let $\circ$ be the composition law. $(Aut(\mathcal{L}), \circ)$ is a group.*

PROOF. Indeed, $(Aut(\mathcal{L}), \circ)$ satisfies:

- The identity map of $\mathcal{L}$ is by definition an automorphism on $\mathcal{L}$, so it belongs to $Aut(\mathcal{L})$;
- For all $\theta_1, \theta_2 \in Aut(\mathcal{L})$, we have $\theta_1 \circ \theta_2 \in Aut(\mathcal{L})$, as $\theta_1 \circ \theta_2$ is a $\mathbb{Z}-$linear bijection; moreover since θ_1 and θ_2 are automorphisms on $\mathcal{L}$, then for all x, y coordinate vectors in $\mathcal{L}$, we have:

$$\Phi(\theta_1 \circ \theta_2(x), \theta_1 \circ \theta_2(y)) \;=\; \Phi(\theta_2(x), \theta_2(y)) \;=\; \Phi(x, y);$$

- For all $\theta_1, \theta_2, \theta_3 \in Aut(\mathcal{L})$, $\theta_1 \circ (\theta_2 \circ \theta_3) = (\theta_1 \circ \theta_2) \circ \theta_3$;
- For all $\theta \in Aut(\mathcal{L})$, θ^{-1} is also an automorphism of $\mathcal{L}$; so $\theta^{-1} \in Aut(\mathcal{L})$.

Definition 23 *Let us consider two lattices $(\mathcal{L}, \Phi)$ and $(\mathcal{D}, \Psi)$ with basis B and D respectively. An **isometry** is a $\mathbb{Z}$-linear bijection g from $(\mathcal{L}, \Phi)$ to $(\mathcal{D}, \Psi)$ that preserves the bilinear forms Φ and Ψ. More precisely,*

$$\Phi(x, y) = \Psi(g(x), g(y)) \quad \forall x, y \text{ coordinate vectors in } \mathcal{L}.$$

*Two lattices are said to be **isometric** if there exists an isometry between them.*
The set of all isometries on a given lattice $\mathcal{L}$ is called the orthogonal group of $\mathcal{L}$.

Proposition 24 *The composition of two isometries is an isometry.*

PROOF. Let us consider three lattices $(\mathcal{L}_1, \Phi_1)$, $(\mathcal{L}_2, \Phi_2)$ and $(\mathcal{L}_3, \Phi_3)$; let g_1 and g_2 be two isometries from $(\mathcal{L}_1, \Phi_1)$ to $(\mathcal{L}_2, \Phi_2)$ and from $(\mathcal{L}_2, \Phi_2)$ to $(\mathcal{L}_3, \Phi_3)$ respectively. We have:

- By definition $g_2 \circ g_1$ is $\mathbb{Z}$-linear bijection;
- Moreover, for all x, y coordinate vectors in $\mathcal{L}_1$, we have:

$$\Phi(g_2 \circ g_1(x), g_2 \circ g_1(y)) \;=\; \Phi(g_1(x), g_1(y)) \;=\; \Phi(x, y).$$

So $g_2 \circ g_1$ is an isometry.

Chapter 3

Computing Short vectors

The method of computation of isometries between two lattices that we will describe relies on the use of the set of short vectors of these lattices. A short vector of a lattice $(\mathcal{L}, \Phi)$ with basis B is a vector v of the lattice $\mathcal{L}$ with norm less than or equal to the maximal norm of the basis vectors of $\mathcal{L}$. All short vectors of a given lattice $(\mathcal{L}, \Phi)$ with basis B, form a set called the set of short vector of $\mathcal{L}$. We denote this set by S, and it is defined by:

$$S = \{v \in \mathcal{L} : \Phi(v, v) \leq M\},$$

where M is the maximum norm of the basis vector of $\mathcal{L}$; More precisely, considering $B = (b_1, \cdots, b_n)$ as the basis vector of $\mathcal{L}$ and F the associated Gram matrix, then

$$M = \max_{1 \leq i \leq n}(\Phi(b_i, b_i)) = \max_{1 \leq i \leq n} F_{ii}.$$

Remark that set S depends not only on the lattice but also on the basis of this lattice. In this section, we present an algorithm to determine S, i.e all the short vectors of a given lattice $(\mathcal{L}, \Phi)$. Before going through this algorithm, we have to be sure that if any lattice $\mathcal{L}$ has a finite number of short vectors. If it is the case, then we are sure that our algorithm will terminate.

Proposition 25 *Consider as above the set S of short vectors of a given lattice $(\mathcal{L}, \Phi)$ with basis $B = (b_1, \cdots, b_n)$, Gram matrix F, and maximal norm of the basis vectors M.*
Then the set S is finite.

Proof. Let x be the coordinate vector of an element of S, suppose $x \neq 0$. As F is positive definite, all its eigenvalues are strictly positive. Let $\alpha \in \mathbb{R}_+$ be the minimum eigenvalue of F.

By definition of x, and since $\alpha x < Fx$, then we have:

$$\begin{aligned}\Phi(x,x) = x^T Fx \leq M &\Rightarrow \alpha||x||^2 < x^T Fx \leq M,\\ &\Rightarrow \alpha||x||^2 < M,\\ &\Rightarrow ||x|| \leq \sqrt{\frac{M}{\alpha}},\\ &\Rightarrow x \in \mathcal{B}_{\mathbb{R}^n}\Big(0, \sqrt{\frac{M}{\alpha}}\Big),\\ &\text{As } x \text{ is the coordinate vector of an element of S then } x \in \mathbb{Z}^n,\\ &\Rightarrow \{x = (x_1, \cdots, x_n) \in \mathbb{Z}^n : \sum_{i=1}^{n} x_i b_i \in S\} \subseteq \mathbb{Z}^n \cap \mathcal{B}_{\mathbb{R}^n}\Big(0, \sqrt{\frac{M}{\alpha}}\Big);\end{aligned}$$

Set $C = \{x = (x_1, \cdots, x_n) \in \mathbb{Z}^n : \sum_{i=1}^{n} x_i b_i \in S\}$ and $E = \Big[-\Big\lfloor\sqrt{\frac{M}{\alpha}}\Big\rfloor, \Big\lfloor\sqrt{\frac{M}{\alpha}}\Big\rfloor\Big]^n$.
We have:

$$\begin{aligned}C \subseteq \mathbb{Z}^n \cap \mathcal{B}_{\mathbb{R}^n}\Big(0, \sqrt{\frac{M}{\alpha}}\Big) &\Rightarrow C \subseteq \mathbb{Z}^n \cap E,\\ &\text{But } card(\mathbb{Z}^n \cap E) = (2\Big\lfloor\sqrt{\frac{M}{\alpha}}\Big\rfloor + 1)^n < \infty,\\ &\Rightarrow card(C) \leq (2\Big\lfloor\sqrt{\frac{M}{\alpha}}\Big\rfloor + 1)^n < \infty,\\ &\text{Since the sets } S \text{ and } C \text{ are in bijection (as } S = BC) \text{ then } card(S) = card(C),\\ &\Rightarrow card(S) \leq (2\Big\lfloor\sqrt{\frac{M}{\alpha}}\Big\rfloor + 1)^n.\end{aligned}$$

Hence S is finite. Now, let us describe the algorithm.

3.1 Algorithm to Generate short vectors

Here, we describe an algorithm to compute the list of all short vectors of a given lattice lattice $(\mathcal{L}, \Phi)$ with basis $B = (b_1, \cdots, b_n)$, Gram matrix F, and maximum norm of the basis vectors M. Since any embedded vector v of S can be expressed by $v = Bx$ and the norm of v is $\Phi(x, x)$, with x its corresponding coordinate vector, therefore to find each vector $v \in S$ it suffices to find all possible coordinate vector x of v, satisfying $\Phi(x, x) \leq M$. The algorithm to described uses the Cholesky decomposition.

3.1.1 Idea of the algorithm

Considering the same notation as above, let us notice that we are reduced to solve the inequality $\Phi(x, x) \leq M$ for x. Let $x = (x_1, \cdots, x_n)$ be the coordinate vector of a short vector v. The idea is to use the inequality $\Phi(x, x) \leq M$ to determine each time, a bound on each x_k (for $1 \leq k \leq n$)

and consider the solutions x as the different vectors formed by the x_k, that are obtained while each x_k takes integer values in between its bounds.
In order to bound each x_k for $1 \leq k \leq n$, we first find the LLL-reduced matrix F^* of F. This is done in order to reduce the large basis form by the columns of the matrix F into a basis F^* whose vectors' norms are bounded. Such a basis reduces the computation such as sums and products of the column vectors of the matrix formed by that basis. After finding the LLL-reduced matrix F^* of F, we compute the Cholesky decomposition of F^*. More precisely, we write F^* as $F^* = LL^T$, where L is a lower triangular matrix with positive and non zero diagonal entries.

Bounds of the $x_k,\ 1 \leq k \leq n$

Assuming that $F^* = LL^T$, let L be defined by:

$$L = \begin{pmatrix} l_{11} & 0 & \cdots & \cdot & 0 \\ l_{21} & l_{22} & \cdot & \cdots & \cdot \\ \vdots & \cdot & \cdot & \cdot & \vdots \\ \cdot & \cdot & \cdot & \cdot & 0 \\ l_{n1} & \cdots & \cdot & \cdot & l_{nn} \end{pmatrix}, \text{so } L^T = \begin{pmatrix} l_{11} & l_{21} & \cdots & \cdot & l_{n1} \\ 0 & l_{22} & \cdot & \cdots & l_{n2} \\ \vdots & \cdot & \cdot & \cdot & \vdots \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdots & \cdot & \cdot & l_{nn} \end{pmatrix}$$

Using the fact that $F = LL^T$, we have: $\Phi(x,x) = x^T LL^T x = (L^T x)^T.(L^T x) = ||L^T x||^2$,

$$L^T x = \begin{pmatrix} l_{11} & l_{21} & \cdots & \cdot & l_{n1} \\ 0 & l_{22} & \cdot & \cdots & l_{n2} \\ \vdots & \cdot & \cdot & \cdot & \vdots \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdots & \cdot & \cdot & l_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \cdot \\ x_n \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{n} l_{i1} x_i \\ \vdots \\ \sum_{i=k}^{n} l_{ik} x_i \\ \vdots \\ l_{nn} x_n \end{pmatrix}$$

As x satisfies $\Phi(x,x) \leq M$, then:

$$\Big(\sum_{i=1}^{n} l_{i1} x_i\Big)^2 + \cdots + \Big(\sum_{i=k}^{n} l_{ik} x_i\Big)^2 + \cdots + \Big(l_{nn} x_n\Big)^2 \leq M \tag{3.1}$$

First, this inequality implies that:

$$\Big(l_{nn} x_n\Big)^2 \leq M \Rightarrow -\frac{\sqrt{M}}{l_{nn}} \leq x_n \leq \frac{\sqrt{M}}{l_{nn}}$$

So the lower bound and upper bound of x_n are $Lb_n = \Big\lceil -\frac{\sqrt{M}}{l_{nn}} \Big\rceil$ and $Ub_n = \Big\lfloor \frac{\sqrt{M}}{l_{nn}} \Big\rfloor$ respectively. The inequality (3.1) implies that for all $1 \leq k \leq n-1$, at the step k, we have:

$$\Big(\sum_{i=k}^{n} l_{ik}x_i\Big)^2 + \cdots + \Big(l_{nn}x_n\Big)^2 \leq M \Leftrightarrow \Big(\sum_{i=k}^{n} l_{ik}x_i\Big)^2 \leq M - \Big[\Big(\sum_{i=k+1}^{n} l_{ik+1}x_i\Big)^2 + \cdots + \Big(l_{nn}x_n\Big)^2\Big],$$

$$\text{Set } U_k = \sum_{j=k+1}^{n} \Big(\sum_{i=j}^{n} l_{ij}x_i\Big)^2, \text{ and } U_n = 0, \tag{3.2}$$

$$\Leftrightarrow \Big|l_{kk}x_k + \sum_{i=k+1}^{n} l_{ik}x_i\Big| \leq \sqrt{M - U_k},$$

$$\text{Set } Y_k = \sum_{i=k+1}^{n} l_{ik}x_i, Y_n = 0 \tag{3.3}$$

$$\Leftrightarrow \frac{1}{l_{kk}}\Big[-\sqrt{M-U_k} - Y_k\Big] \leq x_k \leq \frac{1}{l_{kk}}\Big[\sqrt{M-U_k} - Y_k\Big].$$

So for $1 \leq k \leq n-1$ the lower bound and upper bound on x_k are Lb_k and Ub_k respectively.

$$\begin{cases} Lb_k = \Big\lceil \frac{1}{l_{kk}}\Big[-\sqrt{M-U_k} - Y_k\Big]\Big\rceil \\ Ub_k = \Big\lfloor \frac{1}{l_{kk}}\Big[\sqrt{M-U_k} - Y_k\Big]\Big\rfloor \end{cases} \tag{3.4}$$

Notice that, from (3.2), we have $U_{k-1} = U_k - \sum_{i=k+1}^{n} l_{i,k+1}x_i^2$ for $k = n-1, n-2, \cdots, 1$ and $U_n = 0$. Hence, one may compute U_k using this expression for each $1 \leq k \leq n-1$.

Remark 26 *Notice that for $1 \leq k \leq n$, $Lb_k \leq x_k \leq Ub_k$ is equivalent to $\Phi(x,x) \leq M$. So on the algorithm, for any integer value of x_k between its bounds, the corresponding x must satisfies the inequality $\Phi(x,x) \leq M$.*

3.1.2 Description of the algorithm

Considering the same notation as above, our algorithm works as follows:

a) We first find the LLL-reduced matrix F^* of F by applying the LLL-reduction algorithm to F (See [3]);

b) We compute the Cholesky factor L of the matrix F^* (i.e L satisfies $F^* = LL^T$) using the Cholesky decomposition of F^* (See [3]);

c) We compute $Lb_n = \Big\lceil -\frac{\sqrt{M}}{l_{nn}} \Big\rceil$, $Ub_n = \Big\lfloor \frac{\sqrt{M}}{l_{nn}} \Big\rfloor$, we initialise $x_n = Lb_n$, and we set $N_n = l_{nn}x_n$, $U_n = 0$, and $Y_n = 0$;

d) Using the value of x_n, we compute starting from $k = n-1$ going down to $k = 1$, U_k, Y_k, Lb_k and Ub_k(using the formulas (3.2), (3.3) and (3.4)) and set $x_k = Lb_k$(in order to compute the next coordinate x_{k+1});

e) When we reach $k = 1$(it means that we have found one element of S) we keep the first vector x found; Then we increment $x_1(x_1 = x_1 + 1)$ and we keep the vectors x found until $x_1 = Ub_1$;

f) When $x_1 > Ub_1$, we first increment $x_2(x_2 = x_2 + 1)$; Secondly, we compute again x_1 using U_1, Y_1, Lb_1 and Ub_1 (which are computed now using the new value of x_2), then we go to d). x_2 will be incremented until $x_2 = Ub_2$; The step f) will be done for each $k = 1, \cdots, n$. Therefore this step can be generalised by the step f') as follows;

f') If $x_k > Ub_k$, then we first increment $x_K(x_k = x_k+1$; This is done each time $x_{k+1} \leq Ub_{k+1})$; Secondly, we compute again x_j using U_j, Y_j, Lb_j and Ub_j(which are computed now using the new value of x_{j+1}) for $j = k$ going down to $j = 1$.

Remark 27

- *The steps c) up to f') will be done until $x_n = Ub_n$ or whenever we found x equals to the zero vector.*
- *Since if $\Phi(x, x) = \Phi(-x, -x)$, if x belong to S, then $-x$ belongs also to S. So each time we will find a solution x of the inequality $\Phi(x, x) \leq M$ using our algorithm, we will also add its opposite $-x$ to the list of the solutions.*

Pseudocode

We consider the lattice $(\mathcal{L}, \Phi)$ as above.
Input: F (the Gram matrix of the lattice) and M (the maximum of the norm of the basis vector of the lattice)
Output: S (the list of coordinates of short vectors of the lattice)

Algorithm 1 Set of short vectors with Cholesky decomposition

procedure $S - elements(F, M)$
  $F^* \leftarrow$ LLL-reduction(F)
  $L \leftarrow$ Cholesky-decomposition(F^*)
  $n \leftarrow$ number of columns of L
  $x \leftarrow [x_1, \cdots, x_n],\ fillx_n \leftarrow 0$
  Compute $Lb_n \leftarrow \left\lceil -\frac{\sqrt{M}}{l_{nn}} \right\rceil, Ub_n \leftarrow \left\lfloor \frac{\sqrt{M}}{l_{nn}} \right\rfloor$
  Initialise $x_n \leftarrow Lb_n, N_n \leftarrow l_{nn}x_n$
  Initialise $U_n \leftarrow 0, Y_n \leftarrow 0,\ and\ k \leftarrow n-1$
  while $x_n \leq Ub_n$ **do**
    if $fillx_k = 0$ **then**
      Compute $N_k,\ U_k\ and\ Y_k$(using the formulas (3.2), (3.3) and (3.4))
      Set $x_k \leftarrow Lb_k$
      $fillx_k \leftarrow 1$
    else
      $x_k \leftarrow x_k + 1$
    if $x_k <= Ub_k$ **then**
      $k \leftarrow k - 1$
    else
      $fillx_k \leftarrow 0$
      $k \leftarrow k + 1$
    if $k = 0$ **then**
      while $x_1 \leq Ub_1$ **do**
        if $x = zero - vector$ **then**
          $append(x, S)$
          return S
        else
          $append(x, S)$
          $append(-x, S)$
        $x_1 \leftarrow x_1 + 1$
      $fillx_1 \leftarrow 0$
      $k \leftarrow 1$

Remark 28 *One may follows the same steps to describe a method for computing short vectors using the LDL^T-decomposition instead of the Cholesky decomposition. This method will be better as it help to avoid the square root used in the Cholesky decomposition, and to work on with approached values. Moreover, it is better to use it in case one uses matrices that are not positive definite.*

3.2 Complexity Analysis

In this section, we study the complexity analysis of the algorithm to compute short vectors of a lattice using the Cholesky decomposition. This study follows the method from [4], but we reorganised the proof and added the details that were omitted by Fincke and Pohst. Recall that to find the short vectors of the lattice $\mathcal{L}$ (defined as above) using the Cholesky decomposition, we were reduced to find all vectors $x = (x_1, \cdots, x_n) \in \mathbb{Z}^n$ satisfying $Q(x) \leq M$, where $Q(x) = \sum_{i=1}^{n} \Big(\sum_{j=i}^{n} l_{ij} x_j \Big)^2$ and M is the maximal norm of the basis vectors $(b_1, \cdots, b_n)$. To study the complexity analysis of our algorithm, we start by finding an upper bound for the number of vectors $x = (x_1, \cdots, x_n) \in \mathbb{Z}^n$ generated by the algorithm; Then, we look for the number of arithmetic operations required for the other steps of the algorithm and we deduce the result. Throughout this section, we count each multiplication, addition, extraction to squared root, and the extraction of the ceiling (or the floor) of a real number as one operation.

3.2.1 The number of vectors $x \in \mathbb{Z}^n$ generated by the algorithm

Here, we are looking for an upper bound for the number of vectors $x = (x_1, \cdots, x_n) \in \mathbb{Z}^n$ generated by the algorithm. We first find an upper bound for the number of $x = (x_1, \cdots, x_n)$ $(i \geq 1)$ satisfying $T(x) \leq M$; then we deduce form it the number $x = (x_1, \cdots, x_n)$ of generated by the algorithm. For $1 \leq i \leq n$, we define:

$$T_i(x_i, \cdots, x_n) = \sum_{k=i}^{n} \Big(\sum_{j=k}^{n} l_{ij} x_j \Big)^2 = \sum_{k=i}^{n} (l_{ii} x_i + Y_i)^2$$

where $Y_i = \sum_{j=i+1}^{n} l_{ij} x_j$, and $Y_n = 0$.

$$W_i(r) = card\Big(\{ x = (x_i, \cdots, x_n) \in \mathbb{Z}^{n-i+1} : \ T_i(x_i, \cdots, x_n) \leq r \} \Big),$$

for every positive real number r;
Our first aim is find an upper bound for the number of $(x_i, \cdots, x_n)$ $(i \geq 1)$ satisfying $T_i(x_i, \cdots, x_n) \leq M$; in other words, we want to find an upper bound of $W_i(M)$. This will be deduced from the following proposition.

Proposition 29 *Considering the same notation as above, assume that $\frac{1}{e}$ is a lower bound of l_{ii} $(1 \leq i \leq n)$. Therefore, one has:*

$$W_i(r) \leq \sum_{j=0}^{4\lfloor re \rfloor} W_{i+1}(r - \frac{j}{4})$$

To prove this proposition, we need the following lemma.

Lemma 30 *Considering the above notation and the assumption of the above proposition. If we have found using the algorithm* $(x_{i+1}, \cdots, x_n) \in \mathbb{Z}^{n-i}$ *such that* $T_{i+1}(x_{i+1}, \cdots, x_n) \leq r$, *then there exists at most* ε x_i *say* $x_{i_1}, \cdots, x_{i_\varepsilon}$ *(with* $\varepsilon := 2\lfloor\sqrt{re}\rfloor + 1$ *) such that* $T_i(x_i, x_{i+1}, \cdots, x_n) \leq r$ *and for each* x_{i_j}, *one has* $|l_{ii}x_{i_j} + Y_i| \leq \frac{j}{2}$ *(*$1 \leq j \leq \varepsilon$*).*

Proof. To prove this lemma, we proceed as follows.

a)For $i = n$, one has: $W_i(r) = card\Big(\{x_n \in \mathbb{Z} : \ (l_{nn}x_n)^2 \leq r\}\Big) = 2\Big\lfloor \frac{\sqrt{r}}{l_{nn}} \Big\rfloor + 1$.

b)Now, we suppose that we have found $(x_{i+1}, \cdots, x_n) \in \mathbb{Z}^{n-i}$ such that $T_{i+1}(x_{i+1}, \cdots, x_n) \leq r$ with the algorithm and let us find the number of x_i such that $T_i(x_i, x_{i+1}, \cdots, x_n) \leq r$.
By definition,

$$\begin{aligned} T_i(x_i, \cdots, x_n) &= \sum_{k=i}^{n} \Big(\sum_{j=k}^{n} l_{ij}x_j \Big)^2 \\ &= (l_{ii}x_i + Y_i)^2 + \sum_{k=i+1}^{n} \Big(\sum_{j=k}^{n} l_{ij}x_j \Big)^2 \\ &= (l_{ii}x_i + Y_i)^2 + T_{i+1}(x_{i+1}, \cdots, x_n) \\ &\text{Set } U_i = T_{i+1}(x_{i+1}, \cdots, x_n). \end{aligned}$$

So $T_i(x_i, x_{i+1}, \cdots, x_n) = (l_{ii}x_i + Y_i)^2 + U_i \leq r$.
Using this expression and the assumption on x_i, we get:

$$\begin{aligned} T_i(x_i, x_{i+1}, \cdots, x_n) \leq r &\Rightarrow (l_{ii}x_i + Y_i)^2 \leq r - U_i \leq r \\ &\Rightarrow |l_{ii}x_i + Y_i| \leq \sqrt{re}. \end{aligned}$$

Thus the number of x_i such that $T_i(x_i, x_{i+1}, \cdots, x_n) \leq r$ is $\varepsilon := 2\eta + 1$, with $\eta = \lfloor\sqrt{re}\rfloor$.

c)Using the result obtained in b), we know that we have at most ε possibilities for $x_i \in \mathbb{Z}$, say $x_{i_1}, x_{i_2}, \cdots, x_{i_\varepsilon}$, such that $T_i(x_i, x_{i+1}, \cdots, x_n) \leq r$. We order the x_{i_j} for $1 \leq j \leq \varepsilon$ according to

$$|l_{ii}x_{i_1} + Y_i| \leq |l_{ii}x_{i_2} + Y_i| \leq \cdots \leq |l_{ii}x_{i_\varepsilon} + Y_i|.$$

We claim that for all $1 \leq j \leq \varepsilon$,

$$\frac{j-1}{2} \leq |l_{ii}x_{i_j} + Y_i| \leq \frac{j}{2}.$$

To prove this, we use the inequality $|l_{ii}x_{i_j} + Y_i| \leq \eta$ (for $1 \leq j \leq \varepsilon$) and we distinguish three cases (since from the ordering, $|l_{ii}x_{i_{2p}} + Y_i|$ and $|l_{ii}x_{i_{2p+1}} + Y_i|$ have the same value namely p, for $0 \leq p \leq \eta$):

case 1: For $j = 1$, we have $0 \leq |l_{ii}x_{i_1} + Y_i| \leq \frac{1}{2}$ (as $|l_{ii}x_{i_1} + Y_i| = 0$;

case 2: For $j = 2p$ (with $1 \leq p \leq \eta$), we have:
$\frac{2p-1}{2} \leq |l_{ii}x_{i_j} + Y_i| \leq \frac{p}{2}$ (as $|l_{ii}x_{i_j} + Y_i| = p$);

case 3: For $j = 2p + 1$ (with $0 \leq p \leq \eta$), we have:
$p \leq |l_{ii}x_{i_j} + Y_i| \leq \frac{2p+1}{2}$ (as $|l_{ii}x_{i_j} + Y_i| = p$).
Hence for all $1 \leq j \leq \varepsilon$,

$$\frac{j-1}{2} \leq |l_{ii}x_{i_j} + Y_i| \leq \frac{j}{2}.$$

This conclude the proof of the lemma.

Now, let us proof the proposition.

PROOF. Recall that we would like to show that $W_i(r) \leq \sum_{j=0}^{4\lfloor re \rfloor} W_{i+1}(r - \frac{j}{4})$.
From the previous lemma, we know that if we have found using the algorithm $(x_{i+1}, \cdots, x_n) \in \mathbb{Z}^{n-i}$ such that
$T_{i+1}(x_{i+1}, \cdots, x_n) \leq r$, then there exists at most ε x_i say $x_{i_1}, \cdots, x_{i_\varepsilon}$ (with $\varepsilon := 2\lfloor\sqrt{re}\rfloor + 1$) such that $T_i(x_i, x_{i+1}, \cdots, x_n) \leq r$ and for each x_{i_j}, one has $|l_{ii}x_{i_j} + Y_i| \leq \frac{j}{2}$ $(1 \leq j \leq \varepsilon)$.
This implies that for $x_i = x_{i_j}$, we have:

$$\begin{aligned} T_i(x_{i_j}, x_{i+1}, \cdots, x_n) &= (l_{ii}x_{i_j} + Y_i)^2 + T_{i+1}(x_{i+1}, \cdots, x_n) \\ &= \frac{j^2}{4} + T_{i+1}(x_{i+1}, \cdots, x_n) \qquad (3.5) \\ & \qquad (3.6) \end{aligned}$$

From (3.5), notice that if $T_{i+1}(x_{i+1}, \cdots, x_n) \leq r - \frac{j^2}{4}$, then $T_i(x_{i_j}, x_{i+1}, \cdots, x_n) \leq r$.
For $j = 1, \cdots, \varepsilon$, this implies that :
each tuple $(x_{i+1}, \cdots, x_n)$ such that $T_{i+1}(x_{i+1}, \cdots, x_n) \leq r - \frac{j^2}{4}$ correspond to one tuple $(x_i, x_{i+1}, \cdots, x_n)$ such that $T_i(x_i, x_{i+1}, \cdots, x_n) \leq r$.
Therefore, we have:

$$\begin{aligned} W_i(r) &\leq \sum_{j=1}^{2\lfloor\sqrt{re}\rfloor+1} W_{i+1}(r - \frac{j^2}{4}) \\ &\leq \sum_{j=0}^{2\lfloor\sqrt{re}\rfloor+1} W_{i+1}(r - \frac{j^2}{4}) \\ &= \sum_{j=0}^{2\lfloor\sqrt{re}\rfloor} W_{i+1}(r - \frac{j^2}{4}) \qquad (3.7) \end{aligned}$$

As $r < \frac{(2\lfloor\sqrt{re}\rfloor+1)^2}{4}$ and by definition $W_i(\gamma) = 0$ if $\gamma < 0$.
Hence, by a change of variable in (3.7), we get:

$$W_i(r) \leq \sum_{j=0}^{4\lfloor re \rfloor} W_{i+1}(r - \frac{j}{4}).$$

This conclude the proof of the proposition.

From the above proposition, we can deduce that $W_i(M) \leq \sum_{j=0}^{4\lfloor re \rfloor} W_{i+1}(M - \frac{j}{4})$. Hence, an upper bound for the number of $(x_i, \cdots, x_n)$ $(i \geq 1)$ satisfying $T_i(x_i, \cdots, x_n) \leq M$ is $\sum_{j=0}^{4\lfloor M \rfloor} W_{i+1}(M - \frac{j}{4})$.

Now, let us compute an upper bound of the total number of vectors x generated by the algorithm. We denoted it by $\overline{W}_1(M)$, where $\bar{W}_i(M) = \sum_{j=i}^{n} W_j(M)$. This is will be deduced by the following proposition. In general $\overline{W}_i(r) = \sum_{j=i}^{n} W_j(r)$.

Proposition 31 *Considering the same notation, one has:*

$$\overline{W}_1(r) \leq (2\lfloor \sqrt{re} \rfloor + 1)\binom{4\lfloor re \rfloor + n - 1}{4\lfloor re \rfloor}$$

.

Proof. 1)We start by defining an induction on $\overline{W}_i(M)$. From the above proposition, we know that

$$W_k(r) \leq \sum_{j=0}^{4\lfloor re \rfloor} W_{k+1}(r - \frac{j}{4}).$$

This implies that:

$$\begin{aligned}
\overline{W}_i(r) &= \sum_{k=i}^{n} W_k(r) \\
&\leq \sum_{k=i}^{n} \sum_{j=0}^{4\lfloor re \rfloor} W_{k+1}(r - \frac{j}{4}) \\
&= \sum_{j=0}^{4\lfloor re \rfloor} \sum_{k=i}^{n} W_{k+1}(r - \frac{j}{4}) \\
&= \sum_{j=0}^{4\lfloor re \rfloor} \overline{W}_{i+1}(r - \frac{j}{4})
\end{aligned}$$

So

$$\overline{W}_i(r) \leq \sum_{j=0}^{4\lfloor re \rfloor} \overline{W}_{i+1}(r - \frac{j}{4}) \tag{3.8}$$

For $i=1$, (3.8) says that: $\overline{W}_1(r) \leq \sum_{j=0}^{4\lfloor re\rfloor} \overline{W}_2(r-\frac{j}{4})$. Replacing $\overline{W}_2(r)$ by it corresponding upper bound, we get:

$$\begin{aligned}
\overline{W}_1(r) &\leq \sum_{j=0}^{4\lfloor re\rfloor} \overline{W}_2(r-\frac{j}{4}) \\
&\leq \sum_{j=0}^{4\lfloor re\rfloor}\sum_{k=0}^{4\lfloor re\rfloor} \overline{W}_3(r-\frac{j}{4}-\frac{k}{4}) \\
&= \sum_{j=0}^{4\lfloor re\rfloor}\sum_{l=j} \overline{W}_3(r-\frac{l}{4}) \\
&= \sum_{l=0}^{4\lfloor re\rfloor}\sum_{j=0}^{l} \overline{W}_3(r-\frac{l}{4}) \\
&= \sum_{l=0}^{4\lfloor re\rfloor}(l+1)\overline{W}_3(r-\frac{l}{4}).
\end{aligned}$$

Continuing this way up to n, we get:

$$\overline{W}_1(r) \leq \sum_{l=0}^{4\lfloor re\rfloor} \lambda_{ij}\overline{W}_i(r-\frac{l}{4}), \tag{3.9}$$

where $\lambda_{ij} \in \mathbb{N}$.

2) Let us compute the λ_{ij} for $1 \leq i \leq n$ and $j =, \cdots, 4\lfloor re\rfloor$.

a) For $i=1$, since $\overline{W}_1(r) \leq \overline{W}_1(r)$ then $\lambda_{i0}=1$ and $\lambda_{ij}=0$ for $j>0$.

b) In general, as $\overline{W}_i(r-\frac{j}{4}) \leq \sum_{k=0}^{4\lfloor re\rfloor} \overline{W}_{i+1}(r-\frac{j+k}{4})$ then we have:

$$\begin{aligned}
\sum_{j=0}^{4\lfloor re\rfloor} \lambda_{ij}\overline{W}_i(r-\frac{j}{4}) &\leq \sum_{j=0}^{4\lfloor re\rfloor} \lambda_{ij}\sum_{k=0}^{4\lfloor re\rfloor} \overline{W}_i(r-\frac{j+k}{4}) \\
&= \sum_{l=0}^{4\lfloor re\rfloor}\Big(\sum_{j=0}^{l}\lambda_{ij}\Big)\overline{W}_i(r-\frac{l}{4}) \\
\sum_{j=0}^{4\lfloor re\rfloor} \lambda_{ij}\overline{W}_i(r-\frac{j}{4}) &= \sum_{l=0}^{4\lfloor re\rfloor}\Big(\sum_{j=0}^{l}\lambda_{i+1,l}\Big)\overline{W}_i(r-\frac{l}{4})
\end{aligned}$$

with $\lambda_{i+1,l} = \sum_{j=0}^{l}\lambda_{ij}$.
In general,

$$\begin{cases} \lambda_{i+1,j} = \sum_{k=0}^{j}\lambda_{ik} \\ \lambda_{1,0}=1,\ \lambda_{1,j}=0 \text{ for } j>0. \end{cases}$$

From a lemma (see [4], page 468), one can show by induction that $\lambda_{i+1,j} = \prod_{k=1}^{i-1} \frac{j+k}{k}$. So $\lambda_{nj} = \prod_{k=1}^{n-2} \frac{j+k}{k}$. Using the binomial properties, we have:

$$\begin{aligned}
\lambda_{nj} &= \prod_{k=1}^{n-2} \frac{k+j}{k} \\
&= \frac{(1+j)(2+j)\cdots(n+j-2)}{(n-2)!} \\
&= \frac{(n+j-2)!}{j!(n-2)!} \\
\lambda_{nj} &= \binom{n+j-2}{j} \qquad (3.10)
\end{aligned}$$

c)Let us now deduce our result. By definition, we have:

$$\begin{aligned}
\overline{W}_n(r - \frac{j}{4} &= \sum_{k=n}^{n} W_n(r - \frac{j}{4}) \\
&= W_n(r - \frac{j}{4}).
\end{aligned}$$

And using the fact that $el_{\varepsilon,\varepsilon} \geq 1$ $(1 \leq \varepsilon \leq n)$, we get:

$$\begin{aligned}
W_n(r - \frac{j}{4}) &= card\Big(\{x_n \in \mathbb{Z} :\ (l_{nn}x_n)^2 \leq r - \frac{j}{4}\}\Big) \\
&= 2\Big\lfloor \frac{\sqrt{r - \frac{j}{4}}}{l_{nn}} \Big\rfloor + 1 \\
&\leq 2\lfloor \sqrt{re} \rfloor + 1 \quad \text{as } el_{nn} \geq 1.
\end{aligned}$$

Finally using (3.9) and (3.10), one may deduce that

$$\overline{W}_1(r) = (2\lfloor \sqrt{re} \rfloor + 1) \sum_{j=0}^{4\lfloor re \rfloor} \binom{n+j-2}{j}.$$

Using the binomial property $\sum_{k=0}^{n} \binom{k+n}{k} = \binom{n+m+1}{n}$, for $m \in \mathbb{N}$, we get:

$$\overline{W}_1(r) \leq (2\lfloor \sqrt{re} \rfloor + 1) \binom{4\lfloor re \rfloor + n - 1}{4\lfloor re \rfloor}.$$

This conclude the proof of our proposition.

From this proposition, by taking $r = M$, we deduce that $\overline{W}_1(M) \leq (2\lfloor \sqrt{Me} \rfloor + 1)\binom{4\lfloor Me \rfloor + n - 1}{4\lfloor Me \rfloor}$. Hence, the number of vectors x generated by the algorithm is at most

$$(2\lfloor \sqrt{Me} \rfloor + 1) \binom{4\lfloor Me \rfloor + n - 1}{4\lfloor Me \rfloor}$$

. Now, let us give an upper bound for the total number of operations used by our algorithm.

Theorem 32 *[4] Let $\mathcal{L}$ be a lattice with Gram matrix $F = (F_{ij})_{1\leq i,j\leq n}$.Let e^{-1} be the lower bound for the entries $l_{\varepsilon,\varepsilon}$ $(1 \leq \varepsilon \leq n)$. Let M be the maximal norm of the basis vectors of $\mathcal{L}$. Then the algorithm for computation of short vectors of $\mathcal{L}$ (using the Cholesky decomposition) uses at most*

$$\mathcal{O}\Big(n^2\Big((2\lfloor\sqrt{Me}\rfloor+1)\binom{4\lfloor Me\rfloor+n-1}{4\lfloor Me\rfloor}+1\Big)\Big)$$

arithmetic operations.

Proof. We would like to find an upper bound the number of arithmetic operations used by our algorithm for computing all $x = (x_1, \cdots, x_n) \in \mathbb{Z}^n$ satisfying $T(x) \leq M$; where $T(x) = \sum_{i=1}^{n}\Big(\sum_{j=i}^{n} l_{ij}x_j\Big)^2$ and $L = (l_{ij})_{1\leq i,j\leq n}$ is the Cholesky factor of F^* computed in step b) of the algorithm.
To find this number, let us study the complexity analysis of all the steps of our algorithm.

- In step a), we compute the LLL-reduced matrix F^* of F. From [3], the LLL-reduction algorithm computes F^* in time $\mathcal{O}(n^6(\log M)^3)$;

- In step b), we computes the Cholesky decomposition of F^* and find $F^* = LL^T$. From [3], the computational complexity of the Cholesky algorithm is $\mathcal{O}(n^3)$;

- In step c), we compute the bound of x_n using at most 6 operations; In step d), the computation of Y_k and U_k takes at most $\mathcal{O}(n^2)$ operations; Thus the transition from one vector to the next takes at most $\mathcal{O}(n^2)$. Recall that in our algorithm, each time we find a vector x that is a solution, we add x and $-x$ to the set of solutions. Hence, the execution of our algorithm from step d) to step f') produces at most $\frac{1}{2}\Big((2\lfloor\sqrt{Me}\rfloor+1)\binom{4\lfloor M\rfloor+n-1}{4\lfloor Me\rfloor}+1\Big)$; Therefore the execution of the algorithm from step c) to step f') uses at most $\mathcal{O}\Big(\frac{1}{2}n^2\Big((2\lfloor\sqrt{Me}\rfloor+1)\binom{4\lfloor Me\rfloor+n-1}{4\lfloor Me\rfloor}+1\Big)\Big)$ arithmetic operations.

Hence, we can conclude that our algorithm uses at most

$$\mathcal{O}\Big(n^2\Big((2\lfloor\sqrt{Me}\rfloor+1)\binom{4\lfloor Me\rfloor+n-1}{4\lfloor Me\rfloor}+1\Big)\Big)$$

arithmetic operations.

Theorem 33 *Let $\mathcal{L}$ be a lattice with Gram matrix $F = (F_{ij})_{1\leq i,j\leq n}$. Suppose that $l_{ii} \geq 1$ $(1 \leq i \leq n)$. Assume that n is large enough $(n \gg \Big\lfloor\sqrt{M}\Big\rfloor)$. Let M be the maximal norm of the basis vectors of $\mathcal{L}$. Then the algorithm for computation of short vectors of $\mathcal{L}$ (using the Cholesky*

decomposition) uses at most

$$\mathcal{O}\Big(\Big(1+\frac{n-1}{4\lfloor M\rfloor}\Big)^{4\lfloor Me\rfloor}\Big)$$

arithmetic operations.

PROOF. Notice that for n large enough ($n \gg \Big\lfloor\sqrt{M}\Big\rfloor$), Stirling's formula (namely for $k,\ m \in \mathbb{N}$, $\binom{m}{k} \sim \frac{m^k}{k!}$) yields that $\overline{W}_1(M)$ increases at most like $\mathcal{O}\Big(\Big(1+\frac{n-1}{4\lfloor M\rfloor}\Big)^{4\lfloor M\rfloor}\Big)$ (i.e $\overline{W}_1(M) \sim \mathcal{O}\Big(1+\frac{n-1}{4\lfloor M\rfloor}\Big)^{4\lfloor M\rfloor}$). Indeed, for $k = 4\lfloor M\rfloor$, and $m = n-1$, one has $\binom{n}{k} = \frac{(k+m)^k}{k!} = \frac{k^k}{k!}(1+\frac{m}{k})^k$.

Corollary 34 *Let $\mathcal{L}$ be a lattice with Gram matrix $F = (F_{ij})_{1\leq i,j\leq n}$. Suppose that $l_{ii} \geq 1$ $(1 \leq i \leq n)$. Assume that n is fixed. Let M be the maximal norm of the basis vectors of $\mathcal{L}$ and suppose that M tends to infinity. Then the algorithm for computation of short vectors of $\mathcal{L}$ (using the Cholesky decomposition) uses at most*

$$\mathcal{O}\Big(M^{\frac{n}{2}}\Big)$$

arithmetic operations.

PROOF. We use the above theorems and the assumptions of this corollary to get the following:
If n is fixed then n is considered as a constant. So $n^2 = \mathcal{O}(1)$. Also $(2\lfloor\sqrt{M}\rfloor+1) = \mathcal{O}(\lfloor\sqrt{M}\rfloor) = \mathcal{O}(\sqrt{M})$. Moreover

$$\begin{aligned}\binom{4\lfloor M\rfloor+n-1}{4\lfloor M\rfloor} &= \binom{4\lfloor M\rfloor+n-1}{n-1}\\ &= \mathcal{O}\Big((4\lfloor M\rfloor+n-1)^{n-1}\Big)\\ &= \mathcal{O}(M^{\frac{n-1}{2}})\end{aligned}$$

as M tends to infinity.
Hence we get the result.

Remark 35 *One may proceed similarly for to study the complexity analysis for computing short vectors of a lattice using the LDL^T-decomposition.*

3.3 Runtime of algorithms

The purpose of this section is to give the computation time of the two algorithms described above on an example. We consider the root lattice $\mathbb{A}_n$ for $n = 2, 3, 4, 5, 6, 7, 8$ with Gram matrix $F_{\mathbb{A}_n}$ defined with 2 on the diagonal entries, -1 under and over the diagonal entries and the 0 elsewhere. It is given by:

$$F_{\mathbb{A}_n} = \begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & \cdot & \cdot & \cdots & \cdot \\ 0 & \cdot & \cdot & \cdot & \vdots \\ \vdots & \cdot & \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{pmatrix}$$

we execute the cputime() Sage function and our Sage code for computing short vectors using the Cholesky on a personal computer core I3. We get the following results.

n	time(s)	Card(S)
2	0	7
3	0.063	13
4	0.156	21
5	0.765	31
6	3.728	43
7	21.06	57
8	135.986	73

Table result of the runtime of our algorithm

From these results, we observed that we obtain the computation time of this algorithm increases with the dimension n of the Gram matrix.

Chapter 4

Automorphisms of lattices

We recall from chapter 2 that an **automorphism** on a lattice $(\mathcal{L}, \Phi)$ is a $\mathbb{Z}$-linear bijection $\theta : \mathcal{L} \to \mathcal{L}$ satisfying:

$$\Phi(x, y) = \Phi(\theta(x), \theta(y)) \quad \forall x, y \in \mathcal{L}.$$

Let $B = (b_1, \cdots, b_n)$ be the basis matrix of the lattice $(\mathcal{L}, \Phi)$. To specify an automorphism θ on a lattice $\mathcal{L}$, it suffices to determine images of the vector basis. More precisely, we have to determine a basis $(v_1, \cdots, v_n)$ from the vectors of $\mathcal{L}$ satisfying:

$$\Phi(v_i, v_j) = \Phi(b_i, b_j) \quad \forall 1 \leq i, j \leq n;$$

We shall call such a list of vectors, good image of the basis vectors. Having found this good image, we can find the image of any lattice element under the automorphism θ (as each lattice element is a linear combination of the basis vectors). Indeed, if we consider Bx and By two lattice elements with $x = (x_1, \cdots, x_n)$, $y = (y_1, \cdots, y_n)$, and we suppose that $(v_1, \cdots, v_n)$ is a good image of the basis vectors then as θ is a $\mathbb{Z}$-linear map, we have:
$\theta(x) = \theta(\sum_{i=1}^{n} x_i b_i) = \sum_{i=1}^{n} x_i \theta(b_i) = \sum_{i=1}^{n} x_i v_i$; similarly, $\theta(y) = \sum_{j=1}^{n} y_j v_j$;

$$\begin{aligned}
\Phi(\theta(x), \theta(y)) &= (\sum_{i=1}^{n} x_i v_i).(\sum_{j=1}^{n} y_j v_j) \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} x_i y_i \Phi(v_i, v_j) \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} x_i y_i \Phi(b_i, b_j) \text{ as } \Phi(v_i, v_j) = \Phi(b_i, b_j\ ) \\
&= \Phi(x, y)
\end{aligned}$$

Hence, our aim is to build up the set of all bases $(v_1, \cdots, v_n)$ that are good images of the basis vectors $(b_1, \cdots, b_n)$. So we can identify the group $Aut(\mathcal{L})$ of automorphisms on $\mathcal{L}$ by the set:
$\{(v_1, \cdots, v_n) : (v_1, \cdots, v_n)$ is good image of the basis vectors$(b_1, \cdots, b_n)\}$. These good images will be determined by using the candidate images of the basis vectors. Before going through these candidate images, let us precise that if we proceed as explained above we really get an automorphism.

Lemma 36 *We consider the same notation as above. Let S be the set of short vectors of $\mathcal{L}$.*

If we find vectors $(v_1, \cdots, v_n)$ in S such that $\Phi(v_i, v_j) = \Phi(b_i, b_j)$ for $1 \leq i, j \leq n$, then we have defined an automorphism.

PROOF. We consider the lattice $\mathcal{L}$ as above. Let F be the Gram matrix of the lattice $\mathcal{L}$. We consider $(v_1, \cdots, v_n)$ in S. We assume that $\Phi(v_i, v_j) = \Phi(b_i, b_j)$ for $1 \leq i, j \leq n$. We would like to show that there exists an endomorphism θ on $\mathcal{L}$ such that $\theta(\mathcal{L}) = \mathcal{L}$.
a)Since $(v_1, \cdots, v_n)$ belongs S, then $v_i \in \mathcal{L}\ \forall\ 1 \leq i \leq n$. As $\Phi(v_i, v_j) = \Phi(b_i, b_j)$ for $1 \leq i, j \leq n$, therefore there exists an endomorphism θ on $\mathcal{L}$ such that θ sends the basis vectors $(b_1, \cdots, b_n)$ to the list of vectors $(v_1, \cdots, v_n)$. This implies that $\theta(\mathcal{L}) \subseteq \mathcal{L}$.

b) Let us show that $\mathcal{L} \subseteq \theta(\mathcal{L})$.
Let $M = (M_{ij})$ be the matrix associated with θ in B, then $\theta(b_j) = \sum_{k=1}^{n} M_{kj} b_k$. We have:

$$\begin{aligned}
\Phi(\theta(b_i), \theta(b_j)) = \Phi(b_i, b_j) &\Rightarrow \sum_{l=1}^{n}\sum_{k=1}^{n} M_{li} M_{kj} b_k \Phi(b_i, b_j) = \Phi(b_i, b_j) \\
&\Rightarrow \sum_{l=1}^{n}\sum_{k=1}^{n} M_{li} M_{kj} b_k F_{ij} = F_{ij} \\
&\Rightarrow M^T F M \ = \ F \\
&\Rightarrow det(M^T F M) = det(F) \\
&\Rightarrow det(M^T) \times det(F) \times det(M) = det(F) \\
&\Rightarrow (det(M))^2 = 1 \text{ as } det(M^T) = det(M) \\
&\Rightarrow det(M) = \pm 1
\end{aligned}$$

Hence $det(M) \neq 0$; so M is invertible.
Let M^{-1} be the inverse matrix of M. By definition, $M^{-1} = \frac{1}{det(M)} \times Com(M)$, where $Com(M)$ is the co-matrix of M. As $M \in \mathbb{M}_n(\mathbb{Z})$, then $Com(M) \in \mathbb{M}_n(\mathbb{Z})$.
Since $det(M) = \pm 1$, from the formula of M^{-1}, we can express M^{-1} as:
$M^{-1} = \pm N$ with $N \in \mathbb{M}_n(\mathbb{Z})$.
Let ψ be the morphism associated to the matrix N. We have:

$$\begin{aligned}
N \in \mathbb{M}_n(\mathbb{Z}) &\Rightarrow \psi(\mathcal{L}) \subseteq \mathcal{L} \\
&\Rightarrow \theta \circ \psi(\mathcal{L}) \subseteq \theta(\mathcal{L}) \\
&\text{As } \psi \text{ is an inverse of } \theta \text{ then } \mathcal{L} \subseteq \theta \circ \psi(\mathcal{L}) \\
&\Rightarrow \mathcal{L} \subseteq \theta(\mathcal{L}).
\end{aligned}$$

Hence $\theta(\mathcal{L}) = \mathcal{L}$.

Now, we are sure that if we find vectors $(v_1, \cdots, v_n)$ in S such that $\Phi(v_i, v_j) = \Phi(b_i, b_j)$ for $1 \leq i, j \leq n$, then we have defined an automorphism.
Let us focus on the candidate images of the basis vectors.

4.1 Candidate Images

A candidate image of a basis vector is a vector of $\mathcal{L}$ that is susceptible to be its image under the automorphism involved. At the step k, the set C_{ki}(for each $1 \leq i, k \leq n$) is the set of all candidate images of a basis vector b_i. This set is defined using the set of short vectors S defined in the previous chapter. To construct the set C_{ki}, we first collect all vectors from a certain subset SN_i of the set S. For $1 \leq i \leq n$, SN_i is the set of all vectors from S with same norm as the basis vector b_i. It is defined by:
$SN_i = \{v \in S : \Phi(v, v) = \Phi(b_i, b_i)\}$.

Definition 37 *We say that a vector $v \in SN_i$(for some $1 \leq i \leq n$ fixed) satisfies the **inner product conditions** with respect to a list of vectors $(v_1, \cdots, v_k)$(with $k \leq n$), if $\Phi(v, v_j) = \Phi(b_i, b_j)$ for $j = 1, \cdots, k$ and $j \neq i$.*

Definition 38 *A **candidate vector** at the step k of a basis vector b_i(for $1 \leq i \leq n$) is a lattice element c with same norm as b_i, satisfying the inner product conditions with respect to the list of good images $(v_1, \cdots, v_k)$ of basis vectors $(b_1, \cdots, b_k)$(with $k \leq n$). More precisely, it is a vector $c \in SN_i$ such that $\Phi(c, v_j) = \Phi(b_i, b_j)$ for $j = 1, \cdots, k$ and $j \neq i$.*

Before determining the set C_{ik} (for some $1 \leq k, i \leq n$), Let us first write an algorithm to find the set SN_i.

4.1.1 Algorithm to find SN_i

What follows is an algorithm to compute the list SN of the sets SN_i of lattice elements with same norm as the basis vector b_i. We consider the lattice $(\mathcal{L}, \Phi)$ as above.

Pseudocode

Input: F (the Gram matrix of the lattice) and S (the list of coordinates of short vectors of the lattice)
Output: SN (the list SN of the sets SN_i)

4.1.2 Algorithm to find C_{ki}

Here is an algorithm to find at a step k the list of candidates images C_{ki} of lattice elements with same norm as the basis vector b_i. We consider the lattice $(\mathcal{L}, \Phi)$ as above. We assume that the good image $(v_1, \cdots, v_{k-1})$ of the list of k-vectors $(b_1, \cdots, b_{k-1})$ has been found already. So we set their list of candidate images to be the empty list, and we determine the remaining candidate using the above description of C_{ki}.

Algorithm 2 the list SN of the sets SN_i

```
1: procedure same − norm(F, S)
2:     D ← Diagonal(F)
3:     n ← number of columns of F
4:     for u in S do
5:         if Φ(u, u) ∈ D then
6:             K ← list − indices(Φ(u, u) in D)
7:             for i ∈ K do
8:                 append(u, SN_i)
9:     return SN
```

Pseudocode

input: F (the Gram matrix of the lattice), SN_i (as defined above), $[v_1, \cdots, v_{k)1}]$ a $(k-1)$-partial automorphism and the index k, i.
output: C_{ki} (the list of candidates images)

Algorithm 3 the list of candidates images C_{ki}

```
1: procedure cand − vect(F, SN, [v_1, ··· , v_{k−1}], k, i)
2:     n ← number of columns of F
3:     if i < k then
4:         return [ ]
5:     else
6:         for u ∈ SN_i do
7:             if Φ(u, v_j) = F_{kj} (∀j = 1, ··· , k − 1) then
8:                 append(u, C_i)
9:     return C_i
```

4.2 Background on automorphisms search

Here, we present some background informations for the automorphisms search of lattices. Recall that we would like to describe an algorithm (based on [1]), to compute the group of automorphisms of a given lattice. In general not all list of vectors $(v_1, \cdots, v_n)$ (obtained from the set of candidate images) are good images of the basis vectors. Also, one may find a list of vectors $(v_1, \cdots, v_n)$ such that there exists a rank k such that the list of k vectors $(v_1, \cdots, v_k)$ are good images of the basis vectors $(b_1, \cdots, b_k)$, but the vector v_{k+1} is not a good image of b_{k+1}. Such a list $(v_1, \cdots, v_k)$ of vector is called a $k-$partial automorphism.

Definition 39 *A $k-$**partial automorphism** is a partial map $(v_1, \cdots, v_k)$ that sends b_i to v_i for $1 \leq i \leq k$, satisfying $\Phi(v_i, v_j) = F_{ij}$ for all $1 \leq i, j \leq k$. When $k = n$, it is called an automorphism.*

Example 40 *The trivial k-partial automorphism is the list of first k-basis vector $(b_1, \cdots, b_k)$ for any $k \leq n$. It comes from the identity automorphism, that sends b_i to b_i for $1 \leq i \leq k$.*

In the case described above, we say that this k-partial automorphism cannot be extended into a $(k+1)$-partial automorphism. Also, not every k-partial automorphism can be extended into a $(k+1)$-partial automorphism. Hence, to rule out as soon as possible a k-partial automorphism that will never be an automorphism, one may test whether the number of extensions is preserved.

Definition 41 *The number of extensions of a k-partial automorphism $(v_1, \cdots, v_k)$ is the number of vector $v \in S$ satisfying:*

$$\begin{cases} \Phi(v, v) = \Phi(b_{k+1}, b_{k+1}) \\ \Phi(v, v_j) = \Phi(b_{k+1}, b_j), \ \textit{for } j = 1, \cdots, k. \end{cases}$$

This number is denoted by $nbExt$. It is defined by:
$nbExt(v_1, \cdots, v_k) = |\{v \in SN_{k+1} : \Phi(v, v_j) = \Phi(b_{k+1}, b_j), \text{ for } j = 1, \cdots, k\}|$.

If $(v_1, \cdots, v_k)$ is a k-partial automorphism, then we set this number to be zero since all vectors in the k-partial automorphism are already good images of the basis vector b_1 up to b_k . Therefore, for any $1 \leq i, k \leq n$ and any k-partial automorphism $(v_1, \cdots, v_k)$, we may consider the following:

$$nbExt([v_1, \cdots, v_k], k, i) = \begin{cases} 0 & \text{if } i < k \\ \left|\{v \in SN_i : \Phi(v, v_j) = \Phi(b_i, b_j), \text{ for } j = 1, \cdots, k-1\}\right| & \text{otherwise.} \end{cases}$$

Remark from above that this number of extensions is exactly the number of candidate images C_{ki} for each fixed i and k. To compute this number, we will use the following algorithm.

4.2.1 Algorithm to find the number of extensions

Here is an algorithm to find the list of candidates images C_{ki} of lattice element with same norm as the basis vector b_i. We consider the lattice $(\mathcal{L}, \Phi)$ as above.

Pseudocode

input: F (the Gram matrix of the lattice), SN_i (defined as above), kpartial (the k-partial automorphism $[v_1, \cdots, v_k]$) and the index k, i.
output: $nExt$(the number of extensions)

Algorithm 4 the list of candidates images C_{ki}

```
1: procedure nbExt(F, SN, kpartial, k, i)
2:     n ← number of columns of F
3:     nExt ← length(cand − vect(F, SN, kpartial, k, i))
4:     return nExt
```

Recall that, our goal is to construct the set

$$\{(v_1, \cdots, v_n) : (v_1, \cdots, v_n) \text{ is a good image of } (b_1, \cdots, b_n)\}.$$

To determine each such list of vectors $(v_1, \cdots, v_n)$, the idea of the naive algorithm is to first choose the first k- good images $(v_1, \cdots, v_k)$ (for $k \leq n$) using the set of candidate images; then check if $(v_1, \cdots, v_k)$ is a k-partial automorphism; next, we choose a vector v in C_{k+1}, check if $(v_1, \cdots, v_k, v)$ is a $(k+1)$-partial automorphism; if so, then v is a good image for b_{k+1};we keep the new list of $(k+1)$-partial automorphism and we continue this way up to $k = n$ and get one automorphism if all the above checks are satisfied. Each time that one of these checks is not satisfied, we take another vector in the set C_{k+1} until this set is empty. To find all automorphisms, we proceed as above for any possible partial list of good images $(v_1, \cdots, v_k)$ (for $1 \leq k \leq n$).
However [1] claims that the number of extensions must be preserved under isometries. This is stated in the following proposition.

Proposition 42 *Let $(\mathcal{L}_1, \Phi_1)$ and $(\mathcal{L}_2, \Phi_2)$ two lattices with basis B_1 and B_2 respectively. Let g be an isometry from $\mathcal{L}_1$ to $\mathcal{L}_2$. Let k be an integer. Suppose that g sends $(b_1, \cdots, b_k)$ to $(v_1, \cdots, v_k)$.*
Then $NbExt[(b_1, \cdots, b_k) \mapsto (b_1, \cdots, b_k)] = Nbext[(b_1, \cdots, b_k) \mapsto (v_1, \cdots, v_k)]$.

To simplify the notation, we set:
$NbExt((b_1, \cdots, b_k)) := NbExt[(b_1, \cdots, b_k) \mapsto (b_1, \cdots, b_k)]$,
and $NbExt((v_1, \cdots, v_k)) := Nbext[(b_1, \cdots, b_k) \mapsto (v_1, \cdots, v_k)]$.

Proof. Consider the sets:
$E = \{u \in \mathcal{L}_1 : \ \Phi_1(u, u) = \Phi_1(b_{k+1}, b_{k+1})$ *and* $\Phi_1(u, b_j) = \Phi_1(b_{k+1}, b_j)$ for $j = 1, \cdots, k\}$,
$D = \{v \in \mathcal{L}_2 : \ \Phi_2(v, v) = \Phi_1(b_{k+1}, b_{k+1})$ *and* $\Phi_2(u, v_j) = \Phi_1(b_{k+1}, b_j)$ for $j = 1, \cdots, k\}$,
By definition $NbExt((b_1, \cdots, b_k)) = card(E)$ and $NbExt((v_1, \cdots, v_k)) = card(D)$, where $card(E)$ and $card(D)$ denote the cardinal of E and D respectively.
We want to show that $card(E) = card(D)$. In order to do this, we prove that the sets E and D are in bijection.
We may consider the map ϕ as follows:

$$\begin{aligned} \phi : E \ &\to D \\ u &\mapsto g(u) \end{aligned}$$

- The map ϕ is well defined (as g is a well defined isometry: indeed $\forall\, u \in E$, $\Phi_2(g(u), g(u)) = \Phi_1(u, u) = \Phi_1(b_{k+1}, b_{k+1})$; so $\phi(u) \in D$);
- Moreover, the map $\psi : D \ \to E$ that sends any v in D to $g^{-1}(v)$ is well defined and is an inverse of ϕ.

Therefore, ϕ is a bijection and $card(E) = card(D)$.

Notice that an automorphism on a lattice $\mathcal{L}$ is an isometry from $\mathcal{L}$ to $\mathcal{L}$. Hence, the above proposition remains true for automorphisms.

The statement of this proposition is helpful to detect earlier a k-partial automorphism that will never be an automorphism. To take this into account in our algorithm, we consider the naive algorithm described above; in that algorithm, before checking if the list $(v_1, \cdots, v_k, v)$ is a $(k+1)$-partial automorphism, we first check if the number of extensions of $(v_1, \cdots, v_k, v)$ corresponds to the number of extensions of $(b_1, \cdots, b_k, b_{k+1})$. The number of extensions of the trivial k-partial automorphism $(b_1, \cdots, b_k, b_{k+1})$ will be stored in a matrix called the fingerprint.

4.2.2 Fingerprint

The fingerprint stores at each entry(of index k, i) the number of vectors from SN_i satisfying the inner product conditions with respect to the first $(k-1)-$basis vectors $(b_1, \cdots, b_{k-1})$. As these first $(k-1)$-basis vectors form a $(k-1)-$partial automorphism, the good images of the basis vectors $(b_1, \cdots, b_{k-1})$ are already known. As we are looking for the good image of b_k, we may suppose that each entry (of index k, i) of the fingerprint equals zero for $1 \leq i < k \leq n$. Hence, we may defined the fingerprint as follows.

Definition 43 *The **fingerprint** of given lattice is an upper-triangular matrix denoted by $f = (f_{ki})_{1 \leq k, i \leq n}$ and defined by:*

$$f_{ki} = \begin{cases} 0 & \text{if } i < k \\ |\{v \in SN_i : \Phi(v, b_j) = \Phi(b_i, b_j) \text{ for } j = 1, \ldots, k-1\}| & \text{if } i \geq k. \end{cases}$$

Each entry f_{ki} is exactly the number of candidates vectors that can be good images of the basis vector b_i.

From the definition of the entries of the fingerprint, a different ordering of the basis vectors will give a different fingerprint. The method to find automorphisms that we are going to describe is a backtrack search. For a fast backtrack search, it is better to reorder the matrix of the basis vectors with respect to the number of candidate vectors of each basis vector. More precisely, we will order B starting form the basis vector with the lowest possible candidate images up to the one with the biggest possible candidate image. Indeed, if there is a vector that only have a few possible images under any automorphism(or isometry), this will help us to first find the image of that vector. To optimize our ordering, we may reorder our matrix basis while computing our fingerprint. After computing each k-th row of the fingerprint f, we test if the entry f_{kk} is the minimal non-zero entry in the row. If so, we do nothing. Otherwise, we swap the k-th column of the fingerprint with any column containing the minimal non-zero entry in the row involved. While doing this, we will swap the corresponding basis vectors, compute the new Gram matrix; and we will also swap the corresponding sub-lists of the list of same norm SN (computed above) as well as the entries in any coordinates vector of SN.

Assume that the first row of our fingerprint is $[6, 7, 1]$. This tells us that there are six small vectors whose norm matches b_1, seven small vectors whose norm matches b_2 and only one small vectors whose norm matches b_3. From this, we realise that b_3 has less possible candidate image than b_1 and b_2. Hence, if we start our search with b_3 instead of b_1, then we will have less possible partial maps to verify.

Here, we have an outline of how to compute the fingerprint of a given lattice with basis B, Gram matrix F and list of same norm SN.

Pseudocode

Input: F (the Gram matrix of the lattice) and SN (as defined above).
Output: f (the fingerprint)

Algorithm 5 The fingerprint f

```
1: procedure fingerprint(F, SN)
2:     n ← number of columns of F
3:     for k = 1, ··· , n do
4:         for i = 1, ··· , n do
5:             if i < k then
6:                 f_ki ← 0
7:             else
8:                 f_ki ← nbExt(F, SN, kpartial, k, i)
9:     return f
```

Here are some examples of fingerprints of lattices matrix basis, where the ordering explained above is take into account. these fingerprints are computed using our Sage code.

Example 44 *Consider the lattice generated by the basis* $B = \begin{pmatrix} 2 & 0 & 0 \\ 1 & 2 & 1 \\ 1 & 0 & 1 \end{pmatrix}$. *Its Gram matrix is* $F = \begin{pmatrix} 5 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 2 \end{pmatrix}$. *Using the Sage code of the algorithms explained above, we found it set of small vectors:*

$$\begin{aligned} S = &[(0,1,-2),(0,-1,2),(0,0,-1),(0,0,1),(1,0,-1),(-1,0,1),\\ &(-1,1,-1),(1,-1,1),(0,1,-1),(0,-1,1),(0,-1,0),(0,1,0),\\ &(1,-1,0),(-1,1,0),(-1,0,0),(1,0,0),(0,0,0]. \end{aligned}$$

This lattice has 17 *small vectors. We also compute:*
$SN_1 = [(1,0,-1),(-1,0,1),(-1,1,-1),(1,-1,1),(1,-1,0),(-1,1,0),(-1,0,0),(1,0,0)]$,
$SN_2 = [(0,1,-2),(0,-1,2),(0,-1,0),(0,1,0)]$
and $SN_3 = [(0,0,-1),(0,0,1),(0,1,-1),(0,-1,1)]$
. Hence $SN = [SN_1, SN_2, SN_3]$ *So we have eight elements with same norm as* b_1*, and four with same norm as* b_2 *and* b_3*. We know from the ordering of the basis vectors explained that after computing our fingerprint, we are suppose to have a basis where the first and the second vector are swapped. And this is the case, since after computing the fingerprint with our Sage code, we get:*

- *The fingerprint of this matrix is:* $f = \begin{pmatrix} 4 & 8 & 4 \\ 0 & 2 & 2 \\ 0 & 0 & 2 \end{pmatrix}$;
- *The matrix basis becomes:* $B = \begin{pmatrix} 0 & 2 & 0 \\ 2 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}$;

- *The list SN becomes* $SN = [SN_2, SN_1, SN_3]$. *Also the first and the second entries of each coordinates vectors in SN are swapped. For instance,* SN_3 *becomes* $SN_3 = [(0,0,-1),(0,0,1),(1,0,-1),(-1,0,1)]$.

Having computed the fingerprint, we can now test whether the number of extensions a k-partial automorphism into a $(k+1)$-partial automorphism is preserved. To test this, we consider a k-partial automorphism $(v_1, \cdots, v_k)$ and we check if the number of extensions of this k-partial automorphism is the same as the entry $f_{k+1,k+1}$ of the fingerprint f. More precisely, since:
$nbExt(v_1, \cdots, v_k) = |\{v \in SN_{k+1} : \Phi(v, v_j) = \Phi(b_{k+1}, b_j) \text{ for } j = 1, \ldots, k\}|$ and
$f_{k+1,k+1} = |\{v \in SN_{k+1} : \Phi(v, b_j) = \Phi(b_{k+1}, b_j) \text{ for } j = 1, \ldots, k\}|$,
then we have to check if $nbExt(v_1, \cdots, v_k) = f_{k+1,k+1}$. This is called the fingerprint test. In the automorphism search' algorithm, this will be done at each step k.
Before the description of the algorithm for finding automorphisms with the fingerprint, let us give the naive algorithm of this computation.

4.3 Naive algorithm for automorphisms of lattices

Recall that, our aim is to construct the set

$$\{(v_1, \cdots, v_n) \text{ from } \mathcal{L} : (v_1, \cdots, v_n) \text{ is a good image of } (b_1, \cdots, b_n)\}.$$

To determine such a list of vectors $(v_1, \cdots, v_n)$, the idea of the naive algorithm is to first choose a partial list of good images $(v_1, \cdots, v_k)$(for $1 \leq k \leq n$) using the set of candidate images(we will start with $k = 1$ and take v_1 in SN_1); then check if $(v_1, \cdots, v_k)$ is a k-partial automorphism; if it is the case, then we choose a v vector in C_{k+1}, check if $(v_1, \cdots, v_k, v)$ is a $(k+1)-$partial automorphism; if so, then v is a good image for b_{k+1}; we keep the new list of $(k+1)-$partial automorphism and we continue this way up to $k = n$ and get one automorphism if all the above checks are satisfied. Each time that one of these checks is not satisfied, we take another vector in the set C_{k+1} until this set is empty. To find all automorphisms, we proceed as above for every possible partial list of good images $(v_1, \cdots, v_k)$(for $1 \leq k \leq n$). Indeed, having found an automorphism $(v_1, \cdots, v_{n-1}, v)$, to find others automorphisms, we first keep the $(n-1)-$partial automorphism $(v_1, \cdots, v_{n-1})$; next, we test if this $(n-1)-$partial automorphism together with each candidate image v of b_n form an automorphism. We keep the automorphism $(v_1, \cdots, v_{n-1}, v)$ found. Once all candidate images of b_n have been tested, we go back to the index $k = n - l$(starting from $l = 1$ up to $l = n-1$). For each l, while the set C_{n-l} is non empty, we choose another candidate image of b_{n-l}, we increase the index $k = n - l$ to $k = n - l + 1$ and we proceed as above.
The algorithm will stop whenever the set of candidate images of the first basis vector b_1 is empty.

pseudocode

To find all automorphism, we need the procedure $is-i-$partial automorphism. This procedure checks whether the list of vectors $[v_1, .., v_{i-1}]$ preserves the inner product with the $[b_1, .., b_{i-1}]$; i.e $[v_1, .., v_{i-1}, v_i]$ is again a $i-$partial when we add a vector v_{i-1} to a $(i-1)-$partial $[v_1, .., v_{i-1}]$(the v_i are vectors from the lattice $\mathcal{L}$). This function returns True if so and False otherwise.

Input: F(the Gram matrix of our lattice $\mathcal{L}$) and the list of vectors $[v_1, \cdots, v_k]$ to test.
Output: True or False

Algorithm 6 is-i-partial-automorphism

```
1: procedure is − i − partial − auto(F, [v_1, .., v_{i−1}, v_i])
2:     n ← number of columns of F
3:     ans ← True
4:     for j = 1, ··· , i do
5:         if Φ(v, v_j) ≠ Φ(b_i, b_j) then
6:             ans ← False
7:             break
8:     return ans
```

The following procedure computes all automorphisms on our lattice $\mathcal{L}$ described as above(with basis vectors $(b_1, \cdots, b_n)$). To make our naive backtrack search, we consider the sets of n elements ind(list of horizontal index on the tree formed by the vectors images), k(the number of candidate images of the basis vector b_i), and V(current list of vectors images); we initialise the cells of these sets at zero. We also consider the current list S_{tp} of candidates of vectors images (start with $i = 1$ i.e vectors having same norm with b_1).

Input:F (the Gram matrix of the lattice $\mathcal{L}$) and SN (defined as above).
Output: *sol* (the list of all automorphisms of a lattice)

Algorithm 7 Naive algorithm for computing automorphisms of a lattice

```
1: procedure naive − auto(F, SN)
2:     n ← number of columns of F
3:     S_tp ← SN_1,  k_1 ← len(S_tp),  i ← 1
4:     while i ≤ n and ind_1 < k_1 do
5:         k_i ← len(S_tp),  find − vi ← False
6:         if i < n then
7:             for j = ind_i, ··· , k_i do
8:                 kpartial ← [V_1, ··· , V_i],  append(S_tp[j], kpartial)
9:                 L ← cand − vect(F, SN, kpartial, i + 1, i + 1)
10:                if is − i − partial(F, [v_1, .., v_{i−1}, v_i]) then
11:                    V_i ← S_tp[j],  S_tp ← L,  find − vi ← True
12:                    break
13:                else
14:                    if j < k_i − 1 then
15:                        ind_i ← ind_i + 1
16:            if find − vi then
17:                i ← i + 1
18:            else
19:                while ind_i = k_i − 1 and i > 1 do
20:                    ind_i ← 0,  i ← i − 1
21:                ind_i ← ind_i + 1
22:                S_tp ← NbExt(F, SN, [V_1, ··· , V_i], i, i)
23:        else
24:            for j = 1, ··· , k_i do
25:                if is − i − partial − auto(F, [v_1, .., v_{n−1}, S_tp[j]]) then
26:                    V_n ← S_tp[j]
27:                    append(V, sol)
28:            i ← n − 1
29:            while ind_i = k_i − 1 and i > 1 do
30:                ind_i ← 0,  i ← i − 1
31:            ind_i ← ind_i + 1
32:            if i = 1 and ind_i > k_i − 1 then
33:                return sol
34:                break the while loop
35:            if i = 1 then
36:                S_tp ← SN_1
37:            else
38:                S_tp ← NbExt(F, SN, [V_1, ··· , V_i], i, i)
39:    return sol
```

This algorithm will terminates as the group $Aut(\mathcal{L})$ of all automorphisms on $\mathcal{L}$) is finite. Indeed, if $\theta \in Aut(\mathcal{L})$ then $\theta(b_i) \in S$ for every $1 \leq i \leq n$ and the list $(\theta(b_1), \cdots, \theta(b_n))$ determines the automorphism θ; This implies that $|Aut(\mathcal{L})| \leq |S|^n$ which is finite since the set of short vectors S (shown in chapter 3) is finite.

Now, let us describe the algorithm that uses the fingerprint.

4.4 Computing Automorphisms of lattices with the fingerprint

To determine all automorphisms of a given lattice(using our algorithm), the idea is to make a backtrack search around the set of candidate images of each basis vector and use the fingerprint test to rule out as soon as possible k-partial automorphisms that cannot be extended.
To explain how the algorithm to compute our fingerprint work, let us assume that the basis B of our lattice have 5 vectors. Say we are looking for an automorphism θ. We would like to determine the good image of each basis vector in $B = (b_1, b_2, b_3, b_4, b_5)$ under θ. Suppose the good image of b_1 and b_2 are known say : v_1 and v_2(this is always possible as we may take the identity partial map namely $v_1 = b_1$ and $v_2 = b_2$). So θ sends b_1 to v_1 and b_2 to v_2. We suppose also that our fingerprint is computed using the ordering explained. In order to determine the good images of v_3, v_4 and v_5 through θ, we proceed as follows.

1) We check if (v_1, v_2) is a 2-partial auto(True in this case asv_1 and v_2 are supposed to be good images of b_1 and b_2);

2) If so, we compute the set C_3 of all candidate images of b_3;

3) We test for each vector v in C_3, whether $nbExt(v_1, v_2, v) = f_{3,3}$;

4) If 3) is satisfied, we set $v_3 = v$, and we go to 2).

Now we search for the image of b_4 and b_5 by following the above steps 1) up to 4) (but replacing the index 3 of the vector involved by 4 and 5 respectively). Proceeding this way, we will get one automorphism θ.

In general to find all automorphism of a given lattice, at each step k, we do the following.

Description of the algorithm

1) We start with a list of k-potential image $(v_1, \cdots, v_k)$ of the first $k-$basis vectors $(b_1, \cdots, b_k)$;

2) We check if $(v_1, \cdots, v_k)$ is a k-partial automorphism;

3) If so, we compute the set C_{k+1} of all candidate images of b_{k+1};

4) We take a vector v in C_{k+1}, and test if $nbExt(v_1, \cdots, v_k, v) = f_{k+1,k+1}$;

5) If so, we set $v_{k+1} = v$, $k = k + 1$ and we go to 2); otherwise, we go to 4 and take another vector in C_{k+1} until C_{k+1} is empty;

6) When we reach the last basis vector (i.e at $k = n$), we test for each vector in C_n if $(v_1, \cdots, v_{n-1}, v)$ is an $n-$ partial map; If so we set $v_n = v$. Hence we found our automorphism; Otherwise, we do the same for another vector in C_n until C_n is empty;

Pseudocode

To find all automorphisms, we need the following procedure (called '$is-$partial') to test if the $(n-1)$ vectors of a list L together with a vector V_n form an automorphism.
Input: L (a list of a $(n-1)$-partial automorphism: $[V_1, ..., V_{n-1}]$), a vector V_n and the Gram matrix F of $\mathcal{L}$.
Output: True(if it is an automorphism) or False(otherwise)

Algorithm 8 is-partial

```
1: procedure is − partial(L, V_n, F)
2:     n ← number of columns of F
3:     ans ← True
4:     for j = 1, ··· , length(L) do
5:         if Φ(V_n, L[j]) ≠ F[n − 1, j] then
6:             ans ← False
7:             break
8:     return ans
```

The following procedure computes the list of all automorphisms of a lattice with basis vector $B = (b_1, \cdots, b_n)$, and set of small vector S. To make our backtrack search, we consider the sets of n elements ind(list of horizontal index on the tree formed by the vectors images), k(the number of candidate images of the basis vector b_i), and V(current list of vectors images); we initialise the cells of these sets at zero. We also consider the current list S_{tp} of candidates of vectors images(start with $i = 1$ i.e vectors having same norm with b_1).
Input: F (the Gram matrix of the lattice), SN (defined as above) and f (the computed fingerprint).
Output: *sol* (the list of all automorphisms of a lattice)

Algorithm 9 The list of all automorphisms of a lattice computed using the fingerprint

```
1: procedure auto(F, SN, f)
2:     n ← number of columns of F
3:     S_tp ← SN_1,  k_1 ← len(S_tp),  i ← 1
4:     while i ≤ n and ind_1 < k_1 do
5:         k_i ← len(S_tp),  find − vi ← False
6:         if i < n then
7:             for j = ind_i, ··· , k_i do
8:                 kpartial ← [V_1, ··· , V_i],  append(S_tp[j], kpartial)
9:                 L ← nbExt(F, SN, i + 1, i + 1, kpartial)
10:                if L = f_{i+1,i+1} then
11:                    V_i ← S_tp[j],  S_tp ← L,  find − vi ← True
12:                    break
13:                else
14:                    if j < k_i − 1 then
15:                        ind_i ← ind_i + 1
16:            if find − vi = True then
17:                i ← i + 1
18:            else
19:                while ind_i = k_i − 1 and i > 1 do
20:                    ind_i ← 0,  i ← i − 1
21:                ind_i ← ind_i + 1
22:                S_tp ← nbExt(F, SN, i, i, [V_1, ··· , V_i])
23:        else
24:            for j = 1, ··· , k_i do
25:                if is − partial([V_1, ··· , V_{n−1}], S_tp[j], F) then
26:                    V_n ← S_tp[j]
27:                    append(V, sol)
28:            i ← n − 1
29:            while ind_i = k_i − 1 and i > 1 do
30:                ind_i ← 0,  i ← i − 1
31:            ind_i ← ind_i + 1
32:            if i = 1 and ind_i > k_i − 1 then
33:                return sol
34:                break the while loop
35:            if i = 1 then
36:                S_tp ← SN_1
37:            else
38:                S_tp ← nbExt(F, SN, i, i, [V_1, ··· , V_i])
39:    return sol
```

Remark 45 *This algorithm will terminate as the set $Aut(\mathcal{L})$ is finite. Indeed, if $\theta \in Aut(\mathcal{L})$ then $\theta(b_i) \in S$ for every $1 \leq i \leq n$ and the list $(\theta(b_1), \cdots, \theta(b_n))$ determines θ; hence $|Aut(\mathcal{L})| \leq |S|^n$ which is finite since the set of short vectors S is finite (proof in chapter 3).*

4.5 Computation time measurement of algorithms

The purpose of this section is to compare the computation time of the two algorithms used to compute the automorphisms of some lattices: the naive algorithm and the algorithm with the fingerprint. In this following table, these algorithms are respectively called auto-naif and auto-fingerprint. We consider the identity lattice (whose Gram matrix is the identity matrix Id_n) and the root lattice $\mathbb{A}_n$ for $n = 2, 3, 4, 5, 6, 7, 8$ with Gram matrix $F_{\mathbb{A}_n}$ defined with 2 on the diagonal entries, -1 under and over the diagonal entries and the 0 elsewhere. It is given by:

$$F_{\mathbb{A}_n} = \begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & \cdot & \cdot & \cdots & \cdot \\ 0 & \cdot & \cdot & \cdot & \vdots \\ \vdots & \cdot & \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{pmatrix},$$

We execute the cputime() Sage function and our Sage code for computing automorphisms using these algorithms. In order to be sure of our results, we also use the Sage function to compute automorphisms in order to check if we get the same number of automorphisms. We get the following results.

1) Results obtained by using the identity lattice Id_n:

n	time(s)	My $Card(Aut(Id_n))$	Sage $Card(Aut(Id_n))$
2	0	8	8
3	0.016	48	48
4	0.109	384	384
5	1.888	3840	3840
6	32.682	46080	46080
7	716.341	645120	645120

Table result of auto-naif

n	time(s)	My $Card(Aut(Id_n))$	Sage $Card(Aut(Id_n))$
2	0	8	8
3	0.016	48	48
4	0.125	384	384
5	1.935	3840	3840
6	31.496	46080	46080
7	663.206	645120	645120

Table result of auto-fingerprint

2) Results obtained by using the root lattice $\mathbb{A}_n$:

n	time(s)	My $Card(Aut(\mathbb{A}_n))$	Sage $Card(Aut(\mathbb{A}_n))$
2	0	12	12
3	0.016	48	48
4	0.234	240	240
5	2.792	1440	1440
6	35.99	10080	10080
7	477.769	80640	80640

Table result of auto-naif

n	time(s)	My $Card(Aut(\mathbb{A}_n))$	Sage $Card(Aut(\mathbb{A}_n))$
2	0	12	12
3	0.031	48	48
4	0.234	240	240
5	2.683	1440	1440
6	33.026	10080	10080
7	464.742	80640	80640

Table result of auto-fingerprint

From these tables, we observed that the number of automorphisms computed with each of my functions is the same. This number is the same as the number of automorphisms computed with the Sage function. Also, the auto-fingerprint function is faster than the auto-naif function starting from $n = 6$. For $n \leq 5$, the computation time is almost the same for the two functions.

Chapter 5

Isometries of lattices

An **isometry** is a $\mathbb{Z}$-linear bijection g between two lattices $(\mathcal{L}, \Phi)$ and $(\mathcal{D}, \Psi)$ that preserves the bilinear forms Φ and Ψ. More precisely ,

$$\Phi(x, y) = \Psi(g(x), g(y)) \quad \forall x, y \text{ coordinate vectors in } \mathcal{L}.$$

In this work, we will consider two lattices $(\mathcal{L}, \Phi)$ and $(\mathcal{D}, \Psi))$ with basis $B = (b_1, \cdots, b_n)$ and $D = (d_1, \cdots, d_n)$ respectively.
We also consider the set $Iso(\mathcal{L}, \mathcal{D}) = \{g : (\mathcal{L}, \Phi) \to (\mathcal{D}, \Psi) \ : g \textit{ is an isometry}\}$ of all isometry form $\mathcal{L}$ to $\mathcal{D}$. What follows is a description of a method of computation of the set of all isometries between two arbitrary lattices. The general idea of this algorithm is based on the following proposition.

Proposition 46 *Let $h \in Iso(\mathcal{L}, \mathcal{D})$, and consider the set $\mathcal{H} = \{h\gamma : \gamma \in Aut(\mathcal{L})\}$.*
Then $Iso(\mathcal{L}, \mathcal{D}) = \mathcal{H}$.

PROOF. $Iso(\mathcal{L}, \mathcal{D}) \supseteq \mathcal{H}$: Let $h\gamma \in \mathcal{H}$ then:

a) $h\gamma$ is a bijection as h and γ are bijections.

b) Moreover, since by definition h and γ preserve their corresponding bilinear forms, therefore $\forall Bx, By \in \mathcal{L}$, $\Psi(h\gamma(x), h\gamma(y)) = \Phi(\gamma(x), \gamma(y)) = \Phi(x, y)$.

This implies that $h\gamma \in Iso(\mathcal{L}, \mathcal{D})$;
$Iso(\mathcal{L}, \mathcal{D}) \subseteq \mathcal{H}$: Let $g \in Iso(\mathcal{L}, \mathcal{D})$. Let us find $\gamma \in Aut(L)$ such that $g = h\gamma$
Set $\gamma = h^{-1}g$;

i) $\gamma : (\mathcal{L}, \Phi) \to (\mathcal{L}, \Phi)$ is a bijection since g and h^{-1} are both bijection by definition.

ii) again by definition of g and h^{-1}, we have:
$\forall Bx, By \in \mathbb{L}$, $\Phi(\gamma(x), \gamma(y)) = \Phi(h^{-1}g(x), h^{-1}g(y)) = \Psi(g(x), g(y)) = \Phi(x, y)$.

i) and ii) imply that $\gamma \in Aut(\mathcal{L})$.
Moreover, $\gamma = h^{-1}g \Rightarrow g = h\gamma \in \mathcal{H}$.
So $g \in \mathcal{H}$.
Hence $Iso(\mathcal{L}, \mathcal{D}) = \mathcal{H}$.

Remark 47 *From this proposition, we realise that to determine the set of all isometries between two arbitrary lattices, it suffices to first determine only one isometry $h(\in Iso(\mathcal{L}, \mathcal{D}))$ between the two lattices, then find all the automorphisms of the first lattice $\mathcal{L}$, and deduce $Iso(\mathcal{L}, \mathcal{D})$ by taking the composition of h with each of the automorphism found of $\mathcal{L}$.*

To determine all automorphisms between 2 lattices, we will use the algorithm presented in chapter 4. We start by describing the naive algorithm to compute either one or all isometries, then we present an algorithm that is faster than the naive one. Finally, we use them to describe the algorithm presented in the above remark.
In what follows, we consider the following notations:

- A lattice $(\mathcal{L}_1, \Phi_1)$ with matrix basis $B_1 = (b_1, \cdots, b_n)$, Gram matrix F_1 and maximal norm of basis vector M_1
- A second lattice $(\mathcal{L}_2, \Phi_2)$ with matrix basis $B_2 = (b'_1, \cdots, b'_n)$, Gram matrix F_2 and maximal norm of basis vector M_2
- The set $Iso(\mathcal{L}_1, \mathcal{L}_2)$ defined as above.

To find an isometry g between two lattices $\mathcal{L}_1$ and $\mathcal{L}_1$, it suffices to determine good images of the vector basis. More precisely, we have to determine a basis $(v_1, \cdots, v_n)$ from the set of short vectors S_2 of the lattice $\mathcal{L}_2$ satisfying:

$$\Phi_2(v_i, v_j) = \Phi_1(b_i, b_j) \quad \forall 1 \leq i, j \leq n;$$

As we are looking for vector in $\mathcal{L}_2$ whose norm matches those of the basis vector, then we define S_2 by:

$$S_2 = \{v \in \mathcal{L}_2 : \Phi_2(v, v) \leq M_1\}$$

The algorithm to determine S_2 is similar as the one describe in chapter 3. We only have to use the bilinear form of the second lattice Φ_2 instead of the one of the first lattice.
As in the automorphism case, we need the candidate images in order to find good images of these basis vectors. To find the candidate images, we enumerate all element from the sets $SN2_i$ for all $1 \leq i \leq n$. These sets form the list of vectors from S_2 with same norm as the basis vectors of $\mathcal{L}_1$. This list is called the list of same norm 2 and it is denoted : $SN2 = \{SN2_i, \text{ for } i = 1, \cdots, n\}$. The set $SN2_i$ are defined for every $1 \leq i \leq n$ by

$$SN2_i = \{v \in S_2 : \Phi_2(v, v) = \Phi_1(b_i, b_i)\}$$

We can now define the candidate images in the isometry case.

Definition 48 *We say that a vector $v \in SN2_i$(for some $1 \leq i \leq n$ fixed) satisfies the* ***inner product conditions****(in the isometry case) with respect to a list of vectors $(v_1, \cdots, v_k)$(with $k \leq n$) from $\mathcal{L}_2$, if $\Phi_2(v, v_j) = \Phi_1(b_i, b_j)$ for $j = 1, \cdots, k$ and $j \neq i$.*

Definition 49 *A* ***candidate vector*** *at a step k of a basis vector b_i(for $1 \leq i \leq n$) is a lattice element c with same norm as b_i, satisfying the inner product conditions with respect to a list of vectors $(v_1, \cdots, v_k)$ (with $k \leq n$) from the second lattice $\mathcal{L}_2$. More precisely, it is a vector $c \in SN_i$ such that $\Phi_2(c, v_j) = \Phi_1(b_i, b_j)$ for $j = 1, \cdots, k$ and $j \neq i$. For each i, the set C_{ki} is the set of all candidate images of the basis vector b_i.*

Before determining the set C_{ki} (for some $1 \leq i, k \leq n$), Let us first write an algorithm to define the set $SN2_i$.

5.1 Algorithm to find $SN2_i$

What follows is an algorithm to define the list $SN2$ of the sets $SN2_i$ of lattice element with same norm as the basis vector b_i.

Pseudocode

Input: F_1 (the Gram matrix of the lattice $\mathcal{L}_1$), F_2 (the Gram matrix of the lattice $\mathcal{L}_2$) and S_2 (the list of coordinates of short vectors of the lattice $\mathcal{L}_2$)
Output: $SN2$ (the list $SN2$ of the sets $SN2_i$)

Algorithm 10 the list SN of the sets $SN2_i$

```
1: procedure same − norm(F_1, F_2, S_2)
2:     D_1 ← Diagonal(F_1)
3:     n ← number of columns of F_1
4:     for u ∈ S2 do
5:         if Φ_2(u, u) ∈ D_1 then
6:             K ← list − indices(Φ_2(u, u) ∈ D_1)
7:             for i ∈ K do
8:                 append(u, SN2_i)
9:     return SN2
```

5.2 Algorithm to find C_{ki}

Here is an algorithm to find at a step k the list of candidates images C_{ki} of a lattice element with same norm as the basis vector b_i. We consider the two lattices as above. We assume that the good image of the list of k-vectors $(b_1, \cdots, b_{k-1})$ has been found already. This means that we have a k-partial isometry. A **k-partial isometry** is a partial map $(v_1, \cdots, v_k)$ of vectors from the lattice $\mathcal{L}_2$ that sends b_i to v_i for $1 \leq i \leq k$, satisfying $\Phi_2(v_i, v_j) = \Phi_1(b_i, b_j)$ for all $1 \leq i, j \leq k$. When $k = n$, it is called an isometry. So we set the list of candidate images of the list of $(k-1)$-vectors $(b_1, \cdots, b_{k-1})$ to be the empty list (as their good images are $(v_1, \cdots, v_{k-1})$), and we determine the remaining candidate.

Pseudocode

input: F_1 (the Gram matrix of the lattice $\mathcal{L}_1$), F_2 (the Gram matrix of the lattice $\mathcal{L}_2$), S_2(the list of coordinates of short vectors of the lattice $\mathcal{L}_2$), a $(k-1)-$partial isometry $[v_1, \cdots, v_{k-1}]$ and the index k, i.
output: C_{ki} (the list of candidates images)

Algorithm 11 the list of candidates images C_{ki}

```
1: procedure cand − vect − iso(F1, F2, SN2, [v1, ··· , vk−1], k, i)
2:     n ← number of columns of F1
3:     if i < k then
4:         return [ ]
5:     else
6:         for u in SN2_i do
7:             if Φ2(u, vj) = Φ1(bk, bj) (∀j = 1, ··· , k − 1) then
8:                 append(u, Ci)
9:     return Ci
```

Recall that, our aim is to construct the set

$$\{(v_1, \cdots, v_n) \text{ from } \mathcal{L}_2 : (v_1, \cdots, v_n) \text{ is a good image of } (b_1, \cdots, b_n)\}.$$

We start by the naive algorithm of the search of elements of this set.

5.3 Naive algorithm

To determine such a list of vectors $(v_1, \cdots, v_n)$, the idea of the naive algorithm is to first choose a partial list of good images $(v_1, \cdots, v_k)$(for $1 \leq k \leq n$) using the set of candidate images(we will start with $k = 1$ and take v_1 in $SN2_1$); then check if $(v_1, \cdots, v_k)$ is a k-partial isometry; if it is the case, then we choose a v vector in C_{k+1}, check if $(v_1, \cdots, v_k, v)$ is a $(k+1)-$partial isometry; if so, then v is a good image for b_{k+1}; we keep the new list of $(k+1)-$partial isometry and we continue this way up to $k = n$ and get one isometry if all the above checks are satisfied.

Each time that one of these checks is not satisfied, we take another vector in the set C_{k+1} until this set is empty. To find all isometries, we proceed as above for every possible partial list of good images $(v_1, \cdots, v_k)$(for $1 \leq k \leq n$). Indeed, having found an isometry $(v_1, \cdots, v_{n-1}, v)$, to find find others isometries, we first keep the $(n-1)-$partial isometry $(v_1, \cdots, v_{n-1})$; next, we test if this $(n-1)-$partial isometry together with each candidate image v of b_n form an isometry. We keep the isometry $(v_1, \cdots, v_{n-1}, v)$ found. Once all candidate images of b_n have been tested, we go back to the index $k = n - l$(starting from $l = 1$ up to $l = n-1$). For each l, while the set C_{n-l} is non empty, we choose another candidate image of b_{n-l}, we increase the index $k = n - l$ to $k = n - l + 1$ and we proceed as above.
The algorithm will stop whenever the set of candidate images of the first basis vector b_1 is empty.

pseudocode

To find all isometries, we need the procedure $is - i-$partial isometry. This procedure checks whether the list of vectors $[v_1, .., v_{i-1}]$ preserves the inner product with the $[b_1, .., b_{i-1}]$; i.e $[v_1, .., v_{i-1}, v_i]$ is again a $i-$partial when we add a vector v_{i-1} to a $(i-1)-$partial $[v_1, .., v_{i-1}]$(the v_i are vectors from the second lattice $\mathcal{L}_2$). This function returns True if so and False otherwise.
Input: F_1(the Gram matrix of the lattice $\mathcal{L}_1$) ,F_2(the Gram matrix of the lattice $\mathcal{L}_2$) and the list of vectors $[v_1, \cdots, v_k]$ to test.
Output: True or False

Algorithm 12 is-i-partial

```
1: procedure is − i − partial(F1, F2, [v1, .., vi−1, vi])
2:     n ← number of columns of F1
3:     ans ← True
4:     for j = 1, · · · , i do
5:         if Φ2(v, vj) ≠ Φ1(bi, bj) then
6:             ans ← False
7:             break
8:     return ans
```

The following procedure computes one (or all) isometries between the two lattices described above(with basis vectors $B = (b_1, \cdots, b_n)$). To make our naive backtrack search, we consider the sets of n elements ind(list of horizontal index on the tree formed by the vectors images), k(the number of candidate images of the basis vector b_i), and V(current list of vectors images); we initialise the cells of these sets at zero. We also consider the current list S_{tp} of candidates of vectors images (start with $i = 1$ i.e vectors having same norm with b_n).
Input: F_1 (the Gram matrix of the lattice $\mathcal{L}_1$), F_2 (the Gram matrix of the lattice $\mathcal{L}_2$), and $SN2$ (defined as above).
Output: *sol* (the list of all isometries of a lattice)

Algorithm 13 All (or one) isometries of a lattice

```
1: procedure naive − algo − iso(F_1, F_2, SN2, one − iso = False)
2:     n ← number of columns of F_1
3:     S_tp ← SN2_1,  k_1 ← len(S_tp),  i ← 1
4:     while i ≤ n and ind_1 < k_1 do
5:         k_i ← len(S_tp),  find − vi ← False
6:         if i < n then
7:             for j = ind_i, ··· , k_i do
8:                 kpartial ← [V_1, ··· , V_i],  appen(S_tp[j], kpartial)
9:                 L ← cand − vect − iso(F_1, F_2, SN2, kpartial, i + 1, i + 1)
10:                if is − i − partial(F_1, F_2, [v_1, .., v_{i−1}, v_i]) then
11:                    V_i ← S_tp[j],  S_tp ← L,  find − vi = True
12:                    break
13:                else
14:                    if j < k_i − 1 then
15:                        ind_i ← ind_i + 1
16:            if find − vi then
17:                i ← i + 1
18:            else
19:                while ind_i = k_i − 1 and i > 1 do
20:                    ind_i ← 0,  i ← i − 1
21:                ind_i ← ind_i + 1
22:                S_tp ← cand − vect − iso(F_1, F_2, SN2, [V_1, ··· , V_i], i, i)
23:        else
24:            for j = 1, ··· , k_i do
25:                if is − i − partial(F_1, F_2, [V_1, .., V_{n−1}, S_tp[j]]) then
26:                    V_n ← S_tp[j]
27:                    append(V, sol)
28:                if one − iso and length(sol) = 1 then
29:                    return sol
30:            i ← n − 1
31:            while ind_i = k_i − 1 and i > 1 do
32:                ind_i ← 0,  i ← i − 1
33:            ind_i ← ind_i + 1
34:            if i = 1 and ind_i > k_i − 1 then
35:                return sol
36:                break the while loop
37:            if i = 1 then
38:                S_tp ← SN_1
39:            else
40:                S_tp ← cand − vect − iso(F_1, F_2, SN2, [V_1, ··· , V_i], i, i)
41:    return sol
```

This algorithm will terminates as the set $Iso(\mathcal{L}_1, \mathcal{L}_2)$ is finite. Indeed, if $g \in Iso(\mathcal{L}_1, \mathcal{L}_2)$ then $g(b_i) \in S_2$ for every $1 \leq i \leq n$ and the list $(g(b_1), \cdots, g(b_n))$ determines g; This implies that $|Iso(\mathcal{L}_1, \mathcal{L}_2)| \leq |S_2|^n$ which is finite since the set of short vectors S_2 (shown in chapter 3) is finite.

5.4 Isometries search with fingerprint

Here, we present an a method to compute either one or all isometries between the two lattices described above using the fingerprint test. This fingerprint test is done using the fingerprint f_1 of the first lattice $\mathcal{L}_1$ (that is computed as in chapter 4) and the number of extensions in the isometry case.

Definition 50 *The number of extensions (in the isometry case) of a k-partial isometry $(v_1, \cdots, v_k)$ is the number of vector $v \in S_2$ satisfying:*

$$\begin{cases} \Phi_2(v, v) = \Phi_1(b_{k+1}, b_{k+1}) \\ \Phi_2(v, v_j) = \Phi_1(b_{k+1}, b_j), \ \textit{for } j = 1, \cdots, k. \end{cases}$$

This number is denoted by $nbExt - iso$. It is defined by:
$nbExt - iso(v_1, \cdots, v_k) = |\{v \in SN2_{k+1} : \Phi_2(v, v_j) = \Phi_1(b_{k+1}, b_j), \text{ for } j = 1, \cdots, k\}|$.

For k-partial isometry $(v_1, \cdots, v_k)$, this number is simply zero as all vectors in the k-partial isometry are already good images of the basis vector b_1 up to b_k. Therefore, for any $1 \leq i, k \leq n$ and any k-partial isometry $(v_1, \cdots, v_k)$, we may consider the following:

$$nbExt - iso([v_1, \cdots, v_k], k, i) = \begin{cases} 0 & \text{if } i < k \\ \left| \{v \in SN2_i : \Phi_2(v, v_j) = \Phi_1(b_i, b_j), \text{ for } j = 1, \cdots, k-1\} \right| & \text{otherwise.} \end{cases}$$

This number of extensions is exactly the number of candidate images C_{ki} for each fixed i and k. To compute this number, we will use the following algorithm.

5.4.1 Algorithm to find the number of extensions

Here is an algorithm to find at a step k the list of candidates images C_{ki} of a lattice element with same norm as the basis vector b_i. We consider the lattices $\mathcal{L}_1$ and $\mathcal{L}_2$ as above.

Pseudocode

input: F_1 (the Gram matrix of the lattice $\mathcal{L}_1$), F_2 (the Gram matrix of the lattice $\mathcal{L}_2$) the set of same norm $SN2$, the k-partial isometry $kpartial = [v_1, \cdots, v_k]$ to test and the index k, i.
output: $nExt - iso$ (the number of extensions)

Algorithm 14 the list of candidates images C_{ki}

1: **procedure** $nbExt - iso(F_1,\ F_2,\ SN2,\ kpartial,\ k,\ i)$
2: $n \leftarrow$ number of columns of F
3: $nExt - iso \leftarrow length(cand - vect - iso(F_1,\ F_2,\ SN2,\ kpartial,\ k,\ i))$
4: **return** $nExt - iso$

To determine each such list of good image $(v_1, \cdots, v_n)$ of the basis vector using the fingerprint test, the idea is to first consider the naive algorithm of the isometry described in the previous section; Next, in that algorithm, before checking if the list $(v_1, \cdots, v_k, v)$ is a $(k+1)$-partial automorphism, we first check if the number of extensions of $(v_1, \cdots, v_k, v)$ corresponds to the number of extensions of $(b_1, \cdots, b_k, b_{k+1})$. This is helpful to detect as soon as possible, a k-partial isometry that cannot be extended into an isometry. The algorithm is the following.

Pseudocode

Input: F_1 (the Gram matrix of the lattice $\mathcal{L}_1$), f_1 (the fingerprint of the lattice $\mathcal{L}_1$), F_2 (the Gram matrix of the lattice $\mathcal{L}_2$), and $SN2$ (defined as above).
Output: *sol* (the list of all isometries of a lattice)

Algorithm 15 All (or one) isometries of a lattice

procedure $algo-iso-fingerprint(F_1, f_1, F_2, SN2, one-iso = False)$
  $n \leftarrow$ number of columns of F_1
  $S_{tp} \leftarrow SN2_1, \quad k_1 \leftarrow len(S_{tp}), \quad i \leftarrow 1$
  while $i \leq n$ *and* $ind_1 < k_1$ **do**
    $k_i \leftarrow len(S_{tp}), \quad find-vi \leftarrow False$
    if $i < n$ **then**
      for $j = ind_i, \cdots, k_i$ **do**
        $kpartial \leftarrow [V_1, \cdots, V_i], \quad append(S_{tp}[j], kpartial)$
        $L \leftarrow cand-vect-iso(F_1, F_2, SN2, kpartial, i+1, i+1)$
        if $nbExt-iso(F_1,\ F_2,\ SN2,\ kpartial,\ i+1,\ +1) = f_1[i+1, i+1]$ **then**
          $V_i \leftarrow S_{tp}[j], \quad S_{tp} \leftarrow L, \quad find-vi \leftarrow True$
          break
        else
          if $j < k_i - 1$ **then**
            $ind_i \leftarrow ind_i + 1$
      if $find-vi$ **then**
        $i \leftarrow i + 1$
      else
        while $ind_i = k_i - 1$ *and* $i > 1$ **do**
          $ind_i \leftarrow 0, \quad i \leftarrow i - 1$
        $ind_i \leftarrow ind_i + 1$
        $S_tp \leftarrow cand-vect-iso(F_1, F_2, SN2, [V_1, \cdots, V_i], i, i)$
    else
      for $j = 1, \cdots, k_i$ **do**
        if $is-i-partial(F_1, F_2, [V_1, .., V_{n-1}, S_{tp}[j]])$ **then**
          $V_n \leftarrow S_{tp}[j]$
          $append(V,\ sol)$
        if $one-iso$ and $length(sol) = 1$ **then**
          return *sol*
      $i \leftarrow n - 1$
      while $ind_i = k_i - 1$ *and* $i > 1$ **do**
        $ind_i \leftarrow 0, \quad i \leftarrow i - 1$
      $ind_i \leftarrow ind_i + 1$
      if $i = 1$ *and* $ind_i > k_i - 1$ **then**
        return *sol*
        break *the while loop*
      if $i = 1$ **then**
        $S_{tp} \leftarrow SN_1$
      else
        $S_{tp} \leftarrow cand-vect-iso(F_1, F_2, SN2, [V_1, \cdots, V_i], i, i)$
  return *sol*

5.5 Search for Isometries of lattices

We follow the method describe in the above proposition. Recall that from the above proposition, to determine the set of all isometries between two lattices $\mathcal{L}_1$ and $\mathcal{L}_2$, the idea is to first determine only one isometry g between these two lattices, then find all the automorphisms of the first lattice $\mathcal{L}_1$, and deduce $Iso(\mathcal{L}_1, \mathcal{L}_2)$ by taking the composition of g with each of the automorphism found of $\mathcal{L}$. We will use the algorithm for isometry using fingerprint to determine the first isometry. And the algorithm of automorphisms described in chapter 3 to determine the automorphisms of the first lattice.

pseudocode

input: F_1 (the Gram matrix of the lattice $\mathcal{L}_1$), f_1 (the fingerprint of the lattice $\mathcal{L}_1$), $SN1$ (defined as in the automorphism case), F_2 (the Gram matrix of the lattice $\mathcal{L}_2$) and $SN2$ (defined as above).
output: *sol* (the list of all isometries)

Algorithm 16 the list of candidates images C_{ki}

1: **procedure** $algo - iso - auto(F_1, f_1, SN1, F_2,\ SN2)$
2: $sol\ =\ [\ ]$
3: "Computing the first isometry"
4: $g = algo - iso - fingerprint(F_1, f_1, F_2, SN2, True)$
5: "Computing the list of all automorphisms of the first lattice"
6: $List - auto1 = auto(F_1, SN1, f_1)$
7: **for** θ in $List - auto1$ **do**
8: $append(\theta \circ g,\ sol)$
9: **return** *sol*

5.6 Runtime of algorithms described

The aim of this section is to test and give the computation time of the three algorithms for finding isometries of lattices described above. This will be done on some examples where the bases of the lattices involved are isometrics or not. In the first section, we construct some isometrics bases to be used in order to ensure our algorithms results. The second section provided the test and the comparison of these algorithms on examples based on the construction of isometrics bases made.

5.6.1 Construction of isometrics lattices

In what follows, we would like to illustrate how one may construct two isometrics lattices. It suffices to construct from a given basis matrix B of the first lattice, a second basis matrix V

such that these two basis are isometrics. For this purpose, we consider two lattices with basis vectors: $B = (b_1, \cdots, b_n)$ and $V = (v_1, \cdots, v_n)$, and with bilinear forms given by the inner product. The following lemma is a tool for the construction of such basis.

Lemma 51 *Assume that there exists coefficients $\alpha_{ij} \in \mathbb{Z}$ such that $b_i = \sum_{j=1}^{n} \alpha_{ij} v_j$. Then there exists an isometry g between B and V.*

PROOF. We set $g(b_i) = \sum_{j=1}^{n} \alpha_{ij} v_j := b_i$, for each $i = 1, \cdots, n$.
Since $g(b_i) := b_i$ and the bilinear forms are given by the inner product, then g preserves the bilinear forms. It is a $\mathbb{Z}$-linear bijection by definition. Moreover, from this definition, g sends each coordinates $(0, \cdots, 0, 1, 0, \cdots, 0)^T$ (with 1 in the i-th position) of the basis vectors b_i to the coordinates vectors $(\alpha_{i1}, \alpha_{i2}, \cdots, \alpha_{ii}, \cdots, \alpha_{in})^T$; thus we get:

$$((b_1, \cdots, b_n)) = (v_1, \cdots, v_n) \begin{pmatrix} \alpha_{11} & \alpha_{21} & \cdots & \alpha_{n1} \\ \vdots & \vdots & \cdots & \vdots \\ \alpha_{1n} & \alpha_{2n} & \cdots & \alpha_{nn} \end{pmatrix}$$

Hence, g is really an isometry that sends B to V.

From this lemma, we know that if we have basis vectors B and V as above such that the b_i's are linear combinations of the v_i's, then the basis B and V are isometrics. The following proposition involve an example of construction of isometric basis.
We consider B and V as above and we assume that:

$$\begin{aligned} v_1 &= b_1 \\ v_2 &= \alpha_{21} b_1 + b_2 \\ &\vdots \\ v_k &= \sum_{j=1}^{k-1} \alpha_{kj} b_j + b_k \\ &\vdots \\ v_n &= \sum_{j=1}^{n-1} \alpha_{kj} b_j + b_n \end{aligned}$$

Proposition 52 *Considering these assumptions, the basis B and V are isometrics.*

PROOF. From the above lemma, it suffices to show that there exists $\beta_{ij} \in \mathbb{N}$ such that $b_i = \sum_{j=1}^{n} \beta_{ij} v_j$ for all $i \in \{1, \cdots, n\}$.
We prove this by induction on i.

1) For $i = 1$: since $v_1 = b_1$, set $\beta_{11} = 1$ and $\beta_{1j} = 0$ $\forall j > 1$ and get $b_1 = \sum_{j=1}^{n} \beta_{1j} v_j$.

2)Assume for $i = 1, \cdots, k$ that there exists $\beta_{ij} \in \mathbb{N}$ such that $b_i = \sum_{j=1}^{n} \beta_{ij} v_j$. Let us prove that there exists $\beta_{k+1,j} \in \mathbb{N}$ such that $b_{k+1} = \sum_{j=1}^{n} \beta_{k+1,j} v_j$.
We know that $b_{k+1} = v_{k+1} - \sum_{j=1}^{k} \alpha_{k+1,j} b_j$ and by assumption $\forall j \in \{1, \cdots, k\}$ there exists $\beta_{jl} \in \mathbb{N}$ such that $b_j = \sum_{l=1}^{n} \beta_{jl} v_l$. From this, we get:

$$\begin{aligned} b_{k+1} &= v_{k+1} - \sum_{j=1}^{k} \alpha_{k+1,j} \Big(\sum_{l=1}^{n} \beta_{jl} v_l \Big) \\ &= v_{k+1} - \sum_{l=1}^{n} \Big(\sum_{j=1}^{k} \alpha_{k+1,j} \beta_{jl} \Big) v_l \end{aligned}$$

Since $\sum_{j=1}^{k} \alpha_{k+1,j} \beta_{jl} \in \mathbb{N}$ for all l, then one may set $\beta_{k+1,l} = \sum_{j=1}^{k} \alpha_{k+1,j} \beta_{jl} \in \mathbb{N}$. Thus $b_{k+1} = \sum_{j=1}^{n} \beta_{k+1,j} v_j$.
This proves the proposition.

From this proposition, we can deduce the following corollary.

Corollary 53 *Given a basis $B = (b_1, \cdots, b_n)$ and arbitrary coefficients $\alpha_{ij} \in \mathbb{N}$ such that $\alpha_{ii} = 1$ for $i = 1, \cdots, n$ and $\alpha_{ij} = 0$ for $j > i$ and $j \in \{1, \cdots, n\}$, one can construct an isometric basis $V = (v_1, \cdots, v_n)$ of B, with the v_i's defined by $v_i = \sum_{j=1}^{n} \alpha_{ij} b_j = b_i + \sum_{j<i} \alpha_{ij} b_j$. Moreover, the basis V can be computed using the matrix expression: $V = BM$, where $M = \begin{pmatrix} 1 & \alpha_{21} & \cdots & \alpha_{n1} \\ 0 & \cdot & \cdot & \vdots \\ \vdots & \cdot & \cdot & \alpha_{n,n-1} \\ 0 & \cdot & 0 & 1 \end{pmatrix}$*

Proof. This is just a consequence of the above proposition.

5.6.2 Computation time

Here we use three examples to illustrate the test and to present the computation time of our algorithms.

Example 54 *In this example, we want to verify whether our algorithms returns an empty list when they have as input not isometric lattices. For this purpose, we consider the identity lattice n and the root lattice $\mathbb{A}_n$ with Gram matrix $F_{\mathbb{A}_n}$ defined with 2 on the diagonal entries, -1 under*

and over the diagonal entries and the 0 *elsewhere. It is given by:*

$$F_{\mathbb{A}_n} = \begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & \cdot & \cdot & \cdots & \cdot \\ 0 & \cdot & \cdot & \cdot & \vdots \\ \vdots & \cdot & \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{pmatrix}.$$

These two lattices are not isometric as $det(F_{Id_n}) = 1 \neq 5 = det(F_{\mathbb{A}_n})$. *We run our Sage code in dimension* 4 *and we get the following results.*

Returns	*iso-naive*	*iso-fingerprint*	*iso-auto*
time(s)	*0.0*	*0.0*	*0.0*
$card(Iso(Id_4, \mathbb{A}_4))$	*0*	*0*	*0*

This table ensure our algorithms results when not isometric lattices are given. Moreover, we realised that our Sage code produces this results in time less than a second.

The following example uses the above corollary to compute from a given basis B of a lattice, a second basis V for a second lattice in the case where $M = (\alpha_{ij})_{(1 \leq i,j \leq n)}$, with n equals to the number of b_i's; the matrix M has 1 on the diagonal and under the diagonal, and the 0 in other entries. Then, we find the isometries between B and V using our algorithms. The computation time of these algorithms is also considered.

Example 55 *Here, we take* $B = \begin{pmatrix} 0 & 2 & 0 \\ 2 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$ *and with our Sage code we compute* $V = \begin{pmatrix} 2 & 2 & 0 \\ 3 & 2 & 1 \\ 0 & 1 & 1 \end{pmatrix}$. *Then we compute the Gram matrices* $F_B = B^T B$ *and* $F_V = V^T V$ *Finally, we apply these Gram matrices to our Sage code and we obtain the following results.*

Returns	*iso-naive*	*iso-fingerprint*	*iso-auto*
time(s)	0.0	0.0	0.015
$card(Iso(B, V))$	16	16	16

Chapter 6

Conclusion

In this work, we first give an overview on lattices. This was done by giving definitions and examples of lattice, lattice elements, quadratic and bilinear form associated to a given lattice; Also, we gave a brief description and application of some matrix decomposition; The definitions and properties of automorphisms and isometries were also involved. Second, we describe a method for computing short vectors of a given lattice $(\mathcal{L}, \Phi)$; These are vectors v in $\mathcal{L}$ satisfying $\Phi(v, v) \leq M$, where M is the maximal norm of the basis vectors of the lattice $\mathcal{L}$; These methods use either the Cholesky decomposition or the LDL^T-decomposition of the LLL reduced form of the Gram matrix of the lattice involved; We also study the complexity analysis of these methods based on [4] and a computation time of these algorithms on an example. Third, we present an algorithm for computing the group of automorphisms of lattices; Proposed by [1], this method is achieved using partial maps and the fingerprint of the Gram matrix involved; The description of this backtrack search algorithm is followed by a computation time measurement. We end by presenting methods to compute isometries between two lattices. These methods are based on the backtrack search described on [1] to find the group of automorphisms of lattices. All algorithms described in this essay were coded and tested in Sage 8.1 using standard packages. Future work with these methods would involve reducing the computation time by using for instance vector sums or Bacher polynomials as in [1]. One can also use the algorithm for computing isometries to construct the genus of lattices. Lastly, these methods could also be extended in general to work over number fields.

Acknowledgements

This essay owes its existence to the help, support and inspiration of kind people around me. I would like to extend my sincere thanks to all of them.

Above all, I am grateful to the almighty God for the divine intervention in this academic year and for establishing me to complete this work.

I would like to express my deepest sense of gratitude to my supervisor Dr. Aurel Page. I am extremely grateful and indebted to him for his expertise, patience, sincere, valuable guidance and encouragement extended to me. I place on record, my sincere thanks to my supervisor for his unwavering support, understanding, corrections and encouragement. A special thank to Pr. Dr Christine Bachoc for having allowed me to discover, to like, and to write my essay on Arithmetic Number Theory.

I would like to offer my special appreciation and thanks to Pr.Dr. Marco Garuti, to Pr. Dr Bruno chiarellotto for their devotion and the ALGANT consortium for providing me all the necessary facilities and for allowing me to spend a blessed and memorable academic years. I specially express my sincere thanks to Pr.Dr. Marco Garuti, for his constant encouragement, support and advices. A special thanks to all the lecturers who have greatly boosted my knowledge. I am very grateful.

I am thankful to all my family and friends. My lovely parents Annie Teagho and Etienne Teagho; my brothers and sisters, Frank, Yasmine, Urmes and Stan; my friends Chiara and Marie Christine, thank you to always believe in me and to always being there for me. A special thanks to my husband Walter Simo, for his unwavering support, corrections and encouragement. You have given me your unequivocal support throughout, as always, for which my mere expression of thanks likewise does not suffice. A special thanks, to my course mates of the University of Bordeaux(France), you made my stay here at Bordeaux memorable. I end by placing on record my sense of gratitude to one and all who, directly or indirectly, have lent their helping hand in this venture.

Bibliography

[1] W. PLESKEN and B. SOUVIGNIER. Computing isometries of lattices. Journal of Symbolic Computation, 24(3) :327-334, 1997.

[2] Christine Bachoc. Lecture Note, University of Bordeaux, 2017.

[3] Henri Cohen. A course in computational algebraic number theory, volume 138 of Graduate Texts in Mathematics. Springer-Verlag, Berlin, 1993.

[4] U. Fincke and M. Pohst. Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. Math. Comp., 44(170): 463-471, 1985.

[5] Guillaume Hanrot and Damien Stehlé. Improved analysis of Kannans shortest lattice vector algorithm (extended abstract). In Advances in cryptology-CRYPTO 2007, volume 4622 of Lecture Notes in Comput. Sci., pages 170-186. Springer, Berlin, 2007.

[6] L. Lagrange. Recherches d'arithmetique. Nouv. Mdm. Acad, pages 265-312, 1773

[7] C.F. Gauss. Disquisitiones arithmeticae. 1801.

[8] Hermann Minkowski. Geometrie der Zahlen. Leipzig: Teubner, 1910.

[9] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovasz. Factoring polynomials with rational coefficients. 261(4):515-534, 1982