L3 Internship report: Walking along the infrastructures of real quadratic and pure cubic fields

Joël Felderhoff
ENS de Lyon
Intership realized under the supervision of Pierre-Jean Spaenlehauer in team CARAMBA, a joint project-team with INRIA Nancy - Grand Est, CNRS and Université de Lorraine

Abstract

The SQUFOF algorithm, which heuristically finds in time $\mathcal{O}(N^{\frac{1}{4}})$ a factor of a number N, is based on the walk over the infrastructure of the group class of the real quadratic field $\mathbb{Q}(\sqrt{N})$. The purpose of my internship was to investigate if such an algorithm could be generalized to other number fields of unit rank one: complex cubic and totally complex quartic fields. During this internship, after a study and an implementation of the classical SQUFOF algorithm, I studied the reduction step of the walk over the infrastructure in the particular case of the pure cubic fields $\mathbb{Q}(\sqrt[3]{D})$. In order to do that, I studied and implemented the Voronoï algorithm. This implementation was followed by several tests.

1 Introduction

1.1 Context, motivations and problematic

The problem of factoring integers has theoretical and practical applications. The most common example is the study of the security of the RSA public key cryptosystem. This system, which is used for example in some implementations of the SSH protocol or in the E.M.V. standard for credit cards, is based on the difficulty of the factoring problem. The RSA cryptosystem is based on two keys: a public one to encrypt the messages and a private one to decrypt them. The private key can be found, given the public key, by factoring a RSA number: a number $N = p \cdot q$ with p, q two large prime numbers (about 1024 bits long).

The best algorithm for factoring RSA numbers known today is the general number field sieve, whose complexity is heuristically sub-exponential: $\exp\left(\left(\sqrt[3]{\frac{64}{9}} + o(1)\right)(\ln(N))^{\frac{1}{3}}(\ln(\ln(N)))^{\frac{2}{3}}\right)$. In this algorithm, in order to factorize large integers (more than 500 bits long), one must be able to factorize smaller numbers. This necessity justifies the study of factorization algorithms which work well for a large range of numbers. One of those algorithms is the SQUFOF algorithm, created by Daniel Shanks in 1975. This algorithm computes a non-trivial divisor of a number D in a time in $\mathcal{O}(\sqrt[4]{D})$ under heuristic hypothesis. It is based on a research of a certain kind fractional ideals (ideals divided by an integer) in the ring of algebraic integers of the number field $\mathbb{Q}(\sqrt{D})$, where D is the number one wants to factorize (or a small multiple of this number). One of the reason why this algorithm works is that one can, by a simple algorithm, "skip" from one fractional ideal to another in a determined order: we talk about walking along elements of the infrastructure of $\mathbb{Q}(\sqrt{D})$. This is possible in the case of $\mathbb{Q}(\sqrt{D})$ because the free part of the group of invertible algebraic integers in this field is isomorphic to $\mathbb{Z}$.

There are two other kinds of number fields which have the same property: the complex cubic fields and the totally complex quartic fields. A natural question is then to ask if an algorithm analogous to SQUFOF can be written for those kinds of fields. The goal of this internship was to find out if the SQUFOF algorithm could be generalized, and to implement a cubic and/or a quartic version of

SQUFOF if this is the case. In order to do that, I studied several algorithms used to walk along this infrastructure. The original purpose of those algorithms was to find the regulator of the fields, an invariant used in a lot of modern number theoretic results, either algebraic or analytic.

1.2 State of art

The SQUFOF algorithm was first described in [6] and [7]. This algorithm is well known, and studied in details in [4] and [9]. Originally, Shanks gave an interpretation in terms of continued fraction expansion of the number $\sqrt{D}$ where D is the discriminant of the field $\mathbb{Q}(\sqrt{N})$, with N the positive integer we want to factorize. In [9], H.C. Williams gives an algebraic approach to the reduction step of the SQUFOF algorithm. This approach uses fundamental mathematical objects which enable us to think of a way to generalize the algorithm.

Infrastructures of cubic fields are however less documented. In [9], the reader is refered to an algorithm called "the Voronoï's algorithm" for walking along the infrastructure. Although this algorithm is cited in [9] and improved in [8], I did not find any implementation in the litterature. Williams, in [9] refers to Voronoï's thesis, but this thesis and its transcription in a book were written in Russian. I achieved to find a translated version [5]. In this book, there is a description and explanation of the Voronoï algorithm but there were several typographic mistakes in the translation, and the theory was more general that just the finding of the next step of the walk in the infrastructure. In fact, the Voronoï algorithm is the historical technique to compute the regulator of a pure cubic field.

1.3 My internship

The first part of my 6 weeks internship was to understand the SQUFOF algorithm. I mainly worked on its interpretation in terms of binary quadratic forms. My next task was to reformulate results I understood in terms of walks on the fractional ideals of the field. Thereafter I tried to generalize the algorithm in the case of pure cubic fields $\mathbb{Q}(\sqrt[3]{D})$. In parallel, I realized and tested an implementation of SQUFOF in c++.

The first part of the generalization of SQUFOF was to get an algorithm to walk along the infrastructure. I found out that the paper [3] presented a general algorithm to do that, the only missing part was the "skip" step. I then studied and implemented the algorithm of Voronoï, found in the book [5]. After the implementation, I made several tests on my implementation in order to compare it to the one used to produce the results in [2]. I tested the speed of my implementation and, as the research of the regulator was the primary goal of the Voronoï algorithm, I looked at the number of steps made by my implmentation to find it.

By lack of time, the only fields I studied during my internship were a special case of complex cubic fields: the pure cubic fields $\mathbb{Q}(\sqrt[3]{D})$. I only had the time to study and implement the first part of a "cubic squfof" : the walk among the infrastructure. Another reason why the SQUFOF algorithm works is that information about the factorization of D is stored in a certain kind of fractional ideals (called ambiguous ideals). In order to write a cubic SQUFOF one would have to be sure that this property generalizes to the case of pure cubic field.

1.4 Organization of this report

In Section 2 of this report, I describe some number theoretic notions and theorems which will be used in the algorithms I will study in the next parts. More precisely, I will introduce the notions of number field and their infrastructures. Section 3 will focus on the special case of real quadratic fields, in order to explain the particularities of this kind of fields which enable us to create an algorithm like SQUFOF. Then I will describe precisely this algorithm. In Section 4 I will study the Voronoï algorithm, which is the equivalent of the "skip" step in SQUFOF. Since this algorithm deals with three dimensional lattices, it is very geometric so in order to explain it I will introduce several classical results about positive definite binary quadratic forms. I will conclude this report in Section 5 by a presentation of my productions in the context of my internship: my implementations of the SQUFOF and Voronoï algorithm and some benchmarks.

2 Number fields and infrastructures

2.1 Definitions

I start by introducing the mathematical objects which will be used through this report.

Definition 1. *A* number field *is a finite algebraic extension of $\mathbb{Q}$, i.e. a field K is a number field if it contains $\mathbb{Q}$ and it is a finite dimensional vector space over $\mathbb{Q}$.*

A number field is isomorphic to the quotient of the ring $\mathbb{Q}[X]$ by an irreducible polynomial. Then the degree of the extension is the degree of this polynomial. As $\mathbb{Q}$ has characteristic 0, it is a perfect field: any number field K is a separable extension of $\mathbb{Q}$, so by the *primitive element theorem*, there exists $\theta \in K$ such as $K = \mathbb{Q}(\theta)$, and θ can be chosen as the root of an irreducible monic polynomial with coefficients in $\mathbb{Z}$.

In what follows we set $K = \mathbb{Q}[X]/\Pi$ ($\Pi \in \mathbb{Q}[X]$ an irreducible polynomial) a number field of degree $n = [K : \mathbb{Q}] = \deg(\Pi)$. I will let α denote the class of X in K.

Definition-Proposition 2. *[4, Theorem 4.1.3] If K is a number field, the set*

$$\mathcal{O}_K = \{x \in K \mid \exists P \in \mathbb{Z}[X] \text{ monic s.t. } P(x) = 0\}$$

is a ring, called the ring of integers *of K.*

This ring is the generalization of the notion of integers in $\mathbb{Q}$, and we will see in the following that this ring and more especially the fractional ideals of this ring have a central role in the structure of the number field K.

Definition 3. *A set $I_f \subset K$ is a fractional ideal of $\mathcal{O}_K$ if there exists $d \in \mathbb{Z}$ such that dI_f is an ideal of $\mathcal{O}_K$.*

Basically, fractional ideals are ideals of $\mathcal{O}_K$ divided by integers. These objects can be endowed with a group structure. The following statements describe this structure.

Definition 4 (Multiplication law on ideals)**.** *If I and J are fractional ideals, we denote by IJ the smallest fractional ideal of $\mathcal{O}_K$ containing $\{xy, x \in I, y \in J\}$.*

Proposition 5. *[4, Lemma 4.6.7] Every fractional ideal I of $\mathcal{O}_K$ is invertible, with inverse*

$$I^{-1} = \{x \in K \mid xI \subseteq \mathcal{O}_K\}.$$

Definition 6. *Two fractional ideals I, J are said to be equivalent if there exist $\gamma \in K$ such as $I = \gamma \cdot J$. This relation is an equivalence relation and I will denote it by $I \sim J$.*

A large amount of information about the number field can be extracted from the quotient of the group of fractional ideals of $\mathcal{O}_K$ by the equivalence relation defined in Definition 6.

Definition-Theorem 7 (Class group of K)**.** *[4, Theorem 4.9.2] The set $\{$fractional ideals of $\mathcal{O}_K\}/\sim$ is a group for the multiplication law on ideals. This group is called the* class group *of K and is finite.*

Proposition 8. *As the extension $K/\mathbb{Q}$ is separable, there are exactly n embeddings (injective morphism fixing $\mathbb{Q}$) of K in $\mathbb{C}$, I will denote them $\sigma_1, \ldots, \sigma_n$. Each of those embeddings maps α to one of the root of Π.*

Proof. The n embeddings sending α on one of the root of Π are clearly embeddings of K to $\mathbb{C}$. Let us show that these are the only embeddings. If σ is an embedding from K to $\mathbb{C}$, it is only defined by its value on α because $K = \mathbb{Q}[1, \alpha, \alpha^2, \ldots, \alpha^{n-1}]$. As σ is a field morphism, $\Pi(\sigma(\alpha)) = \sigma(\Pi(\alpha)) = 0$ so $\sigma(\alpha)$ must be a root of Π.

□

As $\Pi \in \mathbb{Q}[X]$, every complex root of Π comes with its complex conjugate. I will denote $\sigma_1, \ldots, \sigma_s$ the embeddings which sends α on a real root the others will be denoted $\sigma_{s+1} \ldots \sigma_{s+2t}$ (I will talk about real and complex embeddings).

Since each complex embedding comes with its conjugate, I will choose the numeration such that for any $i \in \{1, \ldots, t\}, \sigma_{s+i} = \overline{\sigma_{s+t+i}}$.

Definition 9. *The pair (s,t) is called the* signature *of K.*

Example 10. *A real quadratic field $\mathbb{Q}[X]/(X^2-D)$ with D a positive non-square integer has signature $(2,0)$ and its two embeddings in $\mathbb{C}$ are $\alpha \mapsto \sqrt{D}$ and $\alpha \mapsto -\sqrt{D}$.*

Example 11. *The field $\mathbb{Q}[X]/(X^2+X+1)$ has the signature $(0,1)$ and its only complex pair of conjugate embeddings are $X \mapsto j$ and $X \mapsto j^2$ with $j = e^{\frac{2i\pi}{3}}$.*

Those notions are useful to define a certain number of tools for the study of number fields.

Definition 12 (Norm)**.** *The* norm $N(x) \in \mathbb{Q}$ *of $x \in K$ is the product:*

$$N(x) = \prod_{i=1}^{n} \sigma_i(x).$$

Proposition 13. *Every algebraic integer in a number field has an integer norm.*

Proof. Let θ be an element of K. Let $A \in \mathbb{Z}[X]$ be its minimal polynomial, let m be its degree. By [4, Prop 4.3.2], the characteristic polynomial of θ is then $P_\theta(X) = \prod_{i=1}^{n}(X - \sigma_i(\theta)) = A(X)^{\frac{n}{m}} \in \mathbb{Q}[X]$. Since θ is a root of P_θ, the minimal polynomial of θ divides P_θ. If one takes $\theta \in \mathcal{O}_K$, the minimal monic polynomial of θ is monic in $\mathbb{Z}[X]$, so is P_θ. Therefore, $N(\theta) = \pm P_\theta(0) \in \mathbb{Z}$. □

Definition-Proposition 14 (Unit Group)**.** *The* unit group *of the ring of integers $\mathcal{O}_K$ of a field K is the group of invertible elements of $\mathcal{O}_K$. I denote it by $\mathcal{U}(K)$. Units in K are exactly the elements in $\mathcal{O}_K$ of norm ± 1.*

Proof. Let θ be an element of norm ± 1 in $\mathcal{O}_K$. Let us denote $P_\theta(X) = X^n + \sum_{i=1}^{n-1} a_i X^i \pm 1$. Then $P_\theta(\theta) = 0 = \theta^n + \sum_{i=1}^{n-1} a_i \theta^i \pm 1$. So $\theta^n + \sum_{i=1}^{n-1} a_i \theta^i = \pm 1 = \theta\left(\theta^{n-1} + \sum_{i=1}^{n-1} a_i \theta^{i-1}\right)$. As for all i $a_i \in \mathbb{Z}$, $\left(\theta^{n-1} + \sum_{i=1}^{n-1} a_i \theta^{i-1}\right) \in \mathcal{O}_K$. Hence θ is invertible.

Conversely, if $\theta \in \mathcal{O}_K$ is invertible, $N(\theta) \cdot N(\theta^{-1}) = N(\theta \cdot \theta^{-1}) = N(1) = 1$ so $N(\theta)$ is invertible in $\mathbb{Z}$, therefore $N(\theta) = \pm 1$.

□

2.2 Dirichlet unit theorem and logarithmic embedding

One cornerstone of the SQUFOF algorithm is the Dirichlet unit theorem, which describe the unit group in terms of the signature.

Theorem 15 (Dirichlet unit theorem)**.** *[4, Theorem 4.9.5] For a number field K of signature (s,t) the group $\mathcal{U}(K)$ is isomorphic to*

$$\mathbb{Z}/m\mathbb{Z} \times \mathbb{Z}^{s+t-1}$$

where $\mathbb{Z}/m\mathbb{Z}$ correspond to the group of roots of the unity of K, i.e. the elements $x \in K$ such that $\exists l \in \mathbb{Z}_{>0}, x^l = 1$.

The quantity $s+t-1$ is the *rank* of the unit group.

The SQUFOF algorithm uses the fact that the rank of the unit group of any real quadratic field is 1. In order to generalize this algorithm, I will study fields with unit group of rank one. There are 3 types of such fields:

- The real quadratic fields with signature $(2,0)$.
- The complex cubic fields with signature $(1,1)$.
- The totally complex quartic fields with signature $(0,2)$.

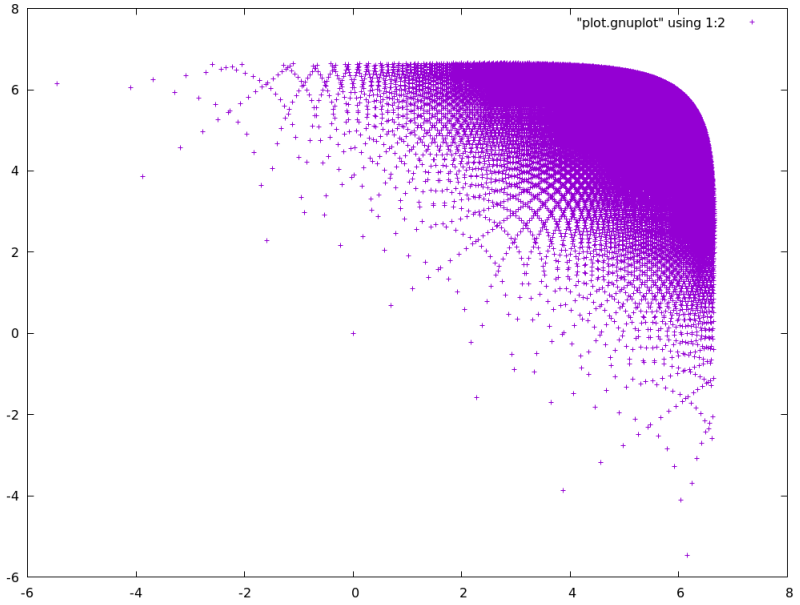

Figure 1: Some points of the logarithmic embedding of the ring of integers of $\mathbb{Q}(\sqrt{23})$.

Let us now introduce a function used to represent our number field in a more graphical way. This morphism is called the *logarithmic embedding* of K.

$$\begin{array}{rccl} \Phi: & K^* & \longrightarrow & \mathbb{R}^{s+t} \\ & x & \longmapsto & (\log(|\sigma_1(x)|), \ldots, \log(|\sigma_s(x)|), 2\log(|\sigma_{s+1}(x)|), \ldots 2\log(|\sigma_{s+t}(x)|)). \end{array}$$

The kernel of Φ is the set of roots of unity ([4, Theorem 4.9.7]), that is to say that any point of K will be represented modulo the action of this group.

Proposition 16. *The logarithmic embedding of a point $x \in K$ is located on the affine hyperplane defined by the equation $\sum_{i=1}^{s+t} x_i = \log|N(x)|$.*

Proof. Let x be an element of K^*, then

$$\sum_{i=1}^{s+t} \Phi(x)_i = \log\left(\prod_{i=1}^{n} |\sigma_i(x)|\right) = \log\left(\left|\prod_{i=1}^{n} \sigma_i(x)\right|\right) = \log|N(x)|\,.$$

□

As said before, in order to generalize SQUFOF, we want to consider only fields whose rank of unit group is 1. From now on, I will suppose that this is the case for the field K.

Remark 17. *As the logarithmic embedding is a group morphism from a multiplicative group to an additive one, the multiplication of a set by an element of K is equivalent to a translation of his image by the logarithmic embedding.*

Proposition 18. *The image of a fractional ideal via the logarithmic embedding is invariant by translation by the image of an unit.*

Proof. Let μ be an unit of $\mathcal{O}_K$ and I be an ideal of $\mathcal{O}_K$ (the proof generalize without any major modification to the case where I is a fractional ideal). I want to prove that $I = \mu I$. As I is an ideal, $\mu I \subset I$. Each element $x \in I$ can be written $\mu \cdot \mu^{-1} \cdot x$. As $\mu^{-1} \in \mathcal{O}_K$ by definition of μ, $\mu^{-1} \cdot x \in I$ so $x \in \mu I$. Therefore $I \subset \mu I$. In conclusion, $I = \mu I$. □

Definition 19. *For a number field with unit rank one, a* fundamental unit*, denoted by the letter ϵ, is a generator of the free part of the unit group, that is to say an element such that $\mathcal{U}(K)/\langle\epsilon\rangle$ is a finite cyclic group.*

During my internship I only had time to study the real quadratic fields, and a special case of complex cubic fields: the pure cubic fields. Therefore they are the only cases I will deal with in this report. From now, I am assuming that the field I am working with is either a real quadratic field $\mathbb{Q}[X]/(X^2 - D)$ or a pure cubic field $\mathbb{Q}[X]/(X^3 - D)$.

2.3 Real quadratic fields and pure cubic fields

In the case of real quadratic fields, I will denote $\sigma_1 : \alpha \mapsto \sqrt{D}$ and $\sigma_2 : \alpha \mapsto -\sqrt{D}$. I will identify this field with the field induced by its first embedding $\mathbb{Q}(\sqrt{D})$ in the rest of the report.

In the case of pure cubic fields, I will denote $\sigma_1 : \alpha \mapsto \sqrt[3]{D}$, $\sigma_2 : \alpha \mapsto j\sqrt[3]{D}$ and $\sigma_3 : \alpha \mapsto j^2\sqrt[3]{D}$. I will identify this field with the field induced by its first embedding $\mathbb{Q}(\sqrt[3]{D})$ in the rest of the report.

According to the Dirichlet theorem (Theorem 15), one can see that those fields have an unit group of rank one.

Definition-Proposition 20. *[4, 4.9.8] For quadratic and pure cubic fields, with the previous notations, the* regulator *of the field is defined as* $R = \log(|\sigma_1(\epsilon)|)$. *This value does not depend on the choice of the fundamental unit of the field.*

Definition 21. *Let* I_f *denote a fractional ideal of* $\mathcal{O}_K$. *An element* $x \in I_f$ *is called* a minimum *if there is no other point* $y \in I_f$ *such as* $\log|\sigma_1(x)| > \log|\sigma_1(y)|$ *and* $\log|\sigma_2(x)| > \log|\sigma_2(y)|$.

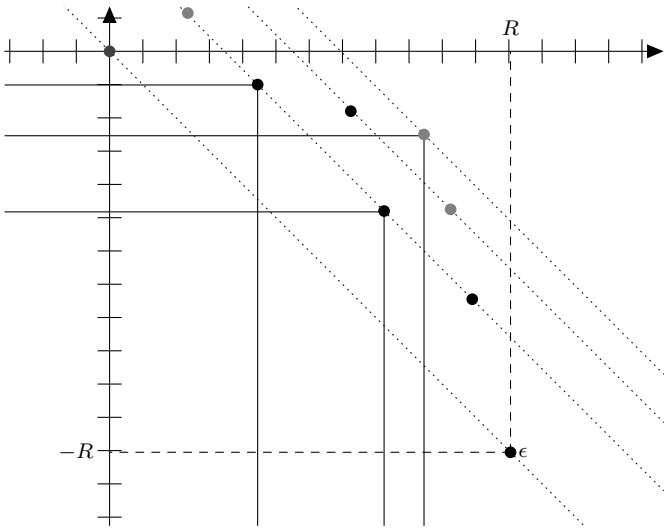

Figure 2: The points in black are the minimums.

Definition 22. *A fractional ideal* I *is said to be* reduced *if it contains* 1 *and if* 1 *is a minimum in it.*

The interest of minimal elements of an ideal occurs when one wants to browse the reduced ideals equivalent to another reduced ideal. We have the following proposition:

Proposition 23. *If* I *is a reduced ideal and* θ *is a minimal element in* I, *then* $\frac{1}{\theta}I$ *is a reduced ideal.*

Proof. Let's suppose that I is reduced and $\frac{1}{\theta}I$ is not. There is an element $x \in \frac{1}{\theta}I$ such that $|\sigma_1(x)| < 1$ and $|\sigma_2(x)| < 1$. By definition of $\frac{1}{\theta}I$, there exists $x' \in I$ such that $x = x'/\theta$. Then for $k = 1, 2, |\sigma_k(x')/\sigma_k(\theta)| < 1$ then $|\sigma_k(x')| < |\sigma_k(\theta)|$. This is a contradiction because θ is minimal in I.

□

The SQUFOF algorithm browses the different reduced ideals equivalent to a given fractional ideal I, this is done by browsing the minimums of I in a certain order.

Definition 24. *Two minimal elements* $\theta_1, \theta_2 \in I$ *such that* $|\sigma_1(\theta_1)| < |\sigma_1(\theta_2)|$ *are said to be* adjacent *if there is no other minimal element* $\eta \in I$ *such that* $|\sigma_1(\theta_1)| < |\sigma_1(\eta)| < |\sigma_1(\theta_2)|$.

2.4 Infrastructures of number fields with unit rank 1

With the previous definitions, one can see that there is a chain of adjacent minimal elements in a reduced fractional ideal I. H.C. Williams has shown in [9, Lemma 3.8] that in fact every minimal element of I is on this chain. The main part of the SQUFOF algorithm is the walk on this chain, there is an algorithm to do that in the general case: Algorithm 2.1.

Algorithm 2.1 Calculate the chain of minimal elements of a reduced fractional ideal

Require: I reduced fractional ideal of $\mathcal{O}_K$

$i \leftarrow 1$
$I_1 \leftarrow I$
$\Theta_1 \leftarrow 1$
loop
  Compute θ_i the minimum adjacent to 1 in I_i ▷ Reduction step
  $\Theta_{i+1} \leftarrow \Theta_i \cdot \theta_i^{-1}$
  $I_{i+1} \leftarrow \frac{1}{\theta_i} I_i$
  $i \leftarrow i + 1$
end loop

Proposition 25. *[3, Prop 2.12] The sequence $(\Theta_n)_{n\geq 1}$ is the chain of adjacent minimal elements of $\mathcal{O}_K$.*

Proposition 26. *If I_i is a reduced fractional ideal of $\mathcal{O}_K$, so is I_{i+1}.*

Proof. This proposition if a direct application of Proposition 23. □

Proposition 27. *[3, Prop 2.6] The sequence $(\theta_n)_{n\geq 1}$ is periodic, and the sequence $(\Theta_n)_{n\geq 1}$ is periodic modulo the fundamental unit of the field.*

Definition 28. *The* reduction *operator, denoted by ρ is the application of a step of the above loop. With this notation $(I_i)_{i\geq 1} = (\rho^{(i)}(I))_{i\geq 1}$.*

The difficulty of the algorithm holds in the reduction step. Even if this algorithm is generic for all the number fields of unit rank one, this step will differ in function of the type of field. This algorithm produce chains of reduced fractional ideals, and every reduced ideal equivalent (for $\sim$) to an ideal I is produced by the above algorithm in the sequence $(I_i)_{i\geq 1}$ because of the minimality of the θ_is. This structure is called the *infrastructure* of the class group of the number field.

In the next section I will explain see how the walk along this infrastructure in the field $\mathbb{Q}(\sqrt{D})$ enables us to find a divisor of D.

3 The SQUFOF algorithm

The SQUFOF (Shank's SQuare FOrm Factorization) algorithm was discovered by Shanks in 1975 and can compute a non-trivial factor of a RSA number D in time (under heuristics hypothesis) $\mathcal{O}(D^{\frac{1}{4}})$ [4, 8.7 p. 437]. This algorithm has the advantage that it uses only numbers less that $\sqrt{D}$, which allows it to be implemented on small calculators [4, 8.7 p. 438].

This algorithm can be studied with many points of view, including in terms of integer binary quadratic forms or in terms of continued fractions (which is the fastest implementation). In this report, in order to create the link between the different kinds of fields, I will describe this algorithm in term of walk on the infrastructure of a quadratic field.

3.1 Result on quadratic fields

Definition 29. *A* real quadratic field *is a field K of the form $\mathbb{Q}[X]/(X^2 - D)$ with D a squarefree positive integer.*

In this section I will denote by K a quadratic number field of discriminant D, with D a square free integer. Moreover, I will consider the ideals of $\mathcal{O}_K$ modulo the multiplicative action of $\mathbb{Q}$.

Theorem 30. *[4, Theorem 5.2.9] Every ideal $I \subseteq \mathcal{O}_K$ is of the form $a\mathbb{Z}+\frac{-b+\sqrt{D}}{2}\mathbb{Z}$ with $a \in \mathbb{Z}\setminus\{0\}, b \in \mathbb{Z}$ and $b^2 \equiv D \mod 4a$. Therefore any ideal of $\mathcal{O}_K$ can be represented by a triplet of integers (a,b,c) with the property that $b^2-4ac=D$. I will note $I \approx (a,b,c)$.*

Definition 31. *A triplet (a,b,c) with b^2-4ac is said to be* in reduced form *if $\left|\sqrt{D}-|2a|\right| < b < \sqrt{D}$. In this case, $a = \min(dI \cap \mathbb{Z}_{>0})$.*

Remark 32. *With this notation, $\mathcal{O}_K \approx (1, b, \frac{b^2-D}{4})$ where b is the greatest integer less than $\sqrt{D}$ such that $b^2 \equiv D \mod 4$.*

Since the ideals are studied modulo the action of $\mathbb{Q}$, any fractional ideal is represented by some triplet (a,b,c) too, refering to the triplet representing dI where $d = min\{n \in \mathbb{Z}_{>0}, nI \subset \mathcal{O}_K\}$.

This property enables us to give an explicit formula to the multiplication law of two fractional ideals. In this report, we will only need the formula for squaring.

Proposition 33. *[4, Lemma 5.4.5] If $I \approx (a,b,c)$ then $I^2 \approx (A,B,C)$ with*

$$A = \frac{a^2}{d^2}, B = b - \frac{2acv}{d}, C = \frac{B^2-D}{4A}$$

where $d = a \wedge b$ and u, v the coefficients such that $ua+vb=d$.

The principle of the SQUFOF algorithm holds on the following result:

Theorem 34. *If I is a fractional ideal such as $I^2 = \mathcal{O}_K$ (such ideal are called* ambiguous*) then $\min(I \cap \mathbb{Z}_{>0})$ divides D.*

Proof. I take a reduced triplet representing the ideal $I \approx (a,b,c)$. As $I^2 = \mathcal{O}_K$, by looking at the first number of the triplet, one can see that (with the same notation than in Proposition 33) $\frac{a^2}{d^2} = 1$ so $a \mid b$. Then $a \mid b^2 - 4ac = D$. □

Then our goal will be to find an ideal "square root" of $\mathcal{O}_K$ in order to find a non-trivial divisor of D. We will use for that another property.

Proposition 35. *[4, Prop 8.7.1] If there is a principal fractional ideal I such that there exists J with $J^2 = I$ then there is an ideal N on the reduction cycle of J such that $N^2 = \mathcal{O}_K$.*

Algorithm 3.1 SQUFOF algorithm

```
I ≈ (a,b,c) ← O_K                ▷ I associate the fractional ideals and their representation
while a is not a square or a is a square and (√a, b, c√a) is principal do
    I ← ρ(I)                     ▷ With the notation of the Def 28.
end while
I ≈ (a,b,c) ← (√a, b, c√a)
while I² ≠ O_K do
    I ← ρ(I)
end while
return a
```

The principle of the SQUFOF algorithm is to walk along the cycle starting from $\mathcal{O}_K$ to find a square ideal i.e. a fractional principal ideal I of the form J^2 where J is not principal, and then when one is found, we calculate J and we walk along its cycle until we find an ideal N whose square is equal to $\mathcal{O}_K$. The detection of a square ideal and the computation of its square root are very easy. An ideal is square if and only if its minimal integer is a square (which is easily tested by a comparison to the square of the integer part of its square root), and the square root of (a^2, B, C) is (a, B, aC) [4][8.7]. For a more precise description, see Algorithm 3.1.

3.2 The reduction step

We saw in the previous section that the most difficult part of the algorithm was the reduction step. The algorithm for the case of the quadratic field was described by Hugh C. Williams in [9]. I will associate every element $x \in K$ with its first real embedding, and denote by x' (I will talk about *conjugate element*) its second real embedding.

Algorithm 3.2 Calculate $\rho(I)$

Require: I reduced fractional ideal of $\mathcal{O}_K$

Let $(1, \mu)$ be a basis of I as $\mathbb{Z}$-module
$\nu \leftarrow \mu - \lfloor \mu' \rfloor$
if $\nu < -1/2$ **then**
    $\psi \leftarrow \nu + 1$
else
    $\psi \leftarrow \nu$
end if
return $\frac{1}{\psi} I$

Remark 36. *When the square root of a square ideal is calculated in SQUFOF, we are not assured to find a reduced ideal. But it was shown in [9, 4] that the application of this algorithm on a non-reduced ideal I will return a reduced ideal equivalent in a logarithmic number of step in $c = \min\{|x|, x \in I \setminus \{0\}\} \cap \mathbb{Z}$.*

4 Voronoï algorithm for pure cubic fields

To generalize the SQUFOF algorithm to complex cubic fields, the first logical step is to describe a reduction algorithm. I studied the reduction algorithm created by Voronoï during his thesis and mainly described in [5] (the original text was written in Russian). By lack of time, I focused on a particular case: the pure cubic fields.

Definition 37. *A* pure cubic field *is a field K of the form $\mathbb{Q}[X]/(X^3 - D)$ with D a cube-free integer.*

In order to simplify the notations and the calculus, I will identify the field K with its real embedding $\mathbb{Q}(\sqrt[3]{D})$. I will denote respectively by $x' \in \mathbb{C}$ and $x'' \in \mathbb{C}$ the *conjugate* elements of $x \in \mathbb{Q}(\sqrt[3]{D})$, that is to say respectively $\sigma_2(x)$ and $\sigma_3(x)$. One can notice that $x'x'' = |x'|^2 = |x''|^2$. In everything that follows, I will denote $\alpha = \sqrt[3]{D}$.

4.1 Positive binary quadratic form

Before the description of the algorithm, I have to introduce some classical results for positive binary quadratic forms.

Definition 38. *A* binary quadratic form *f is a quadratic homogeneous bivariate polynomial, in other terms a function of the form $f(X, Y) = aX^2 + 2bXY + cY^2$ with $a, b, c \in \mathbb{R}$. This form is said to be positive if $b^2 - 4ac < 0$ i.e. if $\forall X, Y \in \mathbb{R}^2 \setminus \{(0, 0)\}, f(X, Y) > 0$. I will denote the forms in simpler way: $f : X, Y \mapsto aX^2 + 2bXY + cY^2 = (a, b, c)$.*

The positive quadratic forms are deeply related to basis of vector in $\mathbb{R}^2$:

Proposition 39. *The positive binary quadratic form (PBQF) are associated bijectively with pairs of vectors of $\mathbb{R}^2$ up to rotation by the mapping $(\vec{u}, \vec{v}) \mapsto \|X\vec{u} + Y\vec{v}\|^2 = \|\vec{u}\|^2 X^2 + 2\langle \vec{u}, \vec{v} \rangle XY + \|\vec{v}\|^2 Y^2$.*

Proof. Follows immediately from the definition of the mapping. □

The change of basis of the space can be interpreted in terms of changing the positive binary quadratic form.

Proposition 40. *The group $GL_2(\mathbb{Z})$ acts on the right on the set of binary quadratic forms in the following way:*

$$f = (A, B, C), P = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in GL_2(\mathbb{Z})$$

$$f^P : X, Y \mapsto f\left(P \begin{pmatrix} X \\ Y \end{pmatrix}\right) = f(aX + bY, cX + dY).$$

If the form is positive, one can see this operation as the right action of P on $(\vec{u}, \vec{v})$. Indeed, some calculus gives us that the form associated with the pair $(a\vec{u} + c\vec{v}, b\vec{u} + d\vec{v})$ is

$$\begin{aligned} f^P(X,Y) &= (a^2 \|\vec{u}\|^2 + c^2 \|\vec{v}\|^2 + 2ac\langle\vec{u},\vec{v}\rangle)X^2 \\ &\quad +(b^2 \|\vec{u}\|^2 + d^2 \|\vec{v}\|^2 + 2bd\langle\vec{u},\vec{v}\rangle)Y^2 \\ &\quad +2XY(ab \|\vec{u}\|^2 + cd \|\vec{v}\|^2 + (ad+bc)\langle\vec{u},\vec{v}\rangle) \\ &= \|a\vec{u} + c\vec{v}\|^2 X^2 + 2\langle a\vec{u} + c\vec{v}, b\vec{u} + d\vec{v}\rangle XY + \|b\vec{u} + d\vec{v}\|^2 Y^2 \\ &= \|X(a\vec{u} + c\vec{v}) + Y(b\vec{u} + d\vec{v})\|^2 \end{aligned}$$

4.2 The Voronoï algorithm

The function d used in the Algorithm 4.1 is the following:

$$d\left(t + t'\alpha + t''\alpha^2\right) = t^2 - t't''D + (t''^2 D - tt')\alpha + (t'^2 - tt'')\alpha^2. \tag{1}$$

Direct computation shows that for $x \in K$, $d(x) = |\sigma_2(x)|^2$. One can find this algorithm very obscure at first glance. In this section I will explain all the steps in order to clarify it. The Voronoï algorithm can be interpreted in a geometric way.

I will represent a point $x = t + t'\alpha + t''\alpha^2, (t, t', t'') \in \mathbb{Q}^3$ by the $\mathbb{R}^3$ point:

$$Map(x) = (x, \Im(x'), \Re(x')) = \left(x, \frac{x' - x''}{2i}, \frac{x' + x''}{2}\right) = \left(t + t'\alpha + t''\alpha^2, \frac{\sqrt{3}}{2}(t'\alpha - t''\alpha^2), t + \frac{1}{2}(t'\alpha + t''\alpha^2)\right).$$

This map is linear and injective. In what follows I will identify $Map(x)$ and x. With this mapping, the ideal I is seen as a *lattice* of $\mathbb{R}^3$, that is to say a subset of $\mathbb{R}^3$ of the form $\mathbb{Z}\vec{b_1} + \mathbb{Z}\vec{b_2} + \mathbb{Z}\vec{b_3}$ with $(\vec{b_1}, \vec{b_2}, \vec{b_3})$ a basis of $\mathbb{R}^3$.

Proposition 41. *A point θ is minimal in I if and only if there is no other point of I in the* normed body of θ, *that is to say the set $C_\theta = \left\{(x, y, z), |x| \leq |\theta|, y^2 + z^2 \leq \Re(\theta')^2 + \Im(\theta')^2\right\}$ (see Fig 3).*

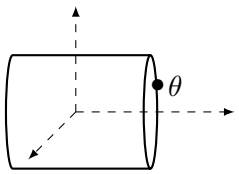

Figure 3: The normed body of θ

Proof. Let θ be a point of I.

$$\begin{aligned} \theta \text{ minimal in } I &\Leftrightarrow \nexists x \in I, \log|x| < \log|\theta| \text{ and } \log(|x'|^2) < \log(|\theta|^2) \\ &\Leftrightarrow \nexists x \in I, |x| < |\theta| \text{ and } |x|^2 < |\theta|^2 \\ &\Leftrightarrow \nexists x \in I, |x| < |\theta| \text{ and } \Re(x')^2 + \Im(x')^2 < \Re(\theta')^2 + \Im(\theta')^2 \\ &\Leftrightarrow \nexists x \in I, \mathrm{Map}(x) \in C_\theta \end{aligned}$$

□

At the beginning of the algorithm, one will want to be able to act only on the "non rational" part of the vector. For that one will project the points on the xy-plane in parallel to $0\vec{1} = (1, 0, 1)$. The

Algorithm 4.1 Voronoï algorithm. Calculate $\rho(I)$ for fractional ideals in pure cubic fields

Require: I reduced fractional ideal of $\mathcal{O}_K$

Let $(1, \phi, \psi)$ be a basis of I as $\mathbb{Z}$ module.
Let $\sigma, m, m', m'', n, n', n''$ be integers such as $\phi = \frac{m+m'\alpha+m''\alpha^2}{\sigma}$ and $\psi = \frac{n+n'\alpha+n''\alpha^2}{\sigma}$.
$A \leftarrow m'^2 + m'm''\alpha + m''^2\alpha^2$
$B \leftarrow m'n' + (m'n'' + m''n')\frac{\alpha}{2} + m''n''\alpha^2$
$C \leftarrow n'^2 + n'n''\alpha + n''^2\alpha^2$
while not ($B > 0$ and $A - B > 0$ and $C - B > 0$) **do** ▷ Reduction loop
    if $B < 0$ **then**
        $(A, B, C) \leftarrow (C, -B, A)$
    end if
    $P \leftarrow I_2$
    if $A < C$ **then**
        $m \leftarrow \lfloor \frac{B}{A} \rfloor$
        $P \leftarrow \begin{pmatrix} 1 & -m \\ 0 & 1 \end{pmatrix}$
    else if $C < A$ **then**
        $m \leftarrow \lfloor \frac{B}{C} \rfloor$
        $P \leftarrow \begin{pmatrix} 1 & 0 \\ -m & 1 \end{pmatrix}$
    end if
    $(A, B, C) \leftarrow (A, B, C)^P$
    $(\phi, \psi) \leftarrow (\phi, \psi)^P$
end while
Let $\sigma, m, m', m'', n, n', n''$ be integers like in the step 2 for the new ϕ and ψ.
$b \leftarrow \frac{m'-m''\alpha}{\sigma}$, $d \leftarrow \frac{n'-n''\alpha}{\sigma}$
for all $M \in \left[I_2, -I_2, \begin{pmatrix} 1 & 1 \\ -1 & 0 \end{pmatrix}, \begin{pmatrix} -1 & -1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & -1 \\ 1 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ -1 & -1 \end{pmatrix} \right]$ **do** ▷ Triangle choice loop
    $(x, y) \leftarrow (b, d) \cdot M$
    if $x > 0$ and $y < 0$ **then**
        $(b, d) \leftarrow (x, y)$
        $(\phi, \psi) \leftarrow (\phi, \psi) \cdot M$
        break
    end if
end for
$(\phi, \psi) \leftarrow (\phi - \lfloor \phi \rfloor, \psi - \lfloor \psi \rfloor)$
Let $\sigma, m, m', m'', n, n', n''$ be integers like in the step 2 for the new ϕ and ψ.
$a \leftarrow \frac{2m-m'\alpha-m''\alpha}{2\sigma}$, $c \leftarrow \frac{2n-n'\alpha-n''\alpha}{2\sigma}$
if $a < 1/2$ **then**
    $\theta_0 \leftarrow \phi$
else
    $\theta_0 \leftarrow 1 - \phi$
end if
if $c < 1/2$ **then**
    $\theta_1 \leftarrow \psi$
else
    $\theta_1 \leftarrow 1 - \psi$
end if
if $\psi - \phi > 0$ **then**
    if $c - a < 1/2$ **then**
        $\theta_2 \leftarrow \psi - \phi$
    else
        $\theta_2 \leftarrow 1 - (\psi - \phi)$
    end if
else

51: **if** $a - c < 1/2$ **then**
52: $\quad\theta_2 \leftarrow \phi - \psi$
53: **else**
54: $\quad\theta_2 \leftarrow 1 - (\phi - \psi)$
55: **end if**
56: **end if**
57: **if** $d(\theta_2) > d(\theta_0)$ and $d(\theta_2) > d(\theta_1)$ and $\phi + \psi < 1$ and $a + c < 1/2$ **then**
58: $\quad L \leftarrow [(d(\theta_0), \theta_0), (d(\theta_2), \theta_2), (d(\phi + \psi), \phi + \psi)]$ ▷ where d is defined in (1).
59: **else**
60: $\quad L \leftarrow [(d(\theta_0), \theta_0), (d(\theta_1), \theta_1), (d(\theta_2), \theta_2)]$
61: **end if**
62: **Sort** L
63: $\theta_g \leftarrow L[1]$, $\theta_h \leftarrow L[2]$
64: **return** $(1, \theta_h/\theta_g, 1/\theta_g)$ ▷ a $\mathbb{Z}$-basis of $\frac{1}{\theta_g} I$.

image of a point by this operation is called the *puncture* of the point. It can be seen as a rotation of our point of view aligning two points when they are different by a rational. The exact formula for $(t, t', t'') \in \mathbb{Q}^3$ is:

$$\text{Punct}(t + t'\alpha + t''\alpha^2) = \left(\frac{3}{2}(t'\alpha + t''\alpha^2), \frac{\sqrt{3}}{2}(t'\alpha - t''\alpha)\right).$$

One can see that this projection holds all the information about the non-rational part of a point.

The application of Punct on I provides us a 2 dimensional lattice where 1 is identified to the origin. I will call this lattice $\mathcal{L}$. The goal of the first part of the algorithm is to find the points adjacent to 0 in this two dimensional lattice. This goal is achieved by finding what Voronoï calls the reduced Zelling triangle of the lattice.

Definition 42. *The* reduced Zelling triangle *of a lattice $\mathcal{L}$ is the basis $(\vec{u}, \vec{v})$ of $\mathcal{L}$ such that the triangle formed with the origin is an acute triangle containing the negative x axis (see Fig 5).*

Calculus shows that the form (A, B, C) is the PBQF associated with the vectors $\text{Punct}(\phi)$ and $\text{Punct}(\psi)$ divided by $\frac{3\alpha^2}{\sigma^2}$. Finding the reduced Zelling triangle is done by reducing (A, B, C) in the reduction loop (Line 6 Algorithm 4.1). This loop is a variant of the Euclid algorithm, for the reduction of positive definite forms.

Proposition 43. *The fact that (A, B, C) is the PBQF associated with the vectors* $\text{Punct}(\phi)$ *and* $\text{Punct}(\psi)$ *is an invariant of the reduction loop.*

Proof. This proposition is a direct application of Proposition 40, plus the fact that the map $x \mapsto \text{Punct}(x)$ is linear. □

Proposition 44. *At the end of the reduction loop, the triangle $0\phi\psi$ is an acute triangle.*

Proof. This proposition is the reformulation of the conditions for ending the loop (line 6 Algorithm 4.1).

- $B > 0 \Leftrightarrow \langle \phi, \psi \rangle > 0 \Leftrightarrow \widehat{\phi 0 \psi} \in [-\pi, \pi]$
- $A - B > 0 \Leftrightarrow \langle \phi, \phi \rangle - \langle \phi, \psi \rangle > 0 \Leftrightarrow \langle \phi, \phi - \psi \rangle > 0 \Leftrightarrow \widehat{\phi 0(\phi - \psi)} \in [-\pi, \pi]$
- $C - B > 0 \Leftrightarrow \langle \psi, \psi \rangle - \langle \phi, \psi \rangle > 0 \Leftrightarrow \langle \psi, \psi - \phi \rangle > 0 \Leftrightarrow \widehat{\psi 0(\phi - \psi)} \in [-\pi, \pi]$

□

So the only thing remaining to show to see that our reduction loop really finds the reduced Zelling triangle of the lattice is to know if its ends. In fact, this loop ends very shortly.

Proposition 45. *The reduction loop ends in at most* $2 + \left\lceil \log\left(\frac{A}{\sqrt{|B^2 - 4AC|}}\right)\right\rceil$ *steps.*

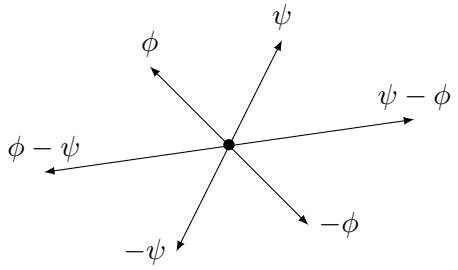

Figure 4: The Zelling hexagon of the lattice

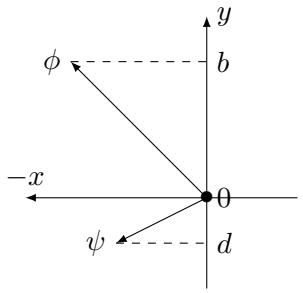

Figure 5: The reduced Zelling triangle

Proof. The reduction loops is just the Euclid's algorithm for the reduction of positive defined forms, which is described in a slightly different way in [4]. The proof of the complexity of the algorithm should be found in [4, Prop 5.4.3].

□

By construction, the numbers b and d are the y-coordinate of the puncture of ϕ and ψ (divided by a positive number). The triangle choice loop (Line 23 Algorithm 4.1) chooses the triangle which contains the negative x-axis by testing the six acute triangles of the Zelling hexagon of the projection of I.

The interest in finding the reduced Zelling triangle of the projection of I holds in the following theorem:

Theorem 46. *[5, §60 p. 276] If one denotes by* $(1,0)$ *and* $(0,1)$ *the vertices of the Zelling triangle of the projection of the reduced ideal* I*, then the puncture of the minimum of* I *adjacent to* 1 *is one of the following points :* $(1,0)$, $(0,1)$, $(-1,0)$, $(0,-1)$, $(1,-1)$, $(-1,1)$, $(1,1)$.

Once one has found the reduced Zelling triangle, s-he has found the seven possible "class" of points where the minimal point adjacent to 1 is in I. Necessarily, the minimal point adjacent to 1 is in $[-1,1] \setminus \{0\}$ because it will have a negative abscissa in the logarithmic embedding. Since -1 is a root of unity, one can choose the minimal point to be in $[0,1]$. Then s-he has restrained the set of possible minimal points to the points such that their puncture have their coordinates described in theorem 46 and are in $[0,1]$. Those points are obtained by setting $\phi \leftarrow \phi - \lfloor \phi \rfloor$, $\psi \leftarrow \psi - \lfloor \psi \rfloor$ and by the following transformations, assuming that $\phi > \psi$ (the symmetric case is done by exchanging $\phi - \psi$ by $\psi - \phi$):

- $\theta_0^{(1)} = \phi$
- $\theta_1^{(1)} = \psi$
- $\theta_2^{(1)} = \phi - \psi$
- $\theta_2^{(3)} = \phi + \psi$.
- $\theta_0^{(2)} = 1 - \phi$
- $\theta_1^{(2)} = 1 - \psi$
- $\theta_2^{(2)} = 1 - (\phi - \psi)$

Theorem 47. *[5, §60 p. 276] The minimum point adjacent to* 1 *in* I *is the closest point with the puncture described in the Definition 42 to the axis* x.

The last part of the algorithm discriminates the points in order to find the closest to the x-axis by doing few comparisons. This si done by compairing their coordinates. It would be possible to just order the seven points below in increasing d but the function d can be long to compute as it can deal

with big integers. The reader can look at [5, §61 p 285] for further details about the discrimination of the $\theta_i^{(j)}$.

5 Implementation and perspectives

5.1 Implementations

All the code I produced during my internship is at disposition at the address `https://gitlab.aliens-lyon.fr/jfelderh` under LGPL.

During my internship, I realized an implementation of SQUFOF in C++, with the formalism of binary quadratic form. This implementation achieves to find the factor 52562646845771 of the number 1342894401046902108489965690 51 (97 bits) in 15s.

I implemented the algorithm of Voronoï in the Magma programming language which has the advantage of containing all the mathematical primitives needed: fractional ideals, basis, exact representation of rational, etc... The main difficulty of this task was that if the reduction algorithm is well documented in the case of real quadratic fields, this is not the case for pure cubic fields. I only found one reference about the Voronoï algorithm ([5, §61]), and there were several mistakes (mainly typographic errors). The main work done during the internship was to find some bibliography about this algorithm (which was not so easy because the original book and thesis are written in Russian) and to understand its geometric significance.

The algorithm of Voronoï has been implemented in 1976 for [2]. This article presents a table of the number of steps of the Voronoï algorithm to calculate the regulator of pure cubic fields in function of the integer D in $\mathbb{Q}[X]/(X^3 - D)$. I produced a similar table (see Fig. 6 in appendix), but the results I observed were a little bit different than the results of [2]. As I can't have access to their implementation, this is hard to say why the results I had are different than theirs.

I implementted SQUFOF and Voronoï's algorithm in a way which does not require floating-point numbers and precision estimates: all the computations are done on either integers or rational numbers (using the Magma backend for Voronoï and the c++ library GMP for SQUFOF). In the case of the Voronoï algorithm, I had to rewrite some primitives in order to keep rational numbers. I implemented the computation of the floor part of the cubic root of a number and positivity test on algebraic cubic numbers. The only approximations used are for plotting the numbers with the logarithmic embedding and for integer parts of squareroots in SQUFOF.

5.2 Experimental results of my Voronoï algorithm implementation

Fig. 6 represents the number of step made and the time took by my implementation of the Voronoï algorithm to find the regulator of the pure cubic field $\mathbb{Q}(\sqrt[3]{D})$. I only represent the value of D whose associated pure cubic fields have a regulator greater than all the previous ones.

This table is similar to the one in [2], but one can see some differences on the number of steps done by the algorithm. For example, in [2], for $D = 1721$, they calculated 3320 steps, when with my implementation I calculated 3300 or for $D = 8429$, they had 15481 steps when I have 15446. This difference is small but hard to explain because the Voronoï algorithm is a deterministic algorithm. As in the general case my algorithm does less reduction steps, there are two possibilities: either my implementation misses some minimal elements or the implementation of [2] does not always choose a minimal element: this is an error that occured many times during the debugging phase of my implementation of the algorithm of Voronoï. As I didn't find the code used in [2], I can't compare it to my implementation in order to find differences.

5.3 Perspectives

The initial goal of this internship was to create a "cubic SQUFOF". There are four main parts in the construction of the SQUFOF algorithm : firstly one must be able to walk among the infrastructure of the class group of our field, secondly s-he must be able to detect if an ideal I in the unit cycle is the square of another ideal J, and to compute this ideal J if this is the case. The third part is that one

D	Steps	Time (s)	D	Steps	Time (s)	D	Steps	Time (s)
2	1	0.01	2283	3950	8.46	28517	50553	778.05
3	3	0	2927	4073	8.99	29063	56565	1001.67
5	4	0.01	3543	4067	9.05	32213	63460	1314.17
6	5	0	3557	5398	12.89	34607	66722	1493.3
15	7	0.01	3821	6393	16.08	36821	67055	1508.41
23	21	0.03	3921	7983	21.97	38039	70493	1675.93
29	35	0.06	4523	8519	24.49	39129	77024	2071.47
41	50	0.08	5153	8571	25.15	39521	81380	2368.06
69	99	0.17	5433	10662	35.25	43863	89669	2948.7
137	119	0.2	6999	12314	44.71	54293	95207	3444.97
167	206	0.36	8093	13533	53.14	55901	101895	4038.38
227	204	0.36	8429	15446	66.2	56993	103747	4246.08
239	392	0.71	10037	15939	71.32	60887	115603	5458.2
411	487	0.89	10067	16314	73.33	62889	119286	5886.53
419	638	1.17	11621	22862	140.3	66431	132697	7547.25
447	714	1.32	14897	25182	173.78	67829	136163	8056.18
573	878	1.64	15261	25147	174.44	72227	136460	8106.4
771	1197	2.26	15527	27895	212.95	72617	148588	9875.83
951	1361	2.6	17669	28345	223.93	76259	152381	10441.44
1163	1589	3.13	19391	38281	413.98	84629	159499	11645.49
1301	2319	4.62	21839	41933	513.08	88661	169395	13526.99
1721	3300	6.82	22469	42593	527.62			
2003	3236	6.74	26417	50223	771.69			

Figure 6: Number of steps of my implementation of the algorithm of Voronoï for some values of D

must be able to know if there is always an ambiguous ideal in the cycle of a square root of a principal ideal, then to know if the fact that an ideal is ambiguous contains information on the factorization of the discriminant of the field.

The Voronoï algorithm is the algorithm used to walk among the infrastructure, but in order to create a cubic SQUFOF, there are 4 algorithmic questions left to answer:

- How to detect if a cubic fractional ideal is a square?
- If this is the case, how to calculate its square root?
- Is there always an ambiguous ideal in the reduction cycle of the square root of a principal ideal?
- Are information about the factorization of D contained in the ambiguous ideals of $\mathcal{O}_{\mathbb{Q}(\sqrt[3]{D})}$?

References

[1] S. Bai, P. Gaudry, A. Kruppa, E. Thomé, and P. Zimmermann. Factorisation of rsa-220 with cado-nfs. 2016.

[2] P. Barrucand, H. C. Williams, and L. Baniuk. A computational technique for determining the class number of a pure cubic field. *Mathematics of Computation*, 30(134):312, 1976.

[3] J. Buchmann and H. C. Williams. On the infrastructure of the principal ideal class of an algebraic number field of unit rank one. *Mathematics of Computation*, 50(182):569, 1988.

[4] H. Cohen. *A course in computational algebraic number theory*. Springer, 1995.

[5] B. N. Delone and D. K. Faddeev. *The theory of irrationalities of the third degree*. American Mathematical Soc., 1978.

[6] D. Shanks. Analysis and improvement of the continued fraction method of factorization. *American Mathematical Society Notices*, 22:1, 1975.

[7] D. Shanks. Squfof notes. *Unpublished manuscript*, 30, 2004. transcribed by S. McMath.

[8] H. Williams, G. Cormack, and E. Seah. Calculation of the regulator of a pure cubic field. *Mathematics of Computation*, 34(150):567–611, 1980.

[9] H. C. Williams. Continued fractions and number-theoretic computations. *Rocky Mountain Journal of Mathematics*, 15(2):621–656, 1985.

6 Appendix

6.1 Context of my internship

I realized my internship in the team CARAMBA of the LORIA under the supervision of Pierre-Jean Spaenlehauer. The LORIA (laboratoire lorrain de recherche en informatique et ses applications) is located in the campus science and technology of the Université de Lorraine.

The team CARAMBA, led by Emmanuel Thomé, has two main research themes : the general number field sieve algorithm (the most efficient algorithm know today for factoring integers) and the study of algebraic curves for cryptography. They also study computer algebra problematics, such as solving polynomial systems and Gröebner basis. It is composed of five INRIA researchers, one CNRS researcher, one professor, an assistant professor, three PhD students and two postdoctoral fellows. One of their subject of research is the improvement of *CADO-NFS*, an implementation of the general number field under licence LGPL-2.1+ which they used to factorize RSA-220 (220 decimal digits, 729 bits) in 2016 [1].

6.2 Code listing

6.2.1 SQUFOF algorithm (c++)

```
#include <iostream>
#include <cmath>
#include "BinQuadForm.hpp"

using namespace std;

bool isSquare(Integer n){
  if(n <= 0)
    return false;
  Integer a = (Integer)sqrt(n);
  return a*a == n;
}

bool isIn(const Set& Q, Integer n){
  return Q.find(n) != Q.end();
}

int main()
{
  Integer n;
  cin >> n;

  Integer factor = 1;
  Integer k = 0;

  while(factor == 1 or k%factor == 0){
    k++;
    initialize(n*k);

    BinQuadForm f;

    Set Q;
    cout << "Factorising D=" << D << endl;
    cout << "Unit form cycle" << endl;
      do{
  if(f.a != 1 and f.a*f.a <= sqrtDInt){
    Q.insert(f.a);
  }

  f = reduce(f);
      }while( not (isSquare(f.a) and !isIn(Q, (Integer)sqrt(f.a)) ));

    if(f.a == 1){
```

```
      cerr << "Fail !" << endl;
      continue;
    }

    cout << endl;

    BinQuadForm g((Integer)sqrt(f.a), -f.b);

    cout << "reduction" << endl;

    while(not g.reduced()){
      g = reduce(g);
    }

    cout << "Cycle" << endl;

    Integer b1 = g.b;

    do{
      b1 = g.b;
      g = reduce(g);
    }while(b1 != g.b);

    if(g.a % 2 == 0){
      factor = abs(g.a/2);
    }else{
      factor = abs(g.a);
    }

    Integer fact = factor%k==0?factor/k:factor;

    cout << "Factor of D = " << n << " : " << fact << endl << "Error ? " << n%fact << endl;

  }

  return 0;
}
```

```
#ifndef BINQUAD_HPP
#define BINQUAD_HPP

#include <cmath>
#include <ostream>
#include <set>
#include <gmpxx.h>

using Integer = mpz_class;
using Scalar = mpf_class;
using Set = std::set<Integer>;

extern Integer N;

inline Integer discriminant(Integer n){ // n is supposed squarefree
  if(n % 4 == 1)
    return n;
  else
    return 4*n;
}

extern Integer D;//number to reduce

extern Scalar sqrtD; // approx of sqrt(D)
extern Integer sqrtDInt; // integer approx of sqrt(D)

inline void initialize(Integer n){
  N = n;
```

```
  D = discriminant(n);
  sqrtD = sqrt(D);
  sqrtDInt = (Integer)sqrtD;
}

Integer getC(Integer a, Integer b);

struct BinQuadForm{
  BinQuadForm(); // the unit binquadform

  BinQuadForm(Integer _a, Integer _b);

  Integer disc() const;

  bool reduced() const;

  Integer a, b, c;
};


std::ostream& operator<<(std::ostream&, const BinQuadForm&);

BinQuadForm reduce(const BinQuadForm & f);

#endif
```

```
#include "BinQuadForm.hpp"

using namespace std;

Integer N = 0;
Integer D = 0;
Scalar sqrtD = 0;
Integer sqrtDInt = 0;

Integer getC(Integer a, Integer b){
  return (Integer)((b*b-D)/(4*a));
}

BinQuadForm::BinQuadForm(){ // the unit binquadform
  a = 1;
  if(D%4 == 1){
    b = 2*(Integer)((sqrtDInt-1)/2) +1;
  }else{
    b = 2*(Integer)(sqrtDInt/2);
  }


  c = getC(a, b);
}

BinQuadForm::BinQuadForm(Integer _a, Integer _b) : a(_a), b(_b), c(getC(_a, _b)){
}

Integer BinQuadForm::disc() const{
  return b*b-4*a*c;
}

bool BinQuadForm::reduced() const{
  return (abs(sqrtD - 2*abs(a)) <= b) and (b <= sqrtD);
}

BinQuadForm reduce(const BinQuadForm & f){
  BinQuadForm g;
  Integer r = (-f.b) % (2*abs(f.c));
  if(abs(f.c) > sqrt(D)){
```

```
    if(r > abs(f.c))
      r -= 2*f.c;
  }else{

    if(r <= sqrtD - 2*f.c){
      r += ceil(((sqrtD - 2*f.c - r)/(2*abs(f.c))))*2*abs(f.c);
    }

    if(r > sqrt(D)){
      r -= 2*abs(f.c)*ceil((r-sqrtD)/(2*abs(f.c)));
    }

    while(r <= sqrtD - 2*f.c){
      r += 2*abs(f.c);
    }
    while(r > sqrtD){
      r -= 2*abs(f.c);
    }
  }

  return BinQuadForm(f.c, r);
}

std::ostream& operator<<(std::ostream& stream, const BinQuadForm& f){
  stream << f.a << " " << f.b << " " << f.c;
  return stream;
}
```

6.2.2 The Voronoï algorithm (Magma)

```
D := 26417;

P := [-D, 0, 0, 1];

prec := 10000;
precOut := 10;

Poly<x> := PolynomialAlgebra(Rationals());

Q<z> := NumberField(Polynomial(P));

R := IntegerRing(Q);

function RealVal(x)
    return RealField(prec)!Conjugates(x)[1];
end function;

//returns true if x1 < D^1/3 < x2 for a polynomial a, b, c with discriminant discr
function TestRac(a, b, discr)
    delta := 8*D*a^3 + b^3 + 3*b*discr;
    return (delta/(3*b^2 + discr))^2 lt discr;
end function;

// exact test if x gt 0 for x a cubic number
function GtZero(x)
    if Q!x eq Q!0 then
  return false;
    end if;

    L := Eltseq(Q!x);

    //polynomial associated to x L[1]+L[2]x+L[3]x^2
    S := elt<Poly | Eltseq(Q!x)>;

    // if this is a constant, easy
    if Degree(S) eq 0 then
```

```
  return L[1] gt 0;
    end if;

     // si this is linear, easy too
    if Degree(S) eq 1 then
  if L[2] gt 0 then
      return D gt -(L[1]/L[2])^3;
  else
      return D lt -(L[1]/L[2])^3;
  end if;
    end if;
    // at this point, the degree of the polynomial is 2
    discr := Discriminant(S);


    // if the poly is defined positive or negative, that is easy
    if discr lt 0 then
  return LeadingCoefficient(S) gt 0;
    end if;

    // if the discriminant has one root, we just look if the number is the root of it
    if discr eq 0 then
  if x eq Q!(-L[2]/(2*L[3])) then
      return false;
  else
      return LeadingCoefficient(S) gt 0;
  end if;
    end if;
    //else, we look if it is between the roots
    if LeadingCoefficient(S) gt 0 then
  return not TestRac(L[3], L[2], discr);
    else
  return TestRac(L[3], L[2], discr);
    end if;
end function;

//calculate floor(alpha*sqrt3(Delt)) without loss of precision with the newton method
function FloorSq3(alpha, Delt)
    if alpha eq 0 then
  return 0;
    end if;

    pr := Floor(Abs(Log(Abs(alpha*Delt)))/Log(10))+5;
    u := RealField(pr)!13;
    u1 := RealField(pr)!0;
    while Abs(u-u1) gt 0.1 do
  u1 := RealField(pr)!u;
  u := u1 - (u1^3-Delt*alpha^3)/(3*u1^2);
    end while;

    N:= Floor(u);
    return Floor(u);
end function;


function VAbs(x)
    if GtZero(x) then
  return x;
    else
  return -x;
    end if;
end function;


// floor without approximations
function PartEntiere(x)
```

```
    if GtZero(1-VAbs(x)) then
  if GtZero(x) or x eq Q!0 then
      return Q!0;
  else
      return Q!-1;
  end if;
    end if;
    L := Eltseq(Q!x);

    n1 := FloorSq3(L[3], (D)^2);
    n2 := FloorSq3(L[2], D);
    n3 := Floor(RationalField()!L[1]);
    N := n1+n2+n3-2;
    //There can be a difference
    while not((GtZero(x-Q!N) or x eq Q!N) and GtZero(Q!(N+1)-x)) do
  N +:= 1;
    end while;
    assert((GtZero(x-Q!N) or x eq Q!N) and GtZero(Q!(N+1)-x));
    return N;
end function;

//log embedding
procedure plot(x)
    C := Conjugates(x);
    print RealField(precOut)!Log(Abs(RealField(prec)!C[1])), RealField(precOut)!(2*Log(RealField(prec)!Abs(Comple
end procedure;

function swap(x, y)
    return y, x;
end function;

//squre of the distance to the x-axis
function rho2(x)
    t := Eltseq(Q!x);
    return t[1]^2 - t[2]*t[3]*D + (D*t[3]^2-t[1]*t[2])*z + (t[2]^2 - t[3]*t[1])*z^2;
end function;


function Comp(x, y)
    if x[1] eq y[1] then
  return 0;
    end if;
    if GtZero(x[1]-y[1]) then
  return 1;
    else
  return -1;
    end if;
end function;


I := ideal<R|1>;

B := Basis(I);


phi := B[2];
psi := B[3];

minAct := R!1;
premierTour := true;

compt := 0;

// While we haven't found an unit (exact precision)
while Norm(minAct) ne 1 or premierTour do
    compt +:= 1;
```

```
    premierTour := false;
    //plot(minAct);
    print compt;
    sigma := Denominator(I);

    phiS := Eltseq(phi);
    psiS := Eltseq(psi);

    for a in phiS do
  sigma *:= Denominator(sigma*a);
    end for;

    for a in psiS do
  sigma *:= Denominator(sigma*a);
    end for;

    for i in [1, 2, 3] do
  psiS[i] *:= sigma;
  phiS[i] *:= sigma;
    end for;

    if phiS[1]*psiS[2]-phiS[2]*psiS[1] lt 0 then
  phi, psi := swap(phi, psi);
  phiS, psiS := swap(phiS, psiS);
    end if;

    A := phiS[2]^2+phiS[2]*phiS[3]*z+(z^2)*(phiS[3]^2);
    B := phiS[2]*psiS[2] + (phiS[2]*psiS[3]+phiS[3]*psiS[2])*z/2 + phiS[3]*psiS[3]*(z^2);
    C := psiS[2]^2+psiS[2]*psiS[3]*z+(z^2)*(psiS[3]^2);


    while not (GtZero(B) and GtZero(A-B) and GtZero(C-B)) do
  if not GtZero(B) then
      A, C := swap(A, C);
      B := -B;
      phi, psi := swap(phi, psi);
      phi *:= -1;
  end if;
  if GtZero(C-A) then
      m := -R!PartEntiere(B/A);
      B1 := B;
      B := B + m*A;
      C := C + 2*m*B1 + A*m^2;
      //modify C -> modify psi
      psi := psi + m*phi;
  else
      m := -R!PartEntiere(B/C);
      B1 := B;
      B := B + m*C;
      A := A + 2*m*B1 + C*m^2;
      //modify A -> modify phi
      phi := phi + m*psi;
  end if;
    end while;

    assert(I eq ideal<R| 1, phi, psi>);


    phiS := Eltseq(sigma*phi);
    psiS := Eltseq(sigma*psi);

    b := (phiS[2]-phiS[3]*z)/sigma;
    d := (psiS[2]-psiS[3]*z)/sigma;

    l := [Matrix(Q, [[1, 0], [0, 1]]), Matrix(Q, [[-1, 0], [0, -1]]), Matrix([[1, 1], [-1, 0]]),
    Matrix([[-1, -1], [1, 0]]), Matrix([[0, -1], [1, 1]]), Matrix([[0, 1], [-1, -1]])];
```

```
    //warning ! difference between the article and this !
    vect := Vector([Q!phi, Q!psi]);
    vVal := Vector([b, d]);
    q := 0;
    for M in l do
  v := vVal*M;
  x := v[1];
  y := v[2];
  if GtZero(x) and not GtZero(y) then
      v := vect*M;
      phi := v[1];
      psi := v[2];
      q +:=1;
  end if;
    end for;

    assert(q eq 1);

    psi -:= PartEntiere(psi);
    phi -:= PartEntiere(phi);

    phiS := Eltseq(sigma*phi);
    psiS := Eltseq(sigma*psi);

    assert(I eq ideal<R| 1, phi, psi>);

    a := ( ( 2*(phiS[1]) - z*phiS[2] - phiS[3]*z^2 )/(2*sigma));
    c := ( ( 2*(psiS[1]) - z*psiS[2] - psiS[3]*z^2 )/(2*sigma));

    if GtZero(1/2 - a) then
  theta0 := phi;
    else
  theta0 := 1-phi;
    end if;

    if GtZero(1/2 - c) then
  theta1 := psi;
    else
  theta1 := 1-psi;
    end if;

    if GtZero(psi-phi) then
  if GtZero(1/2-(c-a)) then
      theta2 := psi-phi;
  else
      theta2 := 1-(psi-phi);
  end if;
    else
  if GtZero(1/2-(a-c)) then
      theta2 := phi-psi;
  else
      theta2 := 1-(phi-psi);
  end if;
    end if;

    r0 := rho2(theta0);
    r1 := rho2(theta1);
    r2 := rho2(theta2);

    if GtZero(r2-r0) and GtZero(r2-r1) and
       GtZero(1-(phi+psi)) and GtZero(1/2-(a+c)) then
  L := [<r0, theta0, 0>, <r2, theta2, 2>, <rho2((phi+psi)), phi+psi, 3>];
    else
  L := [<r0, theta0, 0>, <r1, theta1, 1>, <r2, theta2, 2>];
    end if;
    L := Sort(L, Comp);
```

```
301
302     thetag := L[1][2];
303     thetah := L[2][2];
304
305     minAct /:= thetag;
306     assert(I eq ideal<R| 1, thetag, thetah>);
307
308     I := I/thetag;
309     phi := thetah/thetag;
310     psi := 1/thetag;
311 end while;
312
313 print "cycle size : ", compt;
```